THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Enterprise Modelling: A Declarative Approach for FBPML

Citation for published version:
Chen-Burger, Y-H, Tate, A & Robertson, D 2002, Enterprise Modelling: A Declarative Approach for FBPML. in ECAI 2002 Workshop #5 on Knowledge Management and Organizational Memories.

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
ECAI 2002 Workshop #5 on Knowledge Management and Organizational Memories

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 18. Nov. 2019

Enterprise Modelling: A Declarative Approach for *FBPML*

Yun-Heh Chen-Burger[1], Austin Tate[1], Dave Robertson[2]

[1] **AIAI, The University of Edinburgh**
80 South Bridge, Edinburgh, UK
email: jessicac@aiai.ed.ac.uk a.tate@ed.ac.uk

[2] **Centre for Intelligent Systems and their Applications**
The University of Edinburgh, UK
email: dr@dai.ed.ac.uk

Abstract. Enterprise Modelling (EM) methods are well-recognised for their value in describing complex, informal domains in an organised structure. EM methods are used in practice, particularly during the early stages of software system development, e.g. during the phase of business requirements elicitation. The built model, however, has not always provided direct input to software system development. Despite the provision of adequate training to understand and use EM methods, informality is often seen in enterprise models and presents a major obstacle. This paper focuses on one type of EM methods: business process modelling (BPM) methods. We advocate the use of a BPM language within a three-layer framework. The BPM language merges two main and complimentary business process representations, IDEF3 and PSL, to introduce a Fundamental Business Process Modelling Language (FBPML) that is designed for simplicity of use and under-pinned by rich formality that may be used directly to support software and workflow system development.

Key-words Business Process Modelling, IDEF3, PSL, Workflow Management, Business Modelling, BSDM, Formal Method, Enterprise Modelling, Collaborative (Web-based) Knowledge Management.

1 Introduction - The Gap

Enterprise modelling (EM) methods are well-recognised for their value in organising and describing a complex, informal domain in a more precise semi-formal structure that is intended for more objective understanding and analysis. Example EM methods are business modelling method, business modelling of IBM's BSDM (Business System Development Method) [13], process modelling method, IDEF0[18], IDEF3[17], PSL[21], RAD[19], RACD[3], CommonKADS Communication Model Language (CML)[26], organisational modelling, Ordit[7] Ulrich[10], capability modelling, [22] and (Enterprise) Ontology [23], [25], [9].

Despite their use, Enterprise Models have not always provided direct input for software system development. Obstacles include the necessary training required for users to learn conceptual modelling in general as well as the specific techniques required for the specific method applied. Generic knowledge acquisition techniques are also needed to elicit knowledge from the application domain. One other main obstacle is the lack of direct mapping from EM methods to software system development. Since EM methods are normally described at higher levels of abstraction which are independent of implementation issues, EM methods are often used merely as a description and analysis tool of the application domain. However, as EM methods often describe requirements from the business side, as opposed to from the technical side, the built Enterprise Models are natural candidates to provide a "blueprint" for business requirements when building software systems.

Figure 1 illustrates the gap that exists between Enterprise Models and common software systems built for organisations. It also proposes three possible means, all of them based on formal methods, *Quality Assurance, Mapping of Data Structure and Workflow System*, to bridge the gap by providing direct mappings between Enterprise Models and designing and building of software systems.

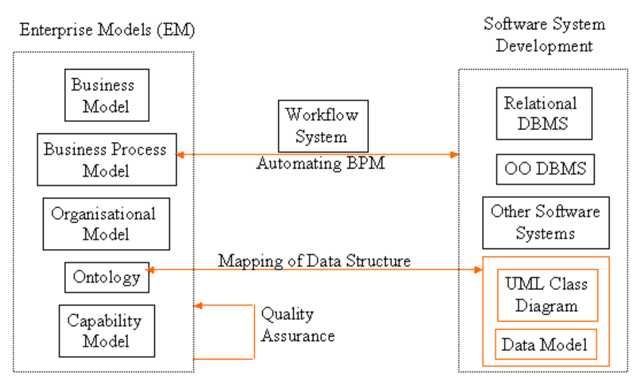


Figure 1. Bridging Gap between Enterprise Models and Software Systems

Formal methods may be used in various ways to facilitate communication between modellers and users of models, e.g. to make tacit information explicit and present it in different (maybe less technical and/or more familiar) forms, or to provide simulation functionalities to allow the reader to run through possible user scenarios in a state machine[2][20][11]. Automatic support such as knowledge sharing and inconsistency checking between different Enterprise Models, when a set of EM has been used, may also be done based on one commonly shared ontology [3]. The automatic support helps the modeller and user of the model understand a model in depth, therefore enhances their ability in error detection and model refinement. As a result, quality of the built models is improved. The refinement process based on computing support is indicated by the “Quality Assurance” arrow in the figure. Another way to bridge the gap is to provide a means to transfer data and knowledge that are held in the EM, particularly in an ontology, to software systems. This may be done by mapping an ontology to ER (Entity-Relational) Model (for Relational Databases) or to Class Diagram (for Object-Oriented Databases) or other types of data structures. This is indicated by the “Mapping of Data Structure” arrow.

This paper focuses on one type of EM method: Business Process Modelling (BPM) Method. One direct and obvious way to make use of BPM methods and to provide a direct input to software systems is to build a workflow system that is based on a business process model[8]. A definition of workflow, that is given by the Workflow Management Coalition, that describes its relationship with a business process is given below:

> *“The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.”[8]*

Although the above approach seems obvious, in practice not all workflow systems have received the full benefit from business process modelling. The BPM approach towards building a workflow system is a recent and gradual approach over the past two years. This is different from the first generation workflow systems where BPM was not used[12]. The reasons for such phenomena are the lack of training and understanding of BPM methods and how they may be applied in an organisation. The business process when it is used often does not separate business and implementation logic, and hence, the resulting workflow system is not flexible in reaction to the dynamic and volatile environment within which the workflow system operates.

Last but not least, while BPM methods are normally described at a higher level of abstraction that enables flexibility for implementation, they do not provide sufficient details of additional information that must be included for process enactment. It is therefore beneficial to provide a means that maintains the flexibility of higher level descriptions, while at the same time providing sufficient information and a mechanism to carry out workflow[14].

This paper proposes a layered business process modelling approach that aims to lessen the above problems, therefore narrowing the gap. The paper also describes the design of FBPML (Fundamental Business Process Modelling Language) and how business processes based on it may be mapped to a visualisation of dynamic states of a workflow system in a collaborative enterprise environment.

2 An Ontology based Three-Layer BPM Framework

Figure 2 describes a layered business process modelling framework which provides the means to allow higher level business processes, objectives and policies to be carried forward and realised in the actual implementation of software (and manual) systems. The upper two levels of the framework describe business operations at a higher level of abstraction; the lower level of the framework describes how these business operations may be implemented in a software system. In this framework, design rationale of a software system is based on a company’s objectives, hence the corresponding software system can be traced back to the initial business requirements and justified. Both of these enable the system to be coherent with the overall business aims.

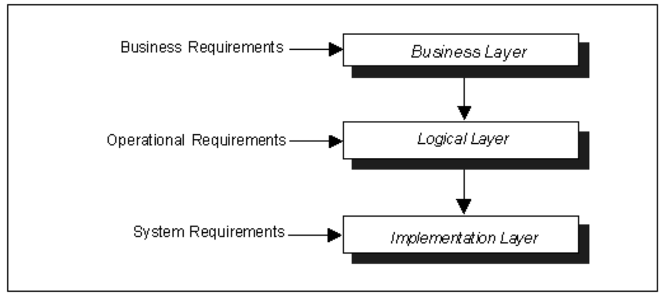


Figure 2. A Three-Layer Business Process Modelling Approach

The first layer, Business Layer, describes business requirements of an organisation, processes that are to be carried out by the organisation and information used by these processes. Information stored in this layer are higher level descriptions that may be written in informal or semi-formal documents. Examples are source data files, mission and organisation goal statements, business plans, and summary and vision of business operations. In this layer, information that is consolidated, such as business policies, longer lasting organisational structure and business-level decisions that are used as guidelines for developing business process models, is in general robust against change of technologies and (automated or manual) practices.

The second layer, Logical Layer, expresses a logical description of business processes. This description dictates the conditions and actions of business processes, the relationships between them as well as operational constraints on data that processes operate on. The Logical Layer is a (semi-formal) business process model that describes business operations in ordered activities. It extracts and formalises business requirements using computer understandable languages, while leaving the corresponding (informal) source data side by side in the model for reference and justification of its formal representation. It also interprets and elaborates the abstract requirements described in the Business Layer into more concrete constraints using the designed language to provide direct design guidelines for the implementation of the software system. The process modelling language, FBPML, that will be described in Section 4 resides in this layer.

The formality described in this layer allows automatic communication with the next layer, the Implementation Layer. Logical layer, however, does not consider the mechanism which may be used to enact the described processes. Such issues are dealt with in the Implementation Layer. Examples of such issues are the software paradigm deployed, software and hardware systems involved, integration issues, and programming languages used. Descriptions in the Logical Layer may have multiple mappings to descriptions in the Implementation Layer. This is particularly applicable in a complex or an agent architecture system where different components may have different functionalities and means to implement the same logical process. They also need to collaborate with each other to accomplish a business process.

The logical layer specifies all of the process-related and the core set of data-related integrity constraints so that the implemented system does not violate any business or operational constraint. Since a business process may be enacted by different system components and they may be carried out concurrently, the business process model provides a common and sharable knowledge base for process communication during enactment. Because a business process model captures operational logic and is independent of technologies used for implementation, it is more robust against changes of technologies.

The Implementation Layer gives detailed step-by-step algorithmic procedures for software modules that implement processes described in the Logical Layer. Such algorithmic procedures may be described in a process modelling language that is capable of describing implementation details, or languages similar to flow-control and dataflow diagrams, or other application or system specific languages. Implementation Layer tends to be technology-dependent, it may be changed very frequently. For instance, an introduction of a new user interface, software or hardware system component may or may not result in a change in the logical layer, but will probably cause a modification of the corresponding descriptions in the Implementation Layer. For this reason, processes given in the Implementation Layer are volatile and disposable, as new technologies become available. They may be easily changed without disturbing a business's operation in a principle way leaving the business a more flexible and agile system.

Information that is manipulated by logical processes is organised in a hierarchical fashion, i.e. a *Domain Ontology*. The Domain Ontology gives semantics of the information stored and is comparable to a subset of classes that may be used to store operation related information in a database. It includes common classes (or a part of the schema for a "relational system") that are shared by different logical processes to allow them to exchange information under a standardised business practice. The Ontology is also mapped to procedures that are described at the Implementation Layer which allows information to be passed between the two levels based on the constraints prescribed in the logical processes.

As a process may be implemented differently in different system components, different versions of implementations may read, write, update or delete the same data sources concurrently following the explicit data management polices defined in the Logical Layer. The enacted processes may also communicate with each other through information that is under-pinned by the Domain Ontology. This mechanism enables a close collaboration between different process enactments and duplication of actions may be avoided and intelligent behaviours of the system may be generated.

The overall aim of the layered BPM framework is to provide a principled way for business process modelling that is flexible and therefore robust against changes in technology through time. It separates business requirements from technical issues when making decisions for developing workflow systems. This separation enables the workflow system to be more robust and agile in response to change of requirements in the dynamic environment that it operates within.

3 Requirements and Design of *FBPML*

To provide a business process modelling language that supports today's ever changing workflow environment and meets diversified requirements is not an easy task. A few design issues have been considered and acted upon, and are listed below.

- *Standard:* Modelling concepts that are described in the new BPM language should meet their specialised requirements but also need to be consistent with the current process modelling language standards. This not only keeps FBPML compliant with standard practices it also aids communication with other BPM languages and practitioners in the field. In essence, this means concepts that are included in standardised process modelling languages are main candidates to be included in FBPML. As a result, FBPML is an inherited, specialised and combined version of these standardised modelling languages. The main languages that have influenced the design of FBPML are IDEF3, PIF, PSL, RAD, CommonKADS CML and the Business Modelling method of IBM's BSDM.
- *Accessible:* The language should be easy to learn and use for both IT and business personnel. As one of the main business requirements for BPMLs is to enable business personnel to do BPM WITHOUT IT support.[12] To achieve this, FBPML covers *fundamental* process concepts that minimise complexity introduced by superfluous notations. It also introduces annotation notations that are informal and not directly understandable by machine. Such annotation is not formally a part of the model, but may provide useful explanation to the model, recording of design rationale or simply a reminder to assist the modelling process.
- *Collaborative:* An enterprise today is a virtual entity: it consists of a variety of enablers that are scattered across different geographical areas. Some enablers are human whereas others are autonomous agents or system components. Each enabler plays a role in its activities and is equipped with specialised functions, capabilities and authorities. Those enablers are characterised in their expertise and often behave in different ways that are best suited for their tasks and environment. However, to achieve organisational goals, they need to work collaboratively to accomplish their tasks. Traditionally, BPM methods do not include or explicitly represent the concept of such enablers, their responsibilities, authorities, how they collaborate with each other and what their relationships are between each other. The *roles* that enablers play, the relationships between them and information about them are captured in *FBPML* in the concept of *Role*.
- *Precise:* As most of the BPM methods are informal methods, they do not provide formal semantics for their notations. To avoid potential mis-use of the modelling language and mis-interpretation of built process models, there is a need for precise definition for notations so that a model may be interpreted correctly and consistently. IDEF3 provides a mature modelling method, graphical notations and sound conceptualisation about processes, but there is no formal semantic for its notation. PSL, on the other hand, does not have a visual presentation or method, but provides formal definitions of its concepts. This presents a natural opportunity to merge the two to gain benefits from both - this is the approach taken by FBPML.
- *Executable:* Semantics that are defined in the BPM language should include (or at least imply) operational definitions. This means the use of common process components, such as trigger, pre-conditions and postconditions, bear prescribed execution mechanisms. In addition, the types of executable activities also need to be identified and to be included as a part of the model. Process modelling methods are inherently rich in their semantics. The semantic of links between processes, for instance, are regarded as dependencies between processes, yet they also bear temporal constraints, and they may also act as triggers for the following processes. Junctions, such as AND, OR and XOR, may be interpreted differently depending on the use in the diagram, e.g. as a joint or split node. In addition, if both triggers and pre-conditions are defined in a process, they may bear distinct implications for

execution. Users of BPM need to understand such implications in order create a correct and appropriate model.

- *Formal:* Formality is important to connect a business process model to its execution phase. Ideally, there is a direct mapping from semantics of a business process model to application logic (as described in the logic layer and implementation layer in the previous section). This enables the separation between process and application logic, yet maintains declarative design of a workflow system. This implies modifications made at the logic layer automatically update processes at the implementation layer. If any inconsistency occurs, the system will give warning to the user. The formal approach has several advantages: automatic/intelligent analysis, verification, validation, and simulation facilities may be supported at the business layer[5][4]; once a business process model is satisfactory stable, it may automatically populate a large part of processes at the implementation layer.

4 A Declarative Executable FBPML - The Semantics

4.1 Activity, Decomposition and Specialisation

As mentioned in the previous section, FBPML should conform with standard practice. IDEF3, being a mature activity modelling method that largely meets our requirements, provides the foundation for *FBPML*. *IDEF3*[17] defines the concept of *decomposition* and *specialisation* of a process that *FBPML* also encompasses. Similar to *IDEF3*, the concept of *decomposition* in *FBPML* allows a process described at a higher level of abstraction to be decomposed into more detailed sub-processes that are more explicit for its implementation procedures. Each sub-process may also be decomposed into more detailed descriptions. The *specialisation* of a process indicates the alternative ways of carrying out a process.

Although there may be more than one alternative way of carrying out a task; unlike *decomposition* where all of the sub-processes must be carried out in order to accomplish the task, *specialisation* requires only one alternative sub-process to be carried out to accomplish the task. However, if one alternative activity does not finish the task due to some circumstances, another alternative activity may collaborate with the current one to accomplish the task. The detailed mechanism about how different alternative processes may work together in a coherent way in all eventualities requires a thorough examination of implementation methods. Since this is implementation dependent and outside the scope of this paper, it is not discussed here.

4.2 Notation

Figure 3 depicts the notation of FBPML as it is shown using KBST-EM (Knowledge Based Support Tool for Enterprise Models)[3]. There are three types of nodes: the *Main Node*, *Junction* and *Annotation*. Four types of *Main Nodes* are included: *Activity*, *Primitive Activity*, *Role* and *Time Point*. Two types of *Annotations* are included: the *Idea Note* and *Navigation Note*. Two types of links are provided: the *precedence-link* and *synchronisation-bar*. There are four types of *Junctions*: *and*, *or*, *start* and *finish*.

Main Nodes: As mentioned earlier, an *activity* node denotes the *type* of process that may be decomposed or specialised into sub-processes. In addition, the notion of *Primitive Activity* (from *PSL*) has been introduced to denote a *leaf node activity* that may not be further decomposed or specialised. Primitive activity is useful to FBPML, as it highlights the connecting point between the higher

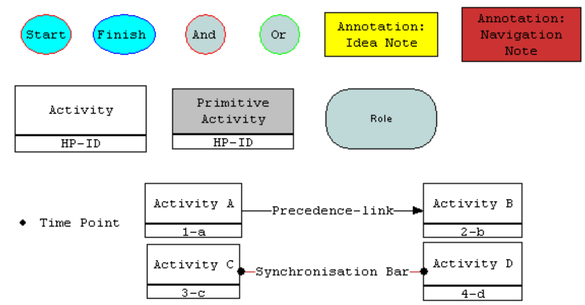


Figure 3. FBPML Notation

level process description and lower level implementation details that are described in the logical and implementation layers, respectively.

Although some process modelling methods distinguish terms between process, activity and task, as one is a higher level description of another, like IDEF3 and PSL *FBPML* does not make the distinction. Since a process may be further decomposed or specialised into sub-processes that may be again further decomposed or specialised, a process at one level is an activity to its "parent" process. As a result, these terms are used interchangeably in this document.

In FBPML, an activity is uniquely identified by its name (or ID)[1]. However, since FBPML (as well as *IDEF3*) permits the same activity to be repeated in different places *in a process model*, that normally exhibits different relationships between itself to other activities, the same activity may be enacted differently in a model in different places. Furthermore, since an activity may be a decomposition or a specialisation of its parent activity, this adds extra meanings depending on the type of sub-activity that it describes.

The *semantics of an activity to a model* is, therefore, defined together by *its location* in the model, *its usage* in the model and the *content* defined within itself, i.e. the *Trigger(s)*, *Pre-condition(s)* and *Action(s)*. *Post-condition(s)* is often defined as a part of a process and recorded in our model as it gives explicit checking points on successful execution of a process. However, since it is derivable from pre-conditions and actions of a process, we do not include it in our formal representation. In FBPML, the location of an activity is recorded in the field *Hierarchical Position (HP)*. Therefore, the tuple

$$< HP, Activity\_name, Trigger, Pre\_condition, Action >$$

defines an activity (type) in a model using FBPML, where each HP is unique and there may be more than one trigger, pre-condition and action. To denote the relevance to and uniqueness in a model, an activity is formally represented as:

activity(Activity_name, Hierarchical_Position)

where Activity_name is the name of the activity and Hierarchical_Position its location in the model. If A is a *primitive activity* in the model, the above predicate name, activity, is changed to primitive_activity. Since this paper only discusses semantics of notations but not their semantics in a model, for simplicity, this section assumes all activities are uniquely used in our examples and therefore

[1] For pragmatic reasons, an activity ID is created for each activity to provide a short hand identity for an activity. Each activity name uniquely maps to an activity ID and vice versa. Logically, we do not represent it, since it does not add additional semantics.

uses Activity_name instead of the above predicates, activity/2, when referring to an activity.

The predicate attribute(Activity, Attribute_name, Attribute_value) holds the specification for an Activity type where Attribute_name stores the corresponding attribute name, such as trigger, precondition and action, and Attribute_value stores the attribute value that may be a structured term or template with variables using specific grammar. Variables that are included in the Attribute_value will be instantiated dynamically by (process or object) instances at run time.[2]

The concept of *Role* is adapted from RAD where a *Role* is described as involving a set of activities which carry out a set of responsibilities. Such activities are "generally carried out by an individual or group within the organisation". Roles are also types and "there can be a number of different instances of a role type active at any one time within an organisation"[19]. In *FBPML*, the definition of *Role* is functional and as described above, it defines the "role" that an enabler plays in the context of the described activities. Upon process enactment, a role may be fulfilled by an individual, a group of people or software components, or a combination of the above. Similar to *RAD*, although different graphical presentation and process concepts are used, *FBPML* highlights interactions between roles: each role may have its own internal as well as communication processes. The communication processes allow explicit definition of interaction methods and boundary of communication within processes of each role. Tasks and issues may be delegated, escalated or transfered between roles as a part of communication processes.

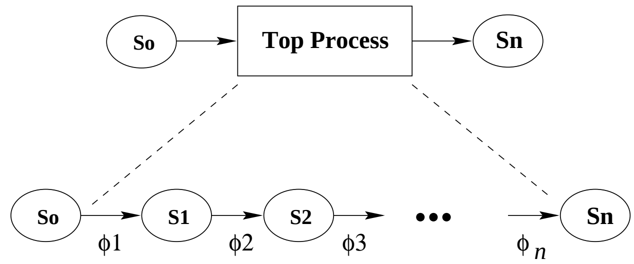
The notation of *time point* indicates a particular point in time during the enactment of a process model. The reference of time point may be decided by the implementation method of the model. A duration of a time interval is indicated by two time points. A length of time may not have association with any particular point of time.

Annotations: Two types of annotations are included: *Idea Note* records textual information that is relevant to, but outside the scope of, a process model, e.g. design rationale or a reminder for analysis for certain parts of a model; *Navigation Note* records the relationships between diagrams in a model. In general, annotation nodes do not contribute semantically to a process model, but they help the organisation and management of the modelling process.

Links: Two types of links are included: *Precedence-link* and *Synchronisation Bar*. *Precedence-link* is comparable to the more constrained Precedence Link, type II, in *IDEF3*. In FBPML, the specification that Activity A is preceded by Activity B is denoted by a *Precedence-link* from Activity A to B as shown in Figure 3. A Precedence-link places a temporal constraint on process execution that the execution of Activity B may NOT start before the execution of Activity A is finished when the two processes are on the same execution path. Figure 4 illustrates the concept of path and the execution of processes[1].[3]

In Figure 4, "Top Process" transforms from state So to Sn. It is also a parent process that may be decomposed or specialised into sub-processes. One way to propagate from state So to Sn then is to activate the appropriate sub-processes and execute them along the state path $\Pi = < So, S1, S2, ...Sn >$ by activating the process sequence Φ (where several process instances may execute synchronised or not to transfer from one state to another). We denote an execution of process instances along a state path Π in the predicate activation/2:

[2] A separate predicate is used to store process instance attributes.
[3] This Figure is adapted from [15].



State Path: Π **= <So, S1, S2, S3,... Sn>**
Execution Path: $\Phi = \{\phi 1, \phi 2, \phi 3, ... \phi_n\}$

Figure 4. Execution Path for Processes

$activation(\Phi, \Pi).$

Given the execution path, one can formally specify the temporal constraint between activity A and B in the formula below:

Axiom 1: Temporal Constraint

$$\begin{array}{l}
\forall Activity_a, Activity_b. \\
preceded_by(Activity_a, Activity_b) \\
\Rightarrow \\
\left(\begin{array}{l}
\forall A, \forall B, \exists P, \exists Act.instance_of(A, Activity_a) \wedge \\
instance_of(B, Activity_b) \wedge \\
activation(Act, P) \wedge \\
A \in Act \wedge B \in Act \\
\Rightarrow \\
end_time(A, activity_a) =< \\
begin_time(B, activity_b)
\end{array}\right)
\end{array}$$

A *Precedence-Link* suggests natural process flow which is if Activity A is executed, Activity B should also be executed along the corresponding execution path unless other conditions interact with it. We use $\triangleright$ to represent this nature of weaker inference that is pronounced as *should be* or *may be*. This definition gives a process model more flexibility and is slightly different from Precedence-Link Type II in IDEF3 where strong inference is prescribed. This rule is described formally below:

Axiom 2: Dependency Constraint

$$\begin{array}{l}
\forall Activity_a, Activity_b, \\
preceded_by(Activity_a, Activity_b) \\
\Rightarrow \\
\left(\begin{array}{l}
\forall A, P, Act.instance_of(A, Activity_a) \wedge \\
activation(Act, P) \wedge \\
A \in Act \\
\triangleright \\
\exists B, instance_of(B, Activity_b) \wedge \\
B \in Act
\end{array}\right)
\end{array}$$

A *Precedence-Link* also indicates that the completion of activity A invokes Activity B to be activated. We introduce a property **Temporal Qualification (TQ)** to denote that Activity B is temporally qualified to be executed. *Temporal Qualification*, however, does not guarantee the execution of an activity because it also depends on the content of trigger and pre-conditions of that activity. We use the predicate *tq(Instance, Process)* to indicate this property and *end/2* to indicate that the execution of a process instance is finished.

Axiom 3: Property of Temporal Qualification
$\forall Activity\_a, Activity\_b,$
$preceded\_by(Activity\_a, Activity\_b)$
$$\Rightarrow \left(\begin{array}{l} \forall A.instance\_of(A, Activity\_a) \wedge \\ end(A, Activity\_a) \\ \Rightarrow \\ \exists B.instance\_of(B, Activity\_b) \wedge \\ tq(B, Activity\_b) \end{array} \right)$$

The property of *TQ* is important as it implies execution logic of a process model that separates the notation between the execution of process instances and those that are only temporally qualified to be executed. We introduce a separate property **Full Qualification (FQ)** to define that a process is *Fully Qualified*, if it is *Temporally Qualified* and that all of its triggers and pre-conditions are satisfied. A fully qualified process instance may be executed immediately. Due to space, we do not describe the formalism here. The properties of *TQ* and *FQ* provide exact semantics for the execution logic that determines the dynamic behaviours of a process model at run time.

The above precise definition of *FBPML* links signifies how it differs from most other business process modelling languages. Since most business process modelling languages focus on the specification ability of a process, the actual implementation steps of a process are left out and are open to interpretation for system developers, e.g. IDEF3, IDEF0, PSL, Business Process Model in BSDM. Since the implementation considerations have not been provided by the original model, it leaves a question of whether the implemented system obeys the intended design of the system and/or whether the implementation has been carried out consistently with respect to the model. Since such process execution rationale has not been recorded at the first place, such questions are difficult to evaluate.[4]

Besides providing precise execution logic and instructions to the implemented workflow system, the above precise semantics allows both static as well as dynamic (state) Verification, Validation and Critiquing (VVC) facilities on the business process model. The static VVC techniques include error and appropriateness checking and critiquing based on the examination and comparison of different parts of the static structure of a business process model without the actual instantiation of the model. The dynamic VVC involves test runs of interesting scenarios through the model in an attempt to understand system behaviours at run time. Similar techniques have been applied and implemented in *KBST-BM*[2] for IBM's business model in BSDM.

As an activity may be decomposed into several sub-processes, the activation of a top process may be accomplished by activation of its sub-processes. In this case, the execution of the top process is not finished unless all of the corresponding sub-processes are finished. Again, we do not describe the formalism here.

The second type of link is *Synchronisation Bar*. A Synchronisation Bar places a temporal constraint between two time points. For example, one may synchronise the starting or finishing of two processes by synchronising the "begin times" or "end times" of the two processes. The Synchronisation between two time points is therefore defined below:

$$\forall A \in time\_point, B \in time\_point.$$
$$synchronisation(A, B) \Leftrightarrow A = B.$$

Junctions: Junctions are special or simplified activities, in that they do not have triggers and pre-conditions, and their actions have predetermined decision logic for starting, ending or branching process execution. Four types of Junctions are included in FBPML: *start, finish, and,* and *or* junctions.

The "start" and "finish" junctions provide an explicit indication of the logical starting and finishing points of a process. They may also isolate a part of a process that can be treated locally as a sub-process. These two junctions provide a clear indication for the entry and leaving points for the reader and when executing a process. It provides a natural decomposition for testing a process and a convenient indication for breaking a long complicated process when developing workflow systems using a divide-and-conquer strategy.

An "and" or "or" junction is a *one-to-many relationship* that describes process execution flow and temporal constraint between the activities that are connecting to it. Figure 5 shows how an "and" or "or" junction may be used in a process model. As shown in the figure, there are two types of interpretations of an "and" or "or" junction: the *joint* or *split* type of junction, depending on the topology of the process model.

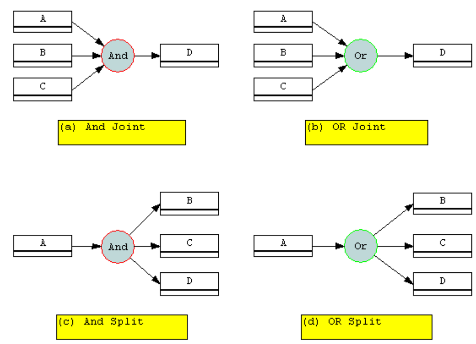


Figure 5. FBPML Joint and Split Junctions

An *and-* or *or-joint* indicates more than one preceding activity before the "and" or "or" junction, and only one activity following the junction. Figure 5(a) and (b) give example graphical representations of an and- and or-joint where each junction is attached to three in-coming arrows and only one out-going arrow. A joint type of junction is sometimes also referred to as a *fan-in* junction in some process modelling languages. Semantically, an and-joint indicates the process execution flow and the temporal constraint that *all* of the *preceding* activities must be finished before the following activity is temporally qualified and therefore be executed. An or-joint indicates *only one* of the *preceding* activities is required to be finished before the following activity becomes temporally qualified and executed.

An and- or or-split indicates that there is only one activity preceding the junction, but there is more than one activity following the junction. Figures 5(c) and (d) illustrate example and- and or-splits. A split junction is sometimes also referred to as a *fan-out* junction in some process modelling languages. Semantically, a split junction indicates process flow, temporal as well as dependency constraints. An and- or or-split indicates that *all* of the following activities become *temporally qualified* when the preceding activity is finished. Furthermore, an and-split also indicates that *all* of the *following* activities must be executed at some point of time after the preceding activity is finished.

On the other hand, an or-split indicates that *at least one* of the following activities of the "or" junction will be triggered and executed

[4] This is a recurrent problem that the authors have to deal with in one of their commercial business process modelling projects and their research projects.

when the preceding activity is finished. It is, however, unclear how many or which of the following activities will be triggered and executed, since it depends upon the corresponding dynamic system state and the trigger and pre-condition statements of the following activities. For both of the and- and or-split, all of the activities that are described after the junction may be executed in parallel or sequentially, when appropriate. The precedence-link and the junction do not specify the exact synchronisation between these activities. Such synchronisation is specified by *Synchronisation Bars*.

4.3 Combinational Use of Branching Junctions

Figure 6 demonstrates the four common combinational uses of “And” and “Or” junctions. The four basic cases of combinations are given in the Figure (a), (b), (c) and (d) accordingly and listed below: And-And, Or-Or, And-Or, Or-And.

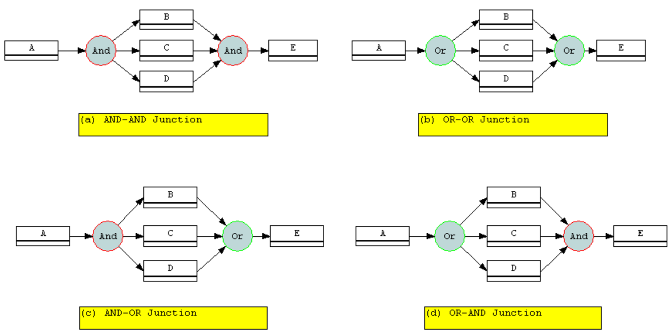


Figure 6. FBPML Junctions Coupled

According to the definitions given for “And” and “Or” junctions in the previous section, the and-and combination defines that activity B, C and D must execute at some point of time after but only after activity A is finished, and that activity E may not start execution before B, C and D have finished.

The or-or combination, on the other hand, gives a more loose constraint in that, similarly to and-and combination, activity B, C or D may only start execution after activity A is finished. However, it may not be the case that all of B, C and D are executed - it depends on the system dynamics and execution requirements of B, C and D. Nevertheless, since an or-split has been used here, at last one of B, C or D must be executed. When either activity B, C or D is finished, activity E may start its execution. The and-and and or-or combinations are demonstrated in Figures 6(a) and (b), respectively.

Similarly, in Figure 6(c), the and-or junction indicates that activities B, C and D may start their execution after activity A is finished, and activity E may start execution as soon as one of activities B, C or D is finished. What is different compared to Figure 6(b) is that activities B, C and D *must all* be executed at some point of time due to the and-split.

Figure 6(d) indicates that *at least one* of the activity B, C or D may be triggered and start execution after activity A is finished. Activity E may not start its execution unless all of the triggered activities, i.e. a combinations of B, C and D, are finished. Note that since an or-split has been used earlier in the process model, it may not be the case that all activities B, C and D are triggered. Nevertheless, all of the triggered activities must be finished before activity E may start its execution.

4.4 Discussion

As it has been described, an “And” or “Or” junction indicates a temporal constraint between the execution of connected processes. Furthermore, they also indicate the “execution” constraints that have been put in the process logic. For instance, an “and-split” indicates that all of the following activities must be executed when the preceding activity is finished. However, the model may not specify that all of the activities must be finished before the “next wave of activities” are started. One such example is given in Figure 6c, the case of and-or junction. Activities B, C and D may start execution in parallel but asynchronously and may finish their execution at different times. Activity E may start execution, as soon as one of them finishes execution. This means that activity E and activities following it may be executing along side the un-finished activity B, C or D. Furthermore, it is possible that all of the following activities after E are finished before activity B, C or D are finished. This may lead to an un-desirable result in the system.

The process model described in Figure 6c, however, is correct and appropriate when describing a situation where the start and execution of activity E is not temporally and semantically bound by activity B, C and D. However, when there is such a constraint at a later stage of the process that requires the finishing of the corresponding activity B, C or D, a limitation may be described in the triggers or pre-conditions of other following activities in the model.

One way to control and avoid “left-over” processes lingering indefinitely in the system is to define a process that is not finished until all of its (“left-over”) sub-processes are finished. Under this definition, the higher level process is not finished unless all of its sub-processes are finished. This is what has been defined in FBPML. Another way to control this is to provide a checking, alarming and repairing mechanism that will be triggered when processes are found lingering longer than a pre-determined period of time.

4.5 Demonstrating Dynamic Behaviours in Process Panels

As a part of the AKT project[6], for AKT-TIE[5], we have developed a small PC configuration business process model that accepts customer enquires and returns with possible pc-configuration specification. A snap shot of the business process model for the role “Edinburgh” is given in Figure 7 as it is shown in our support tool *KBST-EM*. This model has been successfully translated and displayed in a workflow stepping system, *I-X Process Panel*. Upon instantiation, instances of processes appear and are managed in *I-X system’s process panels*[24][16].

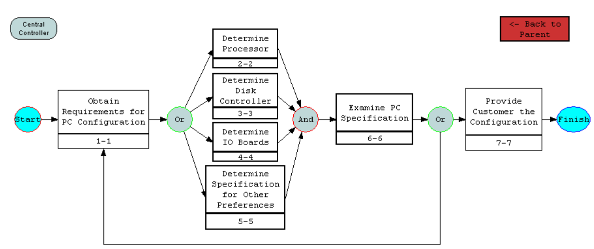


Figure 7. PC Configuration Business Process Model

[5] AKT-TIE is a part of the AKT project collaborating with Peter Gray and Kit Hui, Computer Science Department, Aberdeen University, UK.

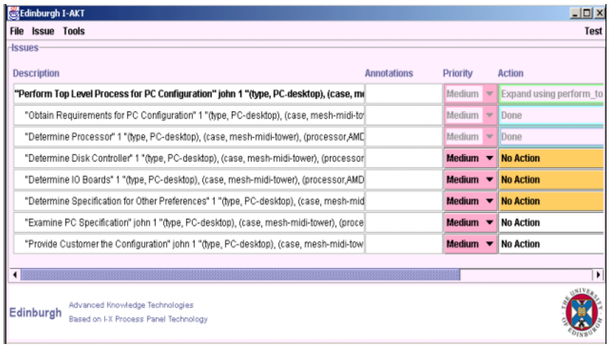


Figure 8. View of I-X System Process Panel

Figure 8 demonstrates how the instantiation of each process presents an entry in the I-X process panel. Each entry consists of two components: the name of the process and variables the process takes. The parent process of processes given in Figure 7, "Perform Top Level Process for PC Configuration", is shown at the top row and in bold face which is decomposed into sub-processes as those described in Figure 7.

In I-X, for each process instance, several actions may be performed upon them and the execution status of each instance is reflected by different colours. In I-X, all process instances may be executed (done), cancelled (Not Applicable), waiting to be processed (No Action (yet)), or decomposed into sub-processes (Expansion). Communication processes in our model may also dispatch tasks to other appropriate "roles" as defined in their processes. Branching of processes is controlled by the availability of actions that may be performed on the instances. For instance, in Figure 7 all processes on the second column of the model that are after the or-split may be executed in parallel, but this operation is only available after the "Obtain Requirements for PC configuration" process completed its execution.

It has become apparent that it is not an easy task to provide a declarative BPML that provides direct support for building and executing workflow systems and that more issues are to be investigated and resolved. Typical action types should be provided by the languages so that any models built using the language benefit directly from it, while at the same time one needs to allow flexibility and ease for addition or modification on existing action types. To safeguard against inconsistencies at the modelling language level is to provide some form of (automatic) inconsistency checking on static models and dynamic environments. Upon executing a process model, it is also vital that static processes are provided but the workflow system must be able to allow the users to dynamically modify or add new processes. Again, this will have to be done within a predetermined safety level.

5 Conclusion

Enterprise Models need to bridge the gap to software system development and execution, but additional mechanisms are needed so that information that is held within them may be transferred and mapped onto software execution. To bridge this gap, however, is not a minor task. Diverse and often conflicting requirements are need to be addressed. In addition, formality needs to be introduced to the informal or semi-formal enterprise modelling paradigm to provide precision and enable automatic support. When domain knowledge is used as a part of software system development and execution, it is also needed to ensure that it has been checked for consistency and appropriateness during the phase of enterprise modelling. This paper proposed a declarative modelling approach in an attempt to bridge the gap between business process modelling methods to (workflow) software systems.

Based on this approach, an initially static, high level business process specification may be represented formally and automatically. Based on the formalism, automatic verification, validation and critiquing may therefore be provided as a part of normal modelling activities. Furthermore, the modelling notation bears exact execution instructions that may be mapped to software modules that are components of a workflow system. This gives the prospect of rapid prototyping and testing of a workflow system that is based on the model. This benefit will not be possible without providing execution semantics in a model.

It will be advantageous that more similar work as reported in this paper is carried out for all Enterprise Models to narrow the gap which currently exists at various places between EM methods and software system development. When this is done, the set of Enterprise Models together may provide a holistic and clearer view as well as more direct instructions, particularly from the business, organisational, knowledge, information and process points of view, to assist the process of software system development for the organisation.

Acknowledgement

This work is carried out as a part of the Advanced Knowledge Technologies (AKT) (IRC) project[6], which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. The EPSRC and the Universities comprising the AKT IRC are authorised to reproduce and distribute reprints for their purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of the EPSRC or any other member of the AKT IRC.

REFERENCES

[1] A. J. Bonner, 'Workflow, transactions, and datalog', *Proceedings of the Eighteen ACM Symposium on the Principles of Database System (PODS), Philadelphia, PA.*, 294–305, (May 1999).

[2] Yun-Heh Chen-Burger, *Formal Support for an Informal Business Modelling Method*, Phd thesis, Artificial Intelligence, The University of Edinburgh, 2001.

[3] Yun-Heh Chen-Burger, 'Knowledge sharing and inconsistency checking on multiple enterprise models', *International Joint Conference on Artificial Intelligence, Knowledge Management and Organizational Memories Workshop, IJCAI 2001, Seattle, Washington, USA.*, (August 2001). Also available as Informatics Division Technical Report, University of Edinburgh. In print as a chapter of book, publisher: Kluwer.

[4] Yun-Heh Chen-Burger, Dave Robertson, and Jussi Stader, 'A case-based reasoning framework for enterprise model building, sharing and reusing', *Proceedings of ECAI Workshop: Knowledge Management and Organizational Memories, Berlin.*, (August 2000).

[5] Yun-Heh Chen-Burger, David Robertson, and Jussi Stader, 'Formal support for an informal business modelling method', *The International Journal of Software Engineering and Knowledge Engineering*, (February 2000).

[6] AKT Consortium. http://www.aktors.org, October 2000. Interdisciplinary Research Collaborations (IRC), Advanced Knowledge Tech-

nologies (AKT) Project. Partners: University of Southampton, Aberdeen, Edinburgh, Sheffield and Open University, UK.

[7] J.E. Dobson, A. J. C. Blyth, J. Chudge, and M. R. Strens, 'The ordit approach to organisational requirements', *Requirements Engineering: Social and Technical Issues*, (1994). London, ed. Jirotka and J.A.Goguen, Academic Press.

[8] *Workflow Handbook 2001*, ed., Layna Fischer, Future Strategies Inc., 2000.

[9] M. S. Fox and M. Gruninger, 'Enterprise modelling', *AI Magazine, AAAI press.*, 109–121, (Fall, 1998).

[10] Ulrich Frank, 'Multi-perspective enterprise models as a conceptual foundation for knowledge management', *Proceedings of Hawaii International Conference on System Sciences, Honolulu.*, (2000).

[11] Norbert. E. Fuchs and David Robertson, 'Declarative specifications', *The Knowledge Engineering Review*, **11**(4), 317–331, (1996).

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

[13] IBM, UK, *Business System Development Method: Business Mapping Part1: Entities*, 2nd edn., May 1992.

[14] Stefan Junginger, Harald Kuhn, Mark Heidenfeld, and Dimitris Karagiannis, 'Building complex workflow applications: How to overcome the limitations of the waterfall model', *Workflow Handbook 2001*, 191–206, (2000).

[15] Michael Kifer, 'Introduction to transaction logic', *A tutorial presented at IPLS'97 (International Logic Programming Symposium), Port Jefferson, Long Island, N.Y.*, (October 1997). www.cs.sunnysb.edu/ kifer/dood/tr-tutorial.html.

[16] J. Levine, A. Tate, and J. Dalton, 'O-P^3: Supporting the planning process using open planning process panels', *IEEE Intelligent Systems*, **15**(6), (November 2000).

[17] 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.

[18] National Institute of Standards and Technology, *Integration Definition for Function Modelling (IDEF0)*, December 1993.

[19] Martyn A. Ould, *Business Processes: Modelling and Analysis for Reengineering and Improvement*, John Wiley and Sons, 1995.

[20] D. Robertson and J. Augusti, *Software Blueprints: Lightweight Uses of Logic in Conceptual Modelling*, Addison Wesley, May 1999. in press.

[21] 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/.

[22] Jussi Stader and Ann Macintosh, 'Capability modelling and knowledge management', *Proceedings of Expert Systems 99, The 19th International Conference of the BCS Specialist Group on Expert Systems*, (1999). Springer-Verlay.

[23] A. Tate, 'Towards a plan ontology', *AI*IA Notizie (Quarterly Publication of The Associazione Italiana per l'Intelligenza Artificiale), Special Issue on "Aspects of Planning Research"*, **9**(1), 19–26, (March 1996).

[24] Autin Tate, 'I-X: Technology for intelligent systems', *www.i-x.info, AIAI, The University of Edinburgh*, (2002).

[25] Mike Uschold, Martin King, Stuart Moralee, and Yannis Zorgios, 'Enterprise ontology', *The Knowledge Engineering Review: Special Issue on Putting Ontologies to Use*, **13**, (1998). Also available as technical report from AIAI, The University of Edinburgh (AIAI-TR-195).

[26] Annika Waern, Kristina Hook, Rune Gustavsson, and Peter Holm, 'The common-kads communication model', Kads-ii/m3/sics/tr/006/v2.0, Swedish Institute of Computer Science, Stockholm, Sweden, (December 1993).