Division of Informatics, University of Edinburgh

Artificial Intelligence Applications Institute
Institute for Representation and Reasoning

Formal Support for an Informal Business Modelling Method

by

Jessica Chen-Burger, Dave Robertson, Justine Stader

Informatics Research Report EDI-INF-RR-0023

Division of Informatics **February 2000**
http://www.informatics.ed.ac.uk/

Formal Support for an Informal Business Modelling Method

Jessica Chen-Burger, Dave Robertson, Justine Stader

Informatics Research Report EDI-INF-RR-0023

DIVISION *of* INFORMATICS
Artificial Intelligence Applications Institute
Institute for Representation and Reasoning

February 2000

Appears in the International Journal of Software Engineering and Knowledge Engineering, February 2000. This paper is an extended version of the paper which was accepted by the Tenth International Conference on Software Engineering and Knowl

Abstract :

Business modelling methods are popular but, since they operate primarily in the early stages of software lifecycles, most are informal. This paper describes how we have used a conventional formal notation (first order predicate logic) in combination with automated support tools to replicate the key components of an established, informal, business modelling method: IBM's Business System Development Method (BSDM). We describe the knowledge which we represent formally at each stage in the method and explain how the move from informal to formal representation allows us to provide guidance and consistency checking during the development lifecycle of the model. It also allows us to extend the original method to a model execution phase which is not described in the original informal method. The role of the formal notation in this case is not to provide a formal semantics for BSDM but to provide a framework for sharing the information supplied at different modelling stages and which we can supplement with simple forms of automated analysis.

Keywords : Business Modelling, BSDM, Formal Method, Process Modelling, Enterprise Modelling.

Formal Support for an Informal Business Modelling Method

Yun-Heh Chen-Burger, David Robertson
Department of Artificial Intelligence
The University of Edinburgh
80 South Bridge, Edinburgh, UK
emails: jessicac@dai.ed.ac.uk, dr@dai.ed.ac.uk

Jussi Stader
AIAI, The University of Edinburgh
80 South Bridge, Edinburgh, UK
email: jussi.stader@aiai.ed.ac.uk

Abstract Business modelling methods are popular but, since they operate primarily in the early stages of software life-cycles, most are informal. This paper describes how we have used a conventional formal notation (first order predicate logic) in combination with automated support tools to replicate the key components of an established, informal, business modelling method: IBM's Business System Development Method (BSDM). We describe the knowledge which we represent formally at each stage in the method and explain how the move from informal to formal representation allows us to provide guidance and consistency checking during the development lifecycle of the model. It also allows us to extend the original method to a model execution phase which is not described in the original informal method. The role of the formal notation in this case is not to provide a formal semantics for BSDM but to provide a framework for sharing the information supplied at different modelling stages and which we can supplement with simple forms of automated analysis. [1]

Key-words Business Modelling, BSDM, Formal Method, Process Modelling, Enterprise Modelling.

1 Introduction

Traditional software engineering techniques focus from the beginning on the system that needs to be built. There is little effort to analyse and understand how the business, in which the new system is to be deployed, is organised and how it operates. As a result, in the past a number of software systems were built that were eventually found largely unsuitable for the organisation. In response to this problem, over the last decade, software specification and requirement engineering were seen to be the key to have the right system developed in the first place, but this alone was not sufficient. Jackson illustrated further problem.

> "Requirements engineering is about the satisfaction of goals. But goals by themselves do not make a good starting point for requirements engineering. To see why, consider a project to develop a computer-controlled turnstile guarding the entrance to a zoo... the real goal is to ensure the profitability of the zoo." [17]

To address this problem, a variety of domain modelling methods can be used prior to the requirements elicitation and specification of a software system. These are the methods, for example, for business modelling[8], process modelling [10] [11] [16], enterprise [12] and organisational modelling [3] [4], business process re-engineering [13] [14], and management of enterprise knowledge [5] [6]. By providing a structural framework, these methods aim to help an enterprise capture its enterprise-wide knowledge which forms the basis for targeted analysis and helps the re-shaping of a business. These methods

[1] This paper is an extended version of the paper which was accepted by the Tenth International Conference on Software Engineering and Knowledge Engineering, SEKE'98, and published in its proceedings, June 18-20. The original paper was selected as one of the best papers for the conference.

also provide a neutral forum where people of different disciplines can communicate with each other. A key goal of applying these methods is to seek ways to improve an organisation's effectiveness, efficiency and profitability.

The benefits of a successful application of domain modelling methods can be tremendous. For example, the U.S. Department of Defence reported in 1997 that the application of a business process re-engineering project, leading to a combined use of modernised business practice and computing technology in its organisation, has led to 1.6 billion US Dollars in savings in inventory management since 1993 [14]. There are also other individual success stories [15] . However, not all applications of these methods have been equally successful. One key factor in the successful application of these methods is the quality of the produced model, i.e. to ensure that the produced model is the right one for the organisation. There are several problems.

- *Availability of expertise:* a modern enterprise today is a virtual entity that consists of many sub-organisations which are distributed across different geographical areas, each possessing different expertise. Hence, it may not be possible to have all of the persons with the right expertise available for the modelling development. Furthermore, the required expertise may be changing as companies have to react — change their goals and processes — to today's fast changing global economies.

- *Lack of a comprehensive evaluation method:* most of the domain modelling methods mentioned above provide a description of how to carry out the modelling tasks and some measurement criteria for how well the model fits reality. However, none of them supports a comprehensive and systematic approach with respect to determining the correctness, completeness and appropriateness of a model, both method-wise and enterprise-wise.

- *Informal or semi-formal modelling context:* The first step in checking if a model is appropriate for its purpose is to understand the content that a model describes. Many of these domain modelling methods are informal methods, some of them are semi-formal which include pre-defined diagrammatic symbols supplemented with natural language. It is generally difficult to ensure the correctness and consistency of informal and semi-formal methods, because the checking normally involves a person to read and check all of the details of the model which for a complete real industrial-sized model is an impossible task.

- *Time pressure:* Very few projects can enjoy the luxury of not having to deal with strict time constraints. In the model building context this means that there is a need to provide an efficient and effective way to maximise the productivity of the modellers in building a model, verifying and validating it, and finding and correcting inadequacies in the model. This normally suggests that a suitable software support system should be provided. Some such software tools have been offered, but most of them concentrate on model building, storage and report generation, without support for the important aspect of model verification and validation. This generally means that there is not enough time to carry out the tedious tasks of validating a model by hand.

- *Lack of efficient and effective knowledge transferring means:* Domain modelling is intended to help enterprise knowledge transfer between software system developers and the managers, but it requires a sufficiently wide use of a particular method, so people can communicate through it. Most methods do not have wide usage at this stage and would require additional training of staff. This may be difficult due to internal resistance in the organisation. A tool which can ease the communication (using a particular method) between people could thus be very helpful in the transfer of knowledge.

- *Dynamic aspects of a model are complex:* A domain modelling method normally captures the static structure of the targeted domain, but it often implies and prescribes the actual activities to be carried out. As many of these dynamic activities may be happening concurrently and interacting with each other, to understand the impact of them becomes in general a task too complex for un-aided human reasoning. Therefore, it is important that these processes can be simulated within the model with the help of a software tool to demonstrate and/or predict their behaviour, to help people understand their implications and restrictions.

To cope with the changes of today's business world caused by the advances of electronic and computing technologies, business organisations must adjust and/or re-shape themselves to thrive in the new post-industrial era.[2] The potentially great rewards offered by applying domain modelling methods have lured many businesses to use them. Unfortunately, these domain modelling methods have problems. One particular problem is to determine and assure the quality of the built model.

[2]The term "post-industrial era" is taken from [1].

We have proposed a formal framework for one of the business modelling method *BSDM*. Based on this framework we have built an automatic support tool *KBSTBM* to assist the user in automatic model verification and validation.

The rest of the paper is organised as follows. Section 2 gives an overview of the business modelling method, *Business System Development Method (BSDM)*. Section 3 summaries the motivation for this work and the overall objectives. An overview of how the formalisation of business models can be used in the context of this work is given in section 4. The basic representational approach is introduced in Section 5. The layers mentioned above are discussed in sections 6, 7 and 8, respectively. Each of these sections includes a discussion of the layer-specific domain knowledge representation and inferencing of new information. The paper concludes with section 9.

2 BSDM

IBM's *Business System Development Method (BSDM)* is an informal method for developing business models. These business models are used to capture and represent the given business environment (for which software has to be written) in graphical and textual format. They serve as a communication tool to converge the views of senior managers and as a basis to build software systems. A good business model also helps software engineers to come to a better understanding of what software is needed for the company. In BSDM terms:

> "*A business model is a model of the business that defines the things that the business needs to manage, and the business rules that govern the behaviour of those things.*" [7]

Initially a BSDM business model consists of an *entity model*, which is later extended to a *process model*, both of which are specified in a semi-formal way using diagrams and English text. An entity model describes the key components of a business' operation, e.g. the actors who are involved in the business operations within and outside of the business, abstract and physical information about the operation, key activities carried out to achieve certain business purposes, and the inter-relationship between these things. The constituent elements of a business are captured and denoted as *entities* with dependencies placed between each entity and those others on which it relies for its existence. A process model is a collection of business processes crucial to the business' operation. The context of a business process, the circumstances which trigger a process and the consequences of its actions are described.

On top of the entity and the process model familiar to conventional BSDM practitioners, we introduce another layer, the *procedural model* which extends the process model. Figure 1 illustrates our layered modelling approach and also shows which domain knowledge is being formalised for each layer and what kind of information can be inferred through the formalisation process. The domain knowledge for each layer can be divided into two types: model components and rules, which are shown in Figure 1 as ellipses and boxes, respectively. From the entity and process models, any violations of modelling rules or guidelines and possible corrections can be inferred. From the procedural model, the state transitions of the model caused by the execution of business processes can be inferred.

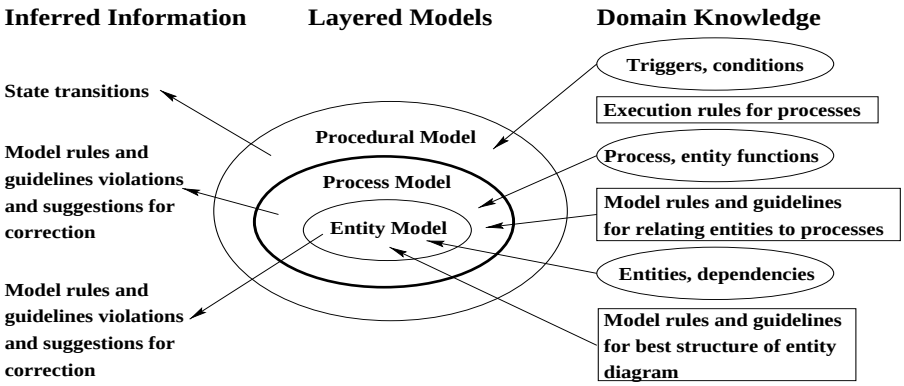

Figure 1. An Overview of The Executable Business Model

3 Approach and Objectives

As is the case for any modelling activity, creating a model is only the first step in a larger cycle. Once a model has been designed, it needs to be verified and validated. *Verification* is the process of checking that no modelling rules have been violated. *Validation* is the process of confirming that the model is a true representation of the real world. The lack of a formal representation in methods like BSDM makes verification of a business model a tedious and error prone task. To validate a model, the modeller must as a minimum be able to work through the execution of typical scenarios for business processes and then compare these with the real world. For all but very simple business models, this is not achievable through a simple paper and pencil exercise and, hence, detailed validation has not been possible in the past.

Our aim is not to improve the method itself, but help to improve the quality of its products. We provide support that is closely tied in with the method so that the original practices are not disturbed and no unnecessary unfamiliarities are introduced to the user. [3] The objective of our work is to provide support not only for creating and storing models, but also for automating, as far as possible, the verification and validation of business models. By doing so, we provide the means to complete the modelling cycle (i.e. the modeller can go through several iterations of design, verification and validation until a satisfactory business model has been produced.)

In formalising BSDM business modelling, the objective was not only to develop an appropriate formal representation of a model, but also to take advantage of the knowledge of BSDM about how to build such models and the existing set of rules about how to evaluate the quality of them. Furthermore, the formal representation of a BSDM business model must be able to capture not only the static aspects of the model, but also the dynamic ones.

Although in conventional BSDM the various states of the model can be captured, there is no explicit way of describing how a process is carried out and how entities are manipulated by a process. To enable execution of a business model, i.e. to simulate the execution of business processes, this knowledge of how to carry out a process is essential. An additional objective of this work was, therefore, the introduction of a procedural model on top of the entity and process model. Also, to be able to deal with the dynamic aspects of the models, the explicit representation of time needed to be introduced into the formalism.

Before describing in more detail how models and modelling rules are formally represented and then reasoned about, we first give a modelling overview below.

4 Modelling Overview

In this section, we briefly describe the overall context in which we use the formalisation of a business model. Figure 2 describes the modelling overview.

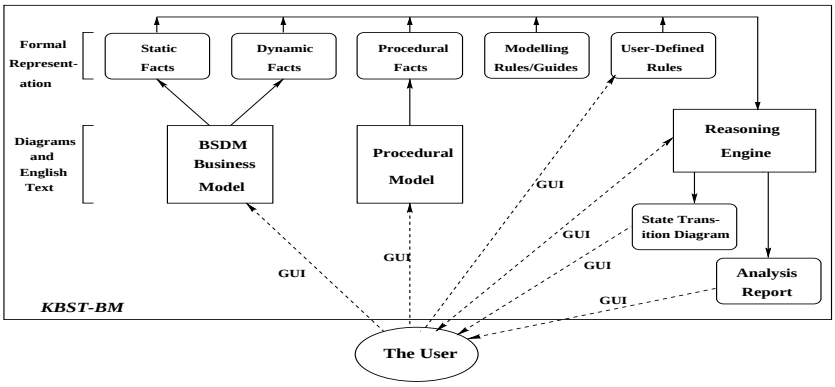

Figure 2. A Model Overview

The tool provides various *GUIs* (Graphical User Interfaces) which allow the user to build *BSDM Business Models* and *Procedural Models* and specify *User-Defined Rules.* Its embedded BSDM knowledge, *Modelling Rules/Guides*, together with

[3] It is important not to disturb the use of the original method, as this may cause unwanted distortions in the method and lead to resistance from the practitioners to use the formal method.

the formalised input from the user are fed into a *Reasoning Engine* which generates *Analysis Reports* and *State Transition Diagrams* of the model for the user.

At the beginning of developing a business model, business managers together with a BSDM facilitator create a business model which is described using conventional BSDM notation. This model is recorded using the tool by the BSDM facilitator (*The User*). From the model, a formal representation can then be automatically derived, one for the static and one for the dynamic aspects of the model. In addition, the user-defined rules which are given by the user are also formalised. The general rules are considered to be fixed and, therefore, require no user input and are already formalised. At this stage, the procedural model is not yet needed.

The formal representation of the model as well as the rules provide input to the *Reasoning Engine* which analyses the business model. An *analysis report* is produced which describes any violation of modelling rules and makes recommendations for how the model can be improved. The user can then modify the business model accordingly and start the next iteration of this process. The cycle is repeated until no further errors and recommendations are produced, or when the user decides not to incorporate any more of the given recommendations. Once no more modelling rule violations are reported by the reasoning engine, verification of the model is complete.

Before the user can simulate and therefore validate the model, he/she must create the *procedural model*, which can then be formalised and fed into the reasoning engine. During simulation of the model the user interacts directly with the reasoning engine — telling it which process to execute from which starting time. The reasoning engine generates and maintains a state transition diagram and information about the various states, which the user can compare with the effects of business processes in the real world. The user can decide to make necessary changes to the business model and again validate it using the reasoning engine.

5 Basic Representational Approach

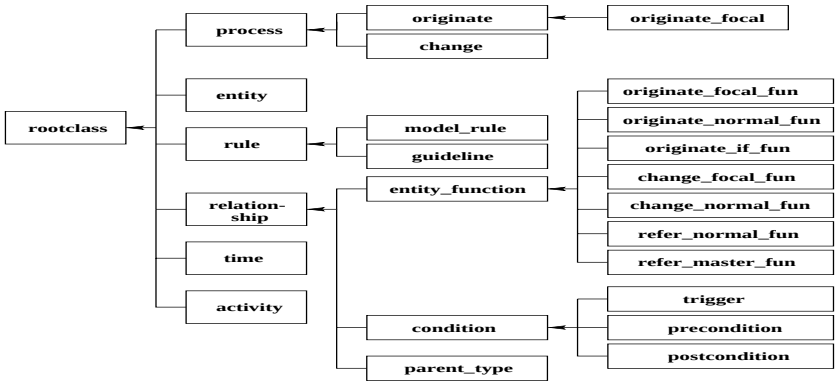

Figure 3. A Simplified Class Inheritance Hierarchy

The formal representation used in this paper is based on an extended version of first order predicate logic. We use either a Prolog-like syntax or first order predicate logic expressions. The formal representation of modelling concepts and rules is organised in an inheritance class hierarchy (Figure 3) which was inspired by the *PIF* core class hierarchy [10]. At the root of this hierarchy is a superclass called *rootclass*. The predicate

class(Super, Sub)

is used to denote the relationship between classes in this inheritance hierarchy: the class *Super* is a superclass of class *Sub*. Similar to the PIF class hierarchy, properties associated with a superclass are passed onto its subclasses through the inheritance hierarchy. Unlike the PIF hierarchy, however, this hierarchy not only captures the facts about a business model, it also includes classes that represent modelling rules.

A rule may be applied to a class anywhere in the hierarchy. It may also be applicable to a set of classes. Subclasses inherit the association with rules from their parents. For example, if a particular rule is used on a class called *originate* process, then that rule applies to all subclasses of *originate*. Modelling these rules as their own set of classes allows for an easier identification and analysis of the modelling rules that apply to a particular domain.

6 A BSDM Entity Model

Figure 4 shows part of an example BSDM business model as it appears in our system. This simplified model describes the selection and evaluation of modules at a university.

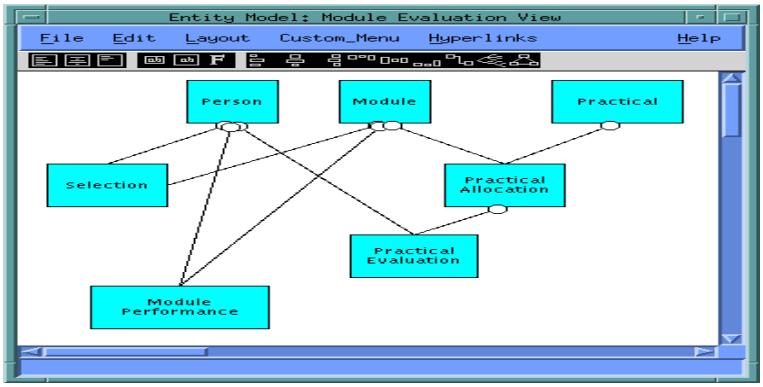

Figure 4. A BSDM Entity Model

The notation used is that described in the BSDM manuals [7], so existing BSDM practitioners are conversant with the notation. Boxes show the entities involved: 'Person', 'Module', 'Practical', 'Practical Allocation', 'Practical Evaluation', 'Selection' and 'Module Performance'. Lines with a circle at one end denote a *dependency* relationship between two entities: the entity with the circle ending is the "parent" entity, whereas the line ending entity is the "child" entity. For example, in this figure, 'Selection' has two parent entities: 'Person' and 'Module'. This denotes that it is impossible to have a 'Selection' (of a module) without knowing the person and the module involved. Hence, a 'dependency' places constraints on a child entity.

6.1 *Representing Entity Models*

An entity represents a class of things in the world. Each 'Person', 'Module', 'Practical', 'Practical Allocation', 'Practical Evaluation', 'Selection' and 'Module Performance' is a specific kind of entity, and they are therefore represented as subclasses of *entity* in the inheritance class hierarchy. Using this class hierarchy, we can represent all entities shown in this business model. For example, entity 'Person' is represented as:

class(entity, 'Person').

To represent the dependency between entities, a predicate

parent_type(Entity, Set_of_Parents)

is used, where *Entity* denotes the child entity and *Set_of_Parents* is the corresponding set of parent entities. For example, 'Person' does not have a parent entity, therefore, its Set_of_Parents is empty, as shown below. 'Practical Allocation' has two parent entities, therefore these two entities are enclosed in the set, also shown below.

parent_type('Person', [ ])
parent_type('Practical Allocation', ['Module','Practical'])

6.2 *Representing BSDM Knowledge*

As mentioned earlier, the BSDM manuals provide advice on how to build good business models. According to the degree of enforcement on a model, this can be classified into two categories: model rules and guidelines. Model rules are strongly recommended if the model is to be sound. Model guidelines are advice for a model of good style. To distinguish these two different strengths of enforcement, two implication operators are deployed in the formalism: '$\Rightarrow$' is used to represent a model

rule and is read as 'must be'; '$\triangleright$' is used for model guidelines and is read as 'should be'. A formula 'P $\Rightarrow$ ($\triangleright$) Q' reads 'if P is true then Q must (should) be true'.

For instance, BSDM recommends that the depth of an entity model should be no more than 4 layers, i.e. 4 steps through parent links. This is to prevent a model from being over-constrained by several layers of dependencies through levels of entities. If we define *property(Entity, level, N)* to mean that *Entity* is located at level *N* in the business model, then the "4-layer"-guideline can be described as:

$$property(Entity, level, N) \triangleright N \leq 4.$$

In addition to advice given explicitly in BSDM manuals, there are rules which are not mentioned in the method but are natural deductions from the method itself and, therefore, must be followed to create a sound model. These rules are also identified and captured as part of the formalisation. An example of such a rule is that there must be no circular dependency relationship (a parent being dependent on its child) in a business model. To describe this rule, we define *ancestor(P, Q)* to mean that *P* is either a parent entity of *Q*, formally defined by the *parent_type* predicate, or that it is an ancestor entity of *Q* through *Q's* parents, i.e. using the transitivity property of the *parent_type* predicate. The "non circular dependency"-rule is then represented by the expression below.

$$class(entity, X) \Rightarrow \neg ancestor(X, X)$$

6.3 *Inference*

Based on the formalisation of rules and guidelines and possibly partial information of an entity model, so-called *critiques* are inferred. Several types of critiques are provided.

- *Correctness* critiques detect structural, syntactical and semantical errors.
- *Completeness* critiques identify incomplete information in the model and suggest which missing concepts may need to be included.
- *Consistency* critiques point out discrepancies in the model.
- *Appropriateness* critiques show deviations from standard practices.
- *Presentation* critiques highlight awkward use of naming style which can lead to misunderstandings or conceptual errors.
- *Alternative* critiques search for similar standard models and present them as alternatives to a given modelling decision. This critique makes use of a case library of known standard models (see below for more comments on the use of case-based reasoning techniques).

To provide these critiques, the logic expressions representing the modelling rules are used. Each expression is negated and translated into a Prolog goal. It is then used to detect instances that satisfy the negated rule, thus finding violations of the modelling rule. For example, logical expressions of the form below:

property1(X) $\Rightarrow$ property2(X) (must always be true), and
property1(X) $\triangleright$ property2(X) (should always be true)

are negated and implemented as Prolog goals for which we try to find all instances that satisfy the goal (using backtracking technique):

```
ruleGoal_n :-  property1(X), not(property2(X)),advise(ruleGoal_n,X).
guidelineGoal_n :-  property1(X), not(property2(X)), advise(guidelineGoal_n, X).
```

For example, the negated "circular dependency"-rule above will be "there exists at least one entity which has a circular dependency relationship". Our implemented Prolog goal will be proven to be true, if it can find at least one entity which has a circular dependency relationship.

In response to a user's request to verify a business model, the inference engine tries to prove each *ruleGoal_n* and *guidelineGoal_n* to be true. If any *ruleGoal* or *guidelineGoal* can be proven to be true, a message is given to the user (via the Prolog goal *advise(Violation_id, Source_data)*) indicating which rule/guideline has been broken and how the error might be corrected. The inference engine can provide information about the specific entities and processes which it has found are the cause of the violation. For instance, if the system has found a cyclic dependency in the model, a warning is given to the user containing information about this type of violation, including a list of entities which are involved in the cycle, and a suggestion for what dependency links may have to be erased.

Although a detailed discussion is outwith the scope of this paper, the inference engine makes use of *case-based reasoning* techniques [9]. BSDM provides a catalogue of generally recommended topologies (standard models) which we have formalised and stored in a case library which the inference engine uses for analysis/correction of new models. The system is capable of detecting when a user's model is similar to an existing standard model in the library, report the differences and suggest actions which would bring the new model in line with a recommended standard model[2].

7 Process Model and Process-Entity Matrix

Conventional BSDM provides an informal method for extending an entity model to a process model. Figure 5 shows a screenshot of our system when describing an example process model which was built on top of the entity model shown in Figure 4. Two processes are shown in the figure: 'PRACTICAL MARK Assignment' and 'MODULE PERFORMANCE Assessment'. Each includes four entities in its scope.

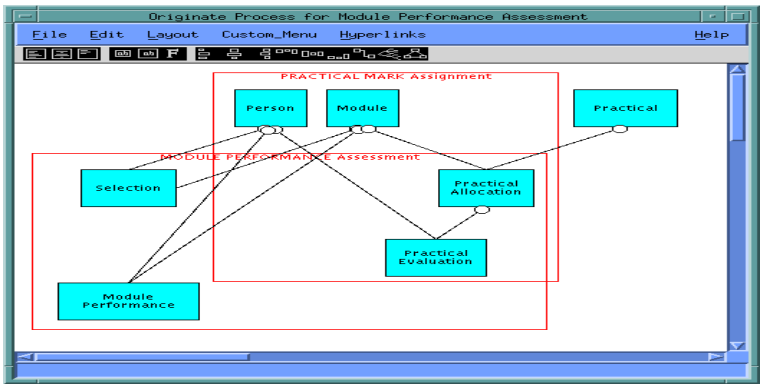

Figure 5. A BSDM Process Model

A BSDM process includes at least one entity in its scope. The relationship between a process and its included entities is determined by the specific role that each entity plays in the process. This relationship is called an *entity function* and is normally summarised in matrices like the one in Figure 7.

Process-Entity Matrix		
	PRACTICAL MARK Assignment	*MODULE PERFORM Assessment*
Person	*refer normal*	
Module	*refer normal*	
Practical Allocation	*refer normal*	*refer normal*
Practical Evaluation	*originate focal*	*originate in-flight*
Selection		*refer normal*
Module Performance		*originate focal*

Figure 6. Process-Entity Matrix

There are seven entity functions in BSDM: *originate focal*, *originate normal*, *originate in-flight*, *change focal*, *change normal*, *refer normal* and *refer master*. The originate type of entity functions, such as *originate focal*, *normal* and *in-flight*,

specifies the creation of occurrences (instances) of the corresponding entity (type). The change type of entity functions, such as *change focal* and *normal*, states the update operations carried out on the properties of the entity. The refer type of entity functions, such as *refer normal* and *master*, dictates the content of referencing the entity in a process execution.

The three types used in our example are: *originate focal*, *refer normal* and *originate in-flight*. The first one is used when a process creates occurrences of the corresponding entity. The second indicates that the process only refers to information in the entity. The third is used for entity/process pairs where the process normally only refers to the information in the entity, but can also create an occurrence of the entity if it doesn't already exist when the process starts execution, i.e. the third case is really a combination of the previous two. The matrix shows the relationships (entity functions) between the corresponding processes and entities.

The BSDM modeller specifies the role of each entity in each process. Based on the various roles the entities play in a process, the process is then classified as a particular type of process in the class hierarchy in the formalism. For example, process *PRACTICAL MARK Assignment* is an *originate focal* process, and it is therefore represented as:

class(originate_focal, 'PRACTICAL MARK Assignment').

This process' primary purpose is to assign a mark to a person's practical, i.e. to create an occurrence of the *Practical Evaluation* entity (the *originate focal* entity). To create the occurrence, information stored in the relevant entities, *Person*, *Module* and *Practical Allocation*, is used. Hence, these entities are *refer normal* entities. In process *MODULE PERFORMANCE Assessment*, *Practical Evaluation* is an *originate in-flight* entity. This indicates that all practical marks must be known before the *Module Performance* can be calculated; otherwise, the corresponding practical marks must be calculated, as part of the process, before the *Module Performance* can be determined.

Since entity functions are relationships between processes and entities, they are modelled as subclasses of *relationship* and *entity_function* in the inheritance class hierarchy (Figure 3). To represent entity functions, a predicate *Entity_function(Process_name, Entity_name)* is used. For instance, the formal representation of the three entity functions which are used in the above example, *refer normal*, *originate focal* and *originate in-flight*, is given below.

refer_normal_fun('PRACTICAL MARK Assignment', 'Person')
originate_focal_fun('PRACTICAL MARK Assignment', 'Practical Evaluation')
originate_if_fun('MODULE PERFORMANCE Assessment', 'Practical Evaluation')

7.1 *Representing BSDM Knowledge*

BSDM provides check lists of rules and guidelines which are used to help the modeller to review if a developed model is appropriate. For instance, the following question should be checked: 'Are all entities originated (created) by at least one process?'. Since the modeller may decide that the creation of an entity lies outwith the scope of his/her model, the question forms a guideline, rather than a strict rule. Since an entity can be created in either of the three entity functions, *originate focal*, *originate normal* or *originate in-flight* entity-function, this guideline states that each entity must play the role of either *originate focal*, *originate normal* or *originate in-flight* entity-function in at least one process:

$$class(entity, Entity\_name) \triangleright \exists Process\_name. \left(\begin{array}{l} originate\_focal\_fun(Process\_name, Entity\_name) \vee \\ origiante\_normal\_fun(Process\_name, Entity\_name) \vee \\ originate\_if\_fun(Process\_name, Entity\_name). \end{array} \right)$$

Given the above guideline, we infer a less restrictive rule with a stronger recommendation which states that all entities in the model *must* be included in at least one process for some roles. For this, we use the following rule:

$$class(entity, Entity\_name) \Rightarrow \exists Process\_name. \left(\begin{array}{l} originate\_focal\_fun(Process\_name, Entity\_name) \vee \\ originate\_normal\_fun(Process\_name, Entity\_name) \vee \\ originate\_if\_fun(Process\_name, Entity\_name) \vee \\ change\_focal\_fun(Process\_name, Entity\_name) \vee \\ change\_normal\_fun(Process\_name, Entity\_name) \vee \\ refer\_normal\_fun(Process\_name, Entity\_name) \vee \\ refer\_master\_fun(Process\_name, Entity\_name). \end{array} \right)$$

There are more model rules and guidelines which we will not mention here. They all follow the same basic notation as the two examples above.

7.2 *Inference*

The process model analysis detects errors existing in processes and in the inter-relationships between entities and processes, e.g. entity functions. Similar techniques as used for the entity model in 6.3 are also applied here. *Correctness* critiques present structural, syntactical and semantical errors in a process. *Completeness* critiques alert the user of missing information in a process and potentially omitted links between processes and entities. *Consistency* critiques highlight contradictions in process properties and entity-process relationships. *Alternative* critiques find subsumed and over-specialised processes and suggest alternative processes (in contrast to entity models, this is not done by CBR, but using guidelines). *Appropriateness* and *Presentation* critiques point out differences between the user model and standard practices and provide correction advice. Each of these entity and process model critiques performs independently. It therefore gives the user the freedom to either run a complete check on the model or to choose to work on a particular aspect of the model, i.e. to check for certain critiques only. This helps the user to focus on a particular design issue and not be overloaded by too much advice which is of no immediate concern.

Inference using the process model follows the same approach as described for the entity model. Model primitives and their properties are each represented formally by logical expressions. Rules and guidelines are negated and translated into Prolog goals which make use of the given class hierarchy and the formalisation of the primitives and the derived information. Once a Prolog goal has been proven to be true (detecting a violation), an explanation of the modelling problem and possible solutions are suggested.

So far only the static aspects of a business model have been discussed. The following section describes how we execute processes.

8 An Extension: Procedural Model

Conventional BSDM business models describe semi-formally what processes are, what they consist of and what can be done by them. The BSDM method, however, is not specific about how a process may be executed. In order to allow the execution of BSDM business processes, we introduce a *procedural model* which specifies the logical sequence of a process' execution. This is not the only possible form of execution consistent with the earlier stages of BSDM, but is characteristic of the sort of execution which fits well with the style of modelling. The notation used to describe this stage of modelling is not present in conventional BSDM manuals, so practitioners require training to use it.

The procedural model was inspired by process modelling methods such as organisational process [11], IDEF3 [12], PSL [16] and workflow modelling methods [6]. These methods, however, emphasise on specifying and managing tasks which are operational and can be realised in an organisation, e.g. the process of designing, building, testing and manufacturing a new product. The procedural model has been designed to specify the *logical* execution sequence of a process which is independent of an organisation's current working practice and therefore can be implemented in different ways depending on the organisation's goal and requirements. In other words, it specifies the data to be manipulated and conditions to be considered, but not how to do them in the real world. This is consistent with the declarative style of conventional BSDM which concerns itself with things which should be done, and does not give a deterministic order on execution. However, some commitments to the order of execution are required at this stage of design and these are reflected in the successive layers of Figure 7.

The figure shows an example Procedural Model for an *originate focal* process. This model is derived from a generic structure of successive layers of tasks which can be used as a template for most *originate focal* processes. A process execution follows the arrows in the diagram. It always begins with a 'start' node after which comes a 'trigger' node which is followed by preconditions and an 'and' node which leads to actions. After actions are the postconditions and finally the 'end' node.

A trigger represents a request to invoke a particular process. It can be caused externally (by the user) or internally (by another process). If a trigger is present, all preconditions specified in the procedure must be satisfied before any actions can be carried out. All postconditions will be confirmed after all actions are finished. In the example, the box labelled 'not exists originate focal entity' is a precondition. An example action box is 'create originate focal entity', and an example postcondition is the 'exists originate focal entity'.

Triggers deal with the dynamic state of a model and can be dynamically generated. A trigger needs to specify *which* type of process it invokes, *when* a process should start to act, and *what* actions are to be carried out. In addition, it also needs to

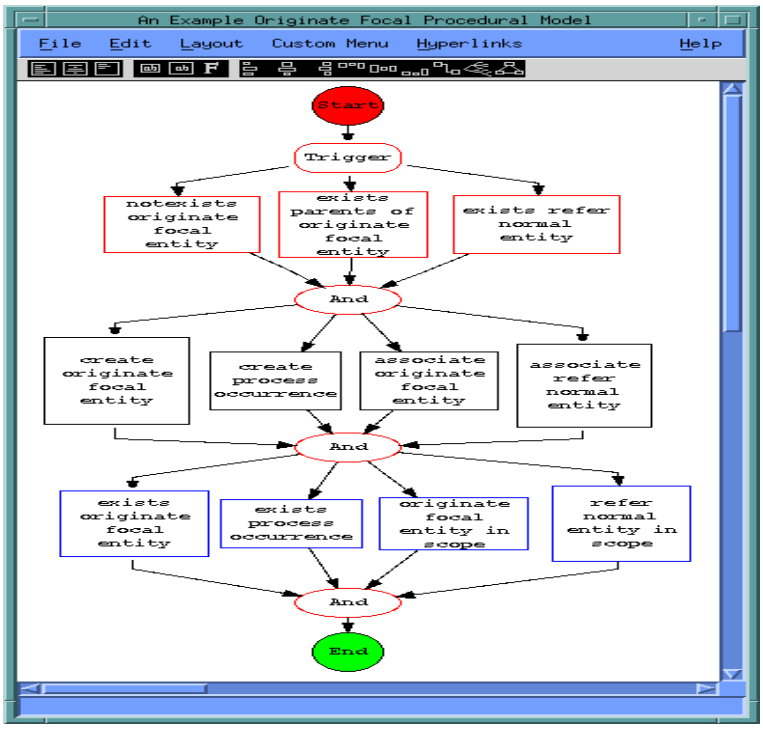

Figure 7. An Example Procedural Model for Originate Focal Process

distinguish itself from other triggers. The formal representation of a trigger and an example trigger with ID 1, which invokes a process called 'Module Performance Assessment' at time 3, is shown below (the action to be carried out is to create an occurrence of the originate focal entity of the process):

trigger(Begin_time, Process_name, Trigger_id, Action_list).
trigger(3, 'MODULE PERFORMANCE Assessment', 1, [originate_focal_entity]).

Preconditions for a process are requirements which make sure that process actions can be carried out successfully. Their existence is often linked to the actions to be carried out by a process. For instance, in Figure 7 the first precondition used is 'notexists originate focal entity', i.e. if a process is to generate its originate focal entity, then this entity must not already exist. This precondition is formally represented as:

precondition(originate_focal_entity, [notexists,originate_focal_entity]).

The representation of a postcondition is similar to that of a precondition; the predicate name is replaced with 'postcondition'.

8.1 *Representing Procedural Knowledge*

Model rules derived from the procedural model are primarily concerned with the operational aspect of a process execution. A process is normally carried out when its trigger is present and all of the precondition statements are satisfied. In some circumstances the process may only sometimes be applied, and therefore, a weak imply, $\triangleright$, is used.

The inference rule below states that *if* a trigger of a process is present, *then* if it is due or has passed due time and all preconditions of the process are satisfied, *then* all of the actions of the process should be put into effect in the next time unit (which becomes the actual end time of the process). An occurrence of this process is created with its begin and end time specified as part of these actions; the end time of a process is denoted by an *occ_end_time* predicate.

$$
\begin{array}{l}
class(process, Process\_name) \wedge \\
trigger(Begin\_time, Process\_name, Trigger\_Id, Process\_actions) \wedge \\
time\_cost(Process\_name, Time\_cost) \\
\Rightarrow \exists Time. \left(\begin{array}{l}
Time >= (Begin\_time + Time\_cost - 1) \wedge \\
\neg \left(\begin{array}{l} \exists Precondition. process\_precond(Process\_name, \\ \qquad Trigger\_Id, Precondition) \wedge \\ not\_valid(process\_precond(Process\_name, \\ \qquad Trigger\_Id, Precondition), Time) \end{array} \right) \\
\triangleright \\
\neg \left(\begin{array}{l} \exists Action. Action \in Process\_actions \wedge \\ no\_effect(process\_action(Process\_name, \\ \qquad Trigger\_Id, Action), Time + 1) \end{array} \right)
\end{array} \right)
\end{array}
$$

When a process occurrence finishes its actions at *End_time*, all of the corresponding postconditions of a process should be true. This is shown in the expression below.

$$
\begin{array}{l}
occ\_end\_time(Process\_name, Process\_Id, End\_time) \\
\triangleright \\
process\_postcond(Process\_name, ProcessId, Postcondition) \wedge \\
valid(process\_postcond(Process\_name, ProcessId, Postcondition), End\_time)
\end{array}
$$

8.2 *Inference and State Transition Diagram*

The inference engine generates the dynamic behaviour of the model, i.e. it simulates the execution of processes, according to the instructions given by the user. More specifically, the user selects a set of triggers for processes to be enacted. The scheduler of the inference engine then determines if any of the triggers in the previous set, or any of the new ones have a due time greater than or equal to the current system time. If so, the corresponding processes are added to a *process agenda*. For those processes in the agenda which have all of their preconditions fulfilled, the process occurrences are created and actions executed. This leads to a new system state. Corresponding feedback is given to the user and the system time advanced. The next cycle begins with the new system state and the user can decide on the next set of triggers. The model simulation ends when the user decides to quit the system. The inference engine is able to backtrack to any previous system state (going backwards in time), thereby allowing the user to experiment with alternative paths of execution of the model. The execution history is maintained in a state transition diagram, the details of which are represented by the underlying formalism.

Notice that a transition from one state to another could be the result of more than one process. We, therefore, assume that processes which are carried out at the same time are not in conflict with each other. To prevent conflicting processes from executing at the same time, process constraints can be deployed, but this issue is not discussed in this paper.

Figure 8 shows an example state transition diagram which demonstrates a possible sequence of states generated by the system. It uses the two processes specified in the process-entity matrix in section 7.

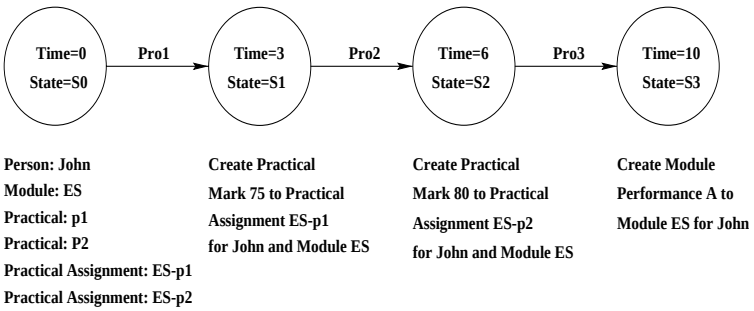

Figure 8. A State Transition Diagram for Originate Focal Process

The first circle denotes an initial state *S0* at time 0. Assuming that in the initial state we only have a few facts (occurrences): a person *John*, a module *ES*, two practicals, *p1* and *p2*, and their assignments to module *ES*: *ES-p1* and *ES-p2*. We also assume

that initially three triggers are given by the user: two invoke the 'PRACTICAL MARK Assignment' processes which assign the practical marks for *John* for *ES-p1* and *ES-p2*, and one trigger that invokes the process 'MODULE PERFORMANCE Assessment' which assigns the module performance for *John* and *ES*. Notice that in this example we have assumed all triggers are given up front at the initial state for simplicity, but those triggers can be specified dynamically by the user, since the reasoning engine is interactive.

The directed link *Pro1* denotes that process *Pro1* was executed. It transferred the initial state *S0* to state *S1*. Assuming that process *Pro1* ends at time 3, the newly created state *S1* is generated at time 3.

Process *Pro1* has assigned a practical mark 75 to practical assignment *ES-p1* for *John* and module *ES*. Process *Pro2* furthermore assigned practical mark 80 to practical assignment *ES-p2* for *John* and module *ES*. This process also transfers state *S1* to state *S2* denoted by a link labelled *Pro2*. Since process *Pro2* has ended at time 6, this has become the system time of the state. By the same reasoning, state *S3* is determined by a process *Pro3* which assigns a module performance A to *ES* for *John*, and the system time is 10.

9 Conclusion

In recent years the software development cycle has been extended with the creation of business models prior to the requirements analysis phase in order to avoid the production of software systems which do not fit the needs of an organisation. IBM's BSDM is one of the development methods which has embraced this idea of business models.

In this paper we have shown how key components of an informally specified business model can be formalised using first order predicate logic. Having achieved the move from an informal to a formal representation of models and modelling rules we were able to provide guidance and consistency checking during the life cycle of a model — making use of techniques such as case-based reasoning. It also allowed us to complement the original method with a model execution phase. This extends the scope of BSDM and, more importantly, it adds to our understanding of how this sort of seemingly informal method can fit into parts of the design lifecycle which require formal models.

References

[1] Alfs Berztiss. *Software Methods for Business Reengineering*. Springer-Verlag New York, Inc., 1996.

[2] Yun-Heh Chen-Burger. *A Support Tool For Business Modeling in BSDM*. In Expert System Specialist Group British Computer Society, editor, *Proceeding of Applications and Innovations in Expert Systems*, December 1995.

[3] John Dobson and Ros Strens. *Organizational Requirements Definition For Information Technology Systems*. Technical report, Department of Computing Science, University of Newcastle upon Tyne, NE1 7RU, 1992.

[4] International Organization for Standardization. Human centred design processes for interactive systems. *ISO DIS 13407*.

[5] John Fraser and Ann Macintosh. *Enterprise State of the Survey, The Enterprise Consortium*. Artificial Intelligence Applications Institute (AIAI), AIAI, The Univ. of Edinburgh,, 80 South Bridge, Edinburgh EH1 1HN, UK, September 1994.

[6] David Hollingsworth. *Workflow Management Coalition, The Workflow Reference Model*. Workflow Management Coalition, Avenue Marcel Thiry 204, 1200 Brussels, Belgium., 1994.

[7] IBM United Kindom Limited, IBM UK Ltd., 389 Chriswick High Road, London W4 4AL, England. *Business System Development Method, Introducing BSDM*, 2nd edition, May 1992.

[8] IBM United Kingdom Limited, IBM UK Ltd., 389 Chriswick High Road, London W4 4AL, England. *Business system Development Method: Business Mapping Part1: Entities*, 2nd edition, May 1992.

[9] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., 2929 Campus Drive, suite260, SanMateo, CA, USA, 1993.

[10] Jintae Lee, Michael Gruninger, Yan Jin, Thomas Malone, Austin Tate, Gregg Yost, and others. *The PIF Interchange Format and Framework*. The PIF Working Group, (Jintae Lee,Dept. of Decision Sciences,Univ. of Hawaii,2401 Maile Way.Honolulu), 1996.

[11] Thomas W. Malone, Kevin Crowston, Jintae Lee, Brian Pentland, Chrysanthos Dellarocas, George Wyner, John Quimby, Charley Osborne, and Abraham Bernstein. *Tools for inventing Organizations: Toward a handbook of organizational processes*. Centre for Coordination Science Massachusetts Institute of Technology.

[12] Richard J Mayer, Michael K Painter, and Paula S deWitte. *IDEF Family of Methods for Concurrent Engineering and Business Reengineering Applications*. Technical report, Knowledge Based Systems Inc., One KBSI Place, 1408 University Drive East, College Station, TX 77840-2335, USA, 1992.

[13] U.S. Department of Defense. *Military Standard: Defense System Software Development.* Dod-std-2167, June 1985.

[14] U.S. Department of Defense. *Leading Change in a New Era.* Technical report, May 1997.

[15] ProSci. *BPR Online Learning Center: the reengineering directory*, January 1999. Web page.

[16] 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.

[17] Pamela Zave and Michael Jackson. *Four Dark Corners of Requirements Engineer*. *ACM Transactions on software Engineering and Methodology*, Vol. 6(No. 1):page 1–30, January 1997.