Formal Support for Adaptive Workflow Systems in a Distributed Environment

Yun-Heh Chen-Burger and Jussi Stader

AIAI, the University of Edinburgh, UK

ABSTRACT

To achieve more widespread application, Workflow Management Systems (WfMS) need to be developed to operate in dynamic environments where they are expected to ensure that users are supported in performing flexible and creative tasks while maintaining organisational norms [Alonso et al, 1997; Sheth & Kochut, 1997]. In order to cope with these demands, the systems must provide knowledge about the business process itself and the organisational context in that these processes operate [Jarvis et al, 1999]. It, however, is not an easy task to provide the appropriate and sufficient knowledge at the right level of abstraction that supports a workflow system at all stages of operation in a dynamic environment and for different types of users.

At the same time, Enterprise Modelling (EM) methods are well recognised for their value in describing complex domains in an organised but usually informal structure. In particular, business process modelling techniques provide rich conceptualisations that tend to describe the type of information required by the adaptive workflow systems. However, because of their lack of formal structure the use of Enterprise Models that have been developed is limited [Junginger, 2000][Chen-Burger, 2001a].

We propose the use of a formal language within a three-layered framework. This language helps to turn the information contained in an informal Enterprise Model into the kind of formal model required by an adaptive Workflow System. In its current state of development, FBPML (Fundamental Business Process Modelling Language) covers business processes, organisational structure, agents and their capabilities as well as execution logic that gives direct instructions to a workflow engine.

We assist modelling efforts of Enterprise Modellers by giving them a visual modelling language, underpinned by a formal representation, that is expressive and easy to use and that lets them specify the information required by a workflow engine. In this paper, we present our formal enterprise modelling language, FBPML. We show how adaptive workflow systems, like those developed at AIAI (e.g. the Task Based Process Manager [Stader et al, 2000], AKT Workflow [Chen-Burger, 2002a] and I-X system [Tate, 2002]), can take advantage of Enterprise Models represented in FBPML to provide effective support to users in real business environments.

INTRODUCTION

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, BSDM (Business System Development Method, IBM) [IBM, 1992], process modelling methods, IDEF0 [NIST, 1993], IDEF3 [Mayer, 1992], PSL [Schlenoff, 2000], PIF [Lee, 1998], RAD [Ould, 1995], RACD [AOEM, 2001], CommonKADS Communication Model Language [Waern, 1993], organisational modelling, Ordit [Dobson et al, 1994], Ulrich [Ulrich, 2000], capability modelling [Jarvis et al, 1999]. EM methods can be used well in conjunction with ontologies such as Enterprise Ontologies [Uschold, 1996, 1998; Fox, 1998; Stader, 1996], DAML-S[Ankolekar, 2002].

Despite their widespread use, Enterprise Models do not often provide direct input for software system development. Obstacles include the necessary training required for users to learn conceptual modelling in general as well as the techniques required for the specific method applied. Generic knowledge acquisition techniques are also needed to elicit knowledge from the application domain. However, the main obstacle is the lack of direct mapping from EM methods to software system development. Since EM methods are normally described at high levels of abstraction that 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 Enterprise Models built are natural "blueprints" 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, to bridge the gap by providing direct mappings between Enterprise Models and designing and building of software systems: Quality Assurance, Mapping of Data Structure and Workflow System. These are all based on formal methods.

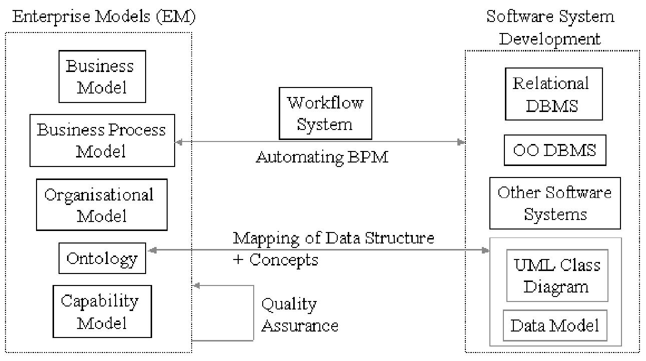

Figure 1: The Gap between Enterprise Models and their Applications

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 or more familiar) forms, or to provide simulation functionality to allow the reader to run through possible user scenarios in a state machine [Chen-Burger, 2001a; Robertson, 1999]. Automatic support for knowledge sharing or inconsistency checking between different Enterprise Models may also be provided based on a shared ontology [Chen-Burger, 2002c].

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, the quality of the models built is improved. The support for the refinement process 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 ontologies, to software systems. This may be done by mapping ontologies to Entity-Relationship Models (for Relational Databases), Class Diagrams (for Object-Oriented Databases) or other types of data structures. This transfer support is indicated by the "Mapping of Data Structure" arrow.

This paper focuses on Business Process Modelling (BPM) and its use in a workflow management system. The BPM approach towards building a workflow system is a recent and gradual approach, developed over the past three years. This is an advance from the first generation workflow systems, which did not use BPM [Delphi, 2002], because of the lack of overlap between the workflow community and the EM community. Thus, the business processes implemented in first-generation workflow systems do not separate business logic and implementation logic, and hence, are not flexible in their reaction to the dynamic and volatile environment within which they operate.

In addition, while BPM methods are normally described at a high level of abstraction that enables flexibility for implementation, they do not provide sufficient details for process enactment. It is therefore beneficial to provide a framework that maintains the flexibility of high-level descriptions, while also providing sufficient information to carry out workflow [Junginger et al, 2000].

This paper proposes a layered BPM framework that separates logic and implementation design of a workflow system by adopting a one-to-many coupling mechanism: one logic design may be implemented in many different ways. It also allows tractability back to business requirements for any implemented business processes.

The paper also describes the visual FBPML (Fundamental Business Process Modelling Language) that is underpinned by a formal language, and how this formal language provides the necessary expressiveness to give direct instructions to develop workflow management systems.

Our approach is based on an ontology which holds contextual information about data that is manipulated by processes and can be mapped to commonly used data or class models. Finally, we demonstrate this approach through an application that is within a virtual organisation where distributed entities have to collaborate with each other to achieve common organisational goals.

The Three-Layered Business Process Modelling Approach

Our three-layered business process modelling approach (Figure 2) amends some of the pitfalls that result from conventional waterfall design methods. The approach was developed independently by AIAI, and its value is recognised in AIAI's various commercial projects, however it can be loosely mapped to the top and bottom layers of the three graphs described in [Junginger, et al, 2000].

The approach supports the development of workflow systems from business process models and provides the means to describe higher level business processes, objectives and policies. It also provides tractable requirements for building business systems that include software and manual procedures.

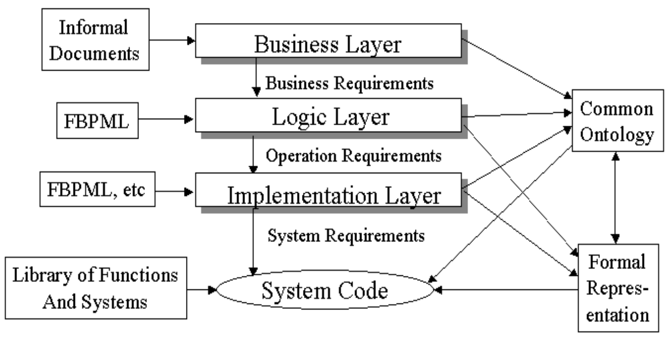

Figure 2: The Ontology-Based Three-Layered BPM Approach

The three layers of the framework are described below:

- The business layer allows business personnel to describe business operations and requirements in a language that is understandable for them. It also allows principles to be realised and carried out by automated and manual procedures. Information that is stored in this layer includes business policies and the context of an organisation, processes that are to be carried out by the organisation and information used by these processes. It contains high-level descriptions that may be written in informal or semi-formal documents. The business-level specifications that are stored here are robust aginst changes in the economic environment, organisational structure and technologyies in the long-term.

- The logic layer gives a logical description of business processes that obeys and keeps track of business principles and requirements that have been described in the business layer. This description depicts the conditions and actions of processes, the relationships and constraints between them and the data upon which the processes operate. The Logical Layer is a semi-formal BPM that is both visual and form-based and can be understood by business personnel. The visual process models in this layer are underpinned by a formal language. Once a process model has been specified the underlying formal representation of the process can be derived automatically. This formal representation can be used to check against errors in the model, it provides a basis for offering advice and a foundation for forcasting organisational behaviour. Processes described in this layer are relatively independent from the deployed technologies, including software and hardware systems, and therefore are more robust compared to processes in the implementation layer. The process modelling language, FBPML, that contains both informal and formal descriptions resides in this layer and will be described in more detail in the next section.
- The implementation layer gives detailed step-by-step algorithmic procedures for software modules and agents that implement processes described in the logical layer. The implementation layer tends to be technology-dependent: a process that has been described in the logic layer may be specialised in different procedures when different methods and hardware have been deployed. For instance, a "get customer payment" process may be implemented differently when it takes place over the phone or via form-based web access. Procedures in this layer may change frequently. The 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 almost certainly cause a modification of the corresponding descriptions in the implementation layer. This layer is also described using FBPML, but other languages may be used in conjunction with FBPML, e.g. other process modelling languages, flow-control diagrams, data-flow diagrams.

Information that is referenced by processes in the logic layer or the implementation layer is stored in a Domain Ontology. The Domain Ontology gives labels and semantics for the information and organises it in a taxonomy. In FBPML this information is encoded using the "class" predicate in the data language. As the ontology is shared across processes, it can be used to coordinate process execution. In the "get customer payment" example above, a process instance in the implementation layer may have been activated by the customer via the form-based web access. If an error occurs that the customer does not know how to handle, the customer may phone in which would invoke the alternative "phone-in" process. If the processes are coordinated, information already gathered from the customer during the web-based process can be used by the phone-in process without having to ask the customer again. Also, knowing that the two processes are alternatives for carrying out the same logic process, a workflow system can be aware of the fact that only one of them needs to be finished.

The system code provides the actual functions and systems that carry out the workflow. In this framework, the formal language that underpins the visual FBPML can be mapped directly to system code. This provides a direct link between the conceptual design of a workflow system and the actual, implemented workflow system. Functions and systems correspond to procedures in the implementation layer, which in turn correspond to FBPML terms used in their triggers, preconditions and actions. However, functions and systems are treated as black boxes and are replaceable. When new functions are introduced to the overall system, new mappings to the implementation layer can be established so that the new functions may be used directly via the business processes.

This direct mapping between processes and system code enables a flexible, open architecture. When a business process is modified its underlying formal model is also automatically modified. Because this formal model is mapped directly to the actual system code the different but appropriate workflow components will immediately be part of the new workflow system. In effect, a new workflow system has been constructed by re-arranging workflow components. The interactions between different functions and sub-systems are managed through the design of processes and the common underlying ontology. When new functions are required by new processes, such functions can be implemented and added to the library. Once appropriate links are established via the new process models, the new functions will be part of the new workflow system.

In summary, our layered modelling approach is a formal, ontology-based approach that aims to achieve the goals below:

- To provide a means to describe business requirements seperate from technological requirements and to allow such requirements to be passed down to and followed in the design stages of workflow system development. These business requirements can also provide justifications and rationale for the design;
- To separate the logical and technological design of a workflow system, i.e. for one logical process several lower level implementation-dependent procedures may be described;
- To provide automatic construction of workflow systems by linking business procedures to the appropriate system code, therefore a workflow system may be constructed at design time;
- To separate the actual implemented system from its design but maintaing a semantic link, so that workflow components may be upgraded while maintaining the same business operations; on the other hand, when the design of the system is changed the actual system is also changed automatically;
- To remove rigid requirements on process execution sequence to allow sharing of common data and flexible process management, especially the coordination of alternative processes that deploy different devices or systems.

The section below gives an informal account of FBPML (Fundamental Business Process Modelling Language).

THE FORMAL LANGUAGE FBPML

FBPML adapts and merges two recognised process modelling languages: PSL and IDEF3. PSL provides formal semantics for commonly shared process modelling concepts as well as theories, such as situation calculus, that support the use of such concepts. As it is designed to be an interchange language between different process languages, it covers the core concepts required for process models, but does not provide visual notations or model development methods.

IDEF3 originates from the manufacturing environment and is one of the richest methods available for process modelling. It provides visual notations and a rich modelling method. Nevertheless, its semantic is informal and its models therefore may be open to interpretation.

Combining the two different methods, FBPML retains IDEF3's rich visual and modelling methods and provides formal semantics and theories of PSL, so that reasoning mechanisms and formal analyses can be performed on those models.

FBPML has two sections to provide theories and formal representations for describing processes and data: the Data Language and the Process Language. Details of those are described below.

The FBPML Data Language

The FBPML Data Language (FBPML-DL) has a strong basis in logic [Chen-Burger, 2002b]. It is based on first-order predicate logic and set theory [Robertson, 1992; Bundy, 1983]. Mathematical theory on the manipulation of integer, rational and real numbers is also included. The language is presented using predicates like those of the programming language Prolog.

FBPML-DL has four parts:

1. **Foundational Model** provides concepts, predicates and functions of background theories that are used in the language. The primitive predicates provided here are used to define other predicates in the other parts of FBPML.
2. **Core Data Language** introduces core predicates and functions for concepts that are common to many applications. Their semantic is defined using constructs from the Foundational Model.
3. **Extension Data Language** includes predicates and functions that are additional to the Core Data Language and are often application and domain-dependent.
4. **Meta-predicates** may define axioms of an application model.

Foundational Model

The Foundational model provides the building blocks for the other parts of FBPML. It includes fundamental theories that are required for formal descriptions of process models. Its definitions include:

- Specifications and notations of basic concepts such as numbers, constants, logical and mathematical functions, variables, terms, lists, logic constants, connectors, quantification and constructs. E.g.
  - a valid Variable is denoted by an unbroken sequence of alphanumeric characters or underscore ('_'), that starts with an upper-case character or an underscore.
  - a valid Term is either a constant, variable, number, list, string, or another predicate that has been defined in FBPML-DL.
- The logical operators "`and`", "`or`", and "`not`". Logical inference is also in-cluded, but is embedded in the structure of predicates, i.e. the constraint and axiom predicates that will be introduced in the core language and meta-predicates.
- The basic mathematical operators `=`, `<`, `>`, `>=` and `=<`.
- Basic functions that check characteristics of their arguments, e.g. `atom(Term)`, `is_list(Term)`, `compound(Term)`. E.g.
  - `is_list(X)` returns `true` if X is a valid list in FBPML, `false` otherwise.
- Logic quantifications `forall`, `exists` and `not_exists`
- Logical constants `true` and `false`.

Core Data Language

The core data language provides common concepts that are used in all application areas. Their definitions use foundational or core FBPML predicates. The core data language includes:

- Basic concepts of set theory such as class, instance, property name, property value, attribute name and attribute value. E.g.:
  - attribute_name(X) specifies that X is an attribute name for instances, where X is a string of characters that may include underscores or hyphens.

- Predicates commonly used in object-oriented languages that implement more complex concepts and relationships of set theory like subclass of, instance of, class property, class relationship, instance attribute, attribute domain, instance relationship. E.g.:
  - instance_of(Instance, Class) specifies that Instance is an instance of class Class. Formally, this is described as:
    instance_of(Instance, Class)
    ⇒ instance(Instance) ∧ class(Class)
  - instance_att(Instance, Attribute, Value) specifies that an Instance, instance, has an Attribute, attribute, whose value is Value.
- Two types of constraints: static and dynamic constraints. Static constraints define properties that a valid goal state *must* have for a designated problem domain, whereas dynamic constraints indicate properties that the goal state *may* have.

Extension for the Data Language

In the Extension for the Data Language, all application specific predicates are implemented. For instance, in the AKT project a set of predicates to describe the PC configuration domain has been defined. They define the domain-specific classes, relationships, functions and constraints. For instance, classes such as motherboard and different types of boards (I/O board, disk controller, etc.) are defined, together with the relationships between these classes, such as the "part-of" relationship. Customer constraints are defined in terms of dynamic or soft constraints, e.g.

dynamic_constraint([forall(M)], [total_cost(M)], [less_than(M, 1000)])

denotes that the total cost of a chosen configuration should be less than 1000 pounds.

Meta-predicates

The meta-predicates give definitions for other predicates and may define axioms of an application model. There are two constructs that can be used here:

- def_predicate(Predicate, Predicate_definition) defines a predicate. The variable Predicate is the predicate name, Predicate_definition is a list that defines the semantics of the predicate using conjunctive normal form in first-order predicate logic.
- axiom(Conclusion, Hypothesis) defines an axiom, specifying that if the hypothesis is true, the conclusion is also true. This is used to define the properties of a predicate; it does not define a new predicate. In FBPML, each axiom is a horn-clause, i.e. there is only one single conclusion. The Hypothesis is written in conjunctive normal form in the same way as the Predicate_definition above.

The FBPML Process Language

The FBPML Process Language (FBPML-PL) contains two parts: the formal language and its visual presentation. Its important concepts are: activity, activity decomposition, triggers, conditions, temporal constraints, junctions, roles, time points, synchronisation, and notes. These concepts are described below followed by an example of a visual model and its formal representations.

The main concept of any process modelling language is the process itself. Different languages have different names for processes (e.g. activity (PSL), unit of behaviour (IDEF3), or task in different languages). FBPML adopts the convention of PSL and uses "Activity" to capture the concept of a step in a process. A process is recognised by most process modelling languages as a sequence of activities that may last for a period of time.

Activity decomposition: Some process modelling languages capture processes at only one level of abstraction, i.e. no decomposition of a process is available. The modeller must decide which level of abstraction is right for the domain and apply that level of abstraction throughout the modelling exercise. Such an approach is used in IBM's Business Modelling Method in BSDM, RAD, RACD and the conventional control-flow and data-flow diagrams.

Other languages allow a process to be described at different levels of abstraction. The modeller may decide to explore details of certain parts of the process. Examples of such languages are IDEF0, IDEF3, PIF, PSL and DAML-S. FBPML describes its processes in its Activity nodes. Like IDEF3, the activity nodes may themselves be decomposed into more detailed activity nodes. There may also be alternative ways to decompose an activity, which correspond to different ways of carrying out the activity. When a process has been decomposed into several sub-processes, by definition, the completion of *all* of the sub-processes completes the higher level process. On the other hand, where there are alternative decompositions for one higher-level process, completion of *one* of those alternative processes completes the higher-level process. An activity that has reached the lowest level of detail and may not be decomposed any further is denoted by a Primitive Activity. Our formal semantics of activity, primitive activity and decomposition follow the ones defined by PSL. The semantics of alternative processes is similar to that informally described by IDEF3.

Representing an Activity: IDEF3 allows informal descriptions of activities. This is not sufficient to support automation, so in FBPML, an activity is formally defined by a tuple of five characteristics underpinned by a formal representation. This tuple is:

activity(Position, Activity_name, Triggers, Preconditions, Actions)

where Position indicates the position of an activity in a process model that encodes the information and whether it is a decomposition or alternative-decomposition of its "parent" process. Activity_name is a unique string that identifies this activity. Triggers include events and conditions that invoke the activity. Preconditions are premises that must be true before an activity may be carried out. Actions is a list of execution instructions that declaratively describe actions to be carried out. This tuple does not include postconditions, although they are captured in a process model. This is because postconditions are derivable from the Triggers, Preconditions and Actions; they do not provide additional information to uniquely identify a process. When implemented, this five tuple is instantiated and a process predicate is generated that holds additional information required for running a workflow system. Details of this executable process predicate will be described later in this section.

IBM's Business Modelling Method, BSDM, classifies activities according to their purpose into three main types: originate, change and refer, i.e. activities whose purpose is to create information, update information, or to refer to existing information. In addition to these types, FBPML provides: decomposition, communication and auditing activities. Decomposition activities refer to the next level of processes. Communication processes specify functions that interface with the user or other software. Auditing activities are processes that monitor system dynamics and react appropriately when required. Primitive actions that may be carried out by those processes are database manipulation actions, such as create, delete, update and refer_to; communication facilities, such as read_user_input, report, create, send and receive issues; mathematical and user-definable functions.

Event, Trigger and Enactment: Events are external or internal incidents that happen in a WfMS. The occurrence of an event may match a trigger's condition and therefore invoke a process. During process enactment, all the triggers and preconditions of an activity must be true before its actions may be carried out. Different from IDEF3, FBPML separates triggers from preconditions in a process and gives them activation semantics. That is to say if all of its triggers are true then a process must be executed at some point in time (but not until all its preconditions are true). This enables precise control of delayed process enactment.

Temporal precedence: a precedence-link expresses a temporal constraint between two activities. The specification that activity A is preceded by activity B indicates the execution of activity B must NOT start before the execution of activity A is finished. In FBPML, it means that when activity A is finished, activity B becomes "Temporally Qualified" and can thus be considered for execution. Temporal qualification is a concept that FBPML introduces to provide more precise execution instructions for workflow system operation. The formal definition of Temporal Qualification of a dependency link is:

$$\forall Act\_a, Act\_b, A, Path.preceded\_by(Act\_a, Act\_b)$$

$$\Rightarrow \begin{pmatrix} instance\_of(A, Act\_a) \wedge activation(A, Path) \wedge end(A) \\ \Rightarrow \exists B.instance(B, Act\_b) \\ \wedge\ activation(B, path) \wedge temporal\_qualified(B) \end{pmatrix}$$

where the predicate preceded_by(Act_a, Act_b) indicates activity Act_a is preceded by activity Act_b in a process model. This precedence link means that only upon the end of execution of a process instance A of activity Act_a will grant the existence and the temporal qualification of another process instance B of activity Act_b.

Role: A role represents the role of an individual agent or the role of a group of agents in an organisation. An agent may be a person or piece of software. FBPML allows activities to be grouped by roles. Each role can carry out certain activities, have capabilities and authorities and is related to other roles. The tasks that a role carries out are described using FBPML activities. Besides its own activities, a role may need to communicate with other roles to accomplish its tasks, giving rise to communication processes. The introduction of roles separates the logic of agents from their real-life incarnations. While activities can be assigned to roles at design-time, they will not be assigned to individual agents until they are about to be performed. This means that the availability and suitability of an agent can be determined at run-time and the most suitable agent can be chosen.

Table 1: Higher Level Capability Hierarchy

Technical Capability (Entity)	**Cognitive Capability**	**Business Capability (Entity)**
➢**Analysis (Device)**	➢**Understanding**	➢Marketing
•Requirements Analysis	➢**Reasoning**	•Customer Liaison
•Hazard Analysis	➢**Creativity**	➢Commercial
•Quality Analysis	➢**Knowledge**	•Contracts
•Testability Analysis	➢**Authoring (Document)**	•Bidding
➢**Testing**	➢**Management (Device)**	•Commissioning
➢**Review**	➢**Development (Device)**	•Procurement
➢**Assessment**	**Physical Capability (Entity)**	➢Personnel
➢**Modelling**	➢**Manual Capability**	➢Financial
➢**Engineering**	•Repair	
➢**Design**	•Removal	
➢**Integration**	•Fitting	
➢**Maintenance**	•Connecting	
➢**Use**	•Replacement	
IT-Capability	➢**Sensory Capability**	
➢**Database Capability (Data)**	➢**Project Capability (Project)**	
•Storage	•Management	
•Structured Storage	➢Planning	
•Hierarchical Storage	➢Organising	
•Relational Storage	➢Controlling	
•Unstructured Storage	➢**Communication (Entity)**	
•Retrieval	•Request	
•Search	•Respond	
•Calculation	•Inform	
•Simulation	➢**Co-operation (Entity)**	

Capability: a capability is required for a role or an activity, or provided (held) by an agent. To allow capabilities to be specified and matched effectively, they should use well-defined terms taken from an ontology. For ease of use, it helps if this ontology is organised onto a hierarchy. However, in our experience organising all terms required for specifying capabilities of agents and activities into such a hierarchy is too big a task for any realistic application area. We decided to impose more structure by splitting the specifications into two parts: the capability itself and the area (or "knowledge space") in which the capability can be applied: capability(Capability, Knowledge_space). For example, if a specific database application can store data about technical reports, this may be represented as capability(store, technical reports) Each of the parts uses its own hierarchy of terms. An example hierarchy of capabilities is shown in part in . For more detail, see [Uschold, 1998].

By providing a well-defined ontology of capability and knowledge space terms, statements about capabilities can be made consistently and matched effectively. The use of a generalisation structure within the ontology simplifies the specification of capabilities because specifying a high-level capability implies that all its lower-levels are covered too. It can also be exploited by a workflow system to apply heuristics such as "generalist vs. specialist" and make the best use of the agents available during a task's execution.

Junction: A junction is a control point in a process. There are four types of junctions: "start", "finish", "and" and "or". The start and finish junctions define the starting and finishing points of a process. The "and" and "or" junctions define a one-to-many relationship between connected activities and indicate conjunction and disjunction points of an overall process. They can be classified as fan-in (join) or fan-out (split) points of a model. Figure 3 shows the four basic uses of junctions: and-join, or-join, and-split and or-split.

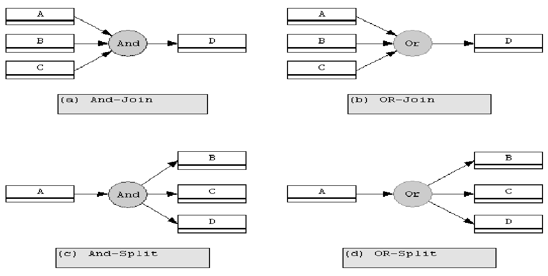

Figure 3: The Four Basic Types of Junctions[1]

FBPML's execution semantics of junctions is more expressive and more precise than that of IDEF3 or PSL. In FBPML, a join indicates that more than one activity precedes the junction and only one activity follows the junction. Semantically, an and-join indicates that all of the *triggered* preceding activities must be finished before the following activity may be executed. That is to say, in Figure 3a, upon the finishing of all triggered activities (A, B, C), activity D is temporally qualified and may be considered to be executed. However, if only activity B and C are triggered, then only activity B and C need to be finished before activity C may be executed. An or-join, on the other hand, indicates only one of the triggered preceding activities is required to be finished before the following activity is temporally qualified and executed.

A split indicates only one activity precedes the junction and more than one activity follows it. Semantically, an and-split indicates that all of the activities that follow the junction must be triggered (and executed at some point) after the preceding activity is finished. An or-split indicates that at least one of the following activities

[1] This is a screen capture of KBST-EM that was developed by AIAI, The University of Edinburgh. KBST-EM is based on Hardy, a diagram-programming platform, that is also built by AIAI.

is required to be triggered (and executed) after the preceding activity is finished. If there are any activities following an and-split that are supposed to be triggered but aren't, this is an issue that the workflow system needs to resolve, as it is inconsistent with the execution semantics specified in the process model.

Time point: FBPML allows the modeller to specify time points within and outside of processes. A time point is typically associated with a process, e.g. the start or finish point of a process. Time points are used to allow accurate coordination and control between processes.

Synchronisation-Bar: A synchronisation-bar places a temporal equivalence between two time points. It is mainly used to synchronise events and co-ordinate processes.

Idea and Navigation Note: The Idea Note records information that is relevant to the model but is not formally part of the model. Examples of idea notes are design rationale or a reminder for future refinement of the model. A Navigation Note records the relationships between diagrams in a model. Both notes are annotation nodes that are typically informal and temporary, and are therefore not part of the formal model.

Executable Process Predicate: Although a process may be defined and uniquely identified using the activity-tuple, as described earlier, this tuple does not contain sufficient information to support workflow enactment and is therefore extended with necessary operational information in a 11-argumented process (instance) predicate.

Table 2: The Generic Template for Describing a Process (Instance)

```
process(-Instance_ID, +Process_name/ID, -Process_Status, -Process_Priority,
         -Begin_time/-End_time, +Duration,
        (?Service_requestor_id/?Requestor_type,
                ?Service_provider_id/?Provier_type),
        +Triggers, +Preconditions, +Actions, +Postconditions).
```

Table 2 shows the generic template for the process predicate that provides the formal representation of a process in FBPML. It stores the information that has been determined during the modelling phase and will not be altered during execution. It also provides place-holders to keep dynamic information that will be instantiated and modified at run-time. We use the plus sign '+' in front of a variable to indicate information that must be provided at design time; a minus sign '-' to indicate the value will be provided at run-time. Variables that begin with a question mark indicate their value may be available either at design or run-time. The example in the next section explains more detail of the formal process specification.

VISUAL LANGUAGE, ITS FORMAL REPRESENTATION AND EXECUTION LOGIC

Describing a process using FBPML normally starts with identifying the beginning of the process and triggering events that invoke the process. Then the activities that carry out the process are defined, before the branching points between activities and the ending point of the process are given. Activities of a process are normally described at the same level of abstraction and some of them may need to be decomposed into sub-activities to give a more detailed description of the actions. Figure 4 shows a simplified decomposition for the "Generate New PC Configuration" process using the visual notation of FBPML [Chen-Burger, 2002a].

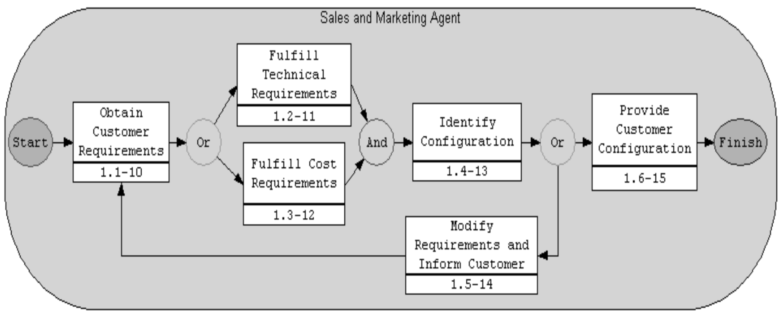
Sales and Marketing Agent
Start
Obtain Customer Requirements
1.1-10
Or
Fulfill Technical Requirements
1.2-11
Fulfill Cost Requirements
1.3-12
And
Identify Configuration
1.4-13
Or
Provide Customer Configuration
1.6-15
Finish
Modify Requirements and Inform Customer
1.5-14

Figure 4: Decomposition for the "Generate New PC Congifuration" Process[5]

Activities are shown as white box nodes, junctions are circular nodes with their types written in them, and precedence links are shown as directional arcs between nodes. Besides the visual notation shown in diagrams, there is additional information that needs to be specified for effective workflow management. Based on that information and the diagrammatic model, a formal representation of the above model can be derived.

The start and finish junctions are formally represented by start(10) and finish(15) where number 10 and 15 are the unique IDs for activities "Obtain User Requirements" and "Provide Customer Configuration" for they are the starting and finishing activities. The first or-junction in the model is an or-split and is represented formally by or_split(10,[11,12]), as its one preceding activity is activity 10 and it has two following activities, activity 11 and 12. Similarly, the following and-join and or-split are represented as and-join([11, 12], 13) and or-split(13, [14, 15]). The last or-split indicates a loop in the process model. The decision whether a loop-back is required depends upon the triggers of activity 14 and 15. The (loop-back) dependency link is denoted by control_link(14, 10).

Table 3 shows a formal description of the activity "Obtain Customer Requirements", which is an instantiation of the generic template for processes given earlier in Table 2. In the formal description, variables start with a capital letter or a "_", whereas constants start with a lower-case letter. Variables will be instantiated at run-time when the activity is performed.

Table 3: Formal Description for Activity "Obtain Customer Requirements"

```
process(Instance_ID, 'Obtain Customer Requirements'/1, Status, Priority,
        Begin_time/End_time, 1,
        (Requester/Requester_type, edinburgh/pc_specification),
        [and(event_occ(Event_instance,
                        customer_request_for_pc_specification,
                        received/Processes, Priority, _Begin/_End,
                        (Requester/Requester_type, Provider/Provider_type),
                        _Event_content),
             not_exist(instance_att(Event_instance,
                                    finalised_configuration, _solution)) ) ],
        [true],
        [cond_action([not_exist(instance_of(Requester, customer))],
                     [create(instance_of(Requester, customer))] ),
         create(instance_of(Instance, event)),
         create(instance_att(Requester, event, Instance))  ],
        [exist(instance_of(Requester, customer)),
         exist(instance_of(Instance, event)),
          exist(instance_att(Requester, event, Instance))   ] ).
```

The first variable, `Instance_ID`, holds the identifier for a process instance that is to be created for this process at run time. The second variable, `Process_name/ID`, holds the unique `Process_name` and its short-hand numerical `ID`. The process name is 'Obtain Customer Requirements' and its ID is ‘10’. The life status of the process instance and the priority for the process will be assigned at run-time and are therefore denoted by variables, `Status` and `Priority`.

The next variables `Begin_time/End_time` hold the start and finish time of a process instance. Since this cannot be known before execution, they are both variables. The fourth variable, `Duration`, holds the expected or average duration for the process. In this case, it is 1 time unit. The actual duration of a process instance is derived from the `Begin_time` and `End_time` arguments. The fifth variable holds information about the service requestor and provider. In this example, the service provider is the Edinburgh site and the provider type is pc-specification. The service is not limited to anyone or type.

The `Triggers` variable holds conditional statements and/or event occurrences that invoke the process. In this case, it is the event of receiving a customer’s request for PC configuration and the fact that a valid solution has not been reported back to the customer yet. Since there is no additional precondition required, the ‘preconditions’ variable is set to ‘true’. There are three actions that this process carries out. Firstly, there is a conditional action to record information of the particular requestor, i.e. if it has not already been stored. Then a new instance for the request is created and information is stored about the request. The last variable, `Postconditions`, stores conditions that must be true when actions are finished. In this case, it is the storage of the customer and the request. The actions that are stored in the `Actions` variable are the lowest level operations of a workflow system may carry out and they will be carried out sequentially. However, if one wishes to describe more complicated activities, one may further decompose the activity.

The example in Table 3 is a simple one and it is possible to define more complicated processes in FBPML. The FBPML-PL constructs are composed using the FBPML-DL plus its specific vocabularies for describing processes, events, actions, conditions, lifecycle status and communication facilities. As FBPML provides precise execution logic, when a workflow system interprets it correctly, its originally static description for processes can be put into dynamic actions in a straightforward way. This is done with a direct mapping between the logic process description of and the modules (or agents) of a workflow system. The lowest-level actions of a process must be mapped correctly to the workflow modules. In this case, the conditional statement, denoted by the cond_action predicate, and the creation of information, i.e. action predicates denoted by “create”, are mapped to the corresponding functions. In addition, descriptions of data structures must be mapped to those of the workflow system and the database systems that may be used. In this case, mappings are required for the occurrence of event, the event_occ predicate, domain specific information, denoted by instance_of predicates, and their attributes, the instance_att predicates. This mapping can be achieved because the meta-model of FBPML-DL is compatible with Object-Oriented Classes and Relational Entities. Logic predicates, such as “and”, “or”, “exist”, “not_exist”, “forall”, are part of the FBPML workflow language, therefore are understood and interpreted by the workflow system correctly.

These facilities make it possible to change the behaviour of a workflow system by changing its design specification. This also ensures a higher degree of consistency in executing processes because the functionality required by processes is checked against available functionality at run-time. In more static systems, execution logic is determined by systems that implement it relying on facilities that are available to them. The execution therefore may not always be consistent and may lead to potential errors.

Our design-based workflow system is also scalable. When new functionality is introduced by the process model (consequently by adding new action predicates), new execution modules must also be introduced into the library of the workflow system with the proper link to the design. Provided this is done, the new actions specified in the process model can be matched with the new execution modules at run-time and be carried out normally. Under careful management of workflow, allowing such changes to be made at runtime enables dynamic and rapid change of process execution that is under proper control. This is feasible because process specifications can be verified and validated even when the process design phase is overlapped with the system deployment phase[Chen-Burger, 2001a] or even with the process execution phase [Jarvis, 1999].

OPEN ARCHITECTURE OF THE WORKFLOW ENGINE

Figure 5 describes a workflow engine implemented at AIAI that uses FBPML. The workflow engine is a combination of two things: a manager for handling the execution of workflow and a meta-interpreter for descriptions of processes and data. Equipped with the appropriate workflow algorithm, the workflow engine periodically retrieves new events that occur, identifies those processes that have been specified in the process model that are relevant to the new events. It examines the triggers of these processes, creates process instances for those whose triggers are true and puts them on the Process Agenda. The workflow engine also looks for discrepancies between the process models specified at design time and the dynamic activation of the models. Any discrepancies found will be reported to the user together with advice for correction.

The Process Agenda stores a list of process instances that are waiting to be executed. The workflow engine interprets the preconditions of these processes and puts those that are ready for execution in an execution queue. The workflow engine then carries out a consistency check on all processes on the execution queue, because the processes to be carried out simultaneously may produce a deadlock or conflicting results. If any inconsistencies are found, the user is notified together with correction advice, and resolutions are expected to be provided by the user.

When all inconsistencies are resolved, the engine induces the appropriate (internal or external) execution capabilities to carry out the appropriate processes after checking that both their triggers and their preconditions are still true. It also monitors and manages pending processes on the agenda and will raise an alarm if a process has been idle in the Process Agenda for a long period of time.

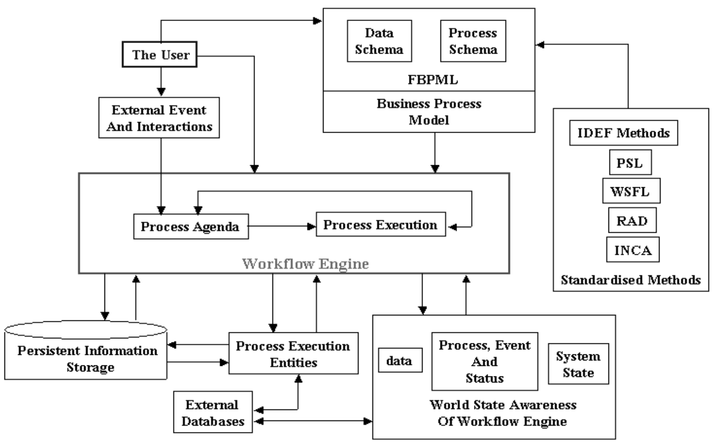

Figure 5: Internal View of Workflow Engine

The workflow engine can correctly map activities described in a process to the appropriate Process Execution Entities because it can match the process descriptions (Process Schema), written using FBPML, against the capabilities specified for the Execution Entities. It also advises Process Execution Entities which data to operate upon, because the (external) data that those Execution Entities use is conceptually mapped to the domain ontology (Data Schema) of the workflow engine. The workflow engine maintains a World State Awareness which stores the temporary status of processes under execution. It also keeps a log of execution progress. The execution log and World State Awareness may be archived in its Persistent Information Storage.

A Process Execution Entity may be a software module, a system, a sub-system or an internal or external agent in a distributed environment. In this open architecture Process Execution Entities can be replaced in a plug-and-play style so that the capabilities of the overall workflow system can evolve and be easily reconfigured. The link between the execution entities and capabilities of the workflow engine is provided by the mapping through process models using FBPML. It is this mapping that enables the flexible coupling between the workflow engine and the execution entities. The link between FBPML and the standardised methods on which FBPML is based (IDEF3, PSL, WSFL and RAD and INCA [Tate, 2002]) is indicated by the top right corner in Figure 5.

The next two sub-sections describe two environments in which the open architecture workflow engine has been used: a distributed environment and a collaboration environment.

Workflow in a Distributed Environment

As mentioned in the previous sub-section, a Process Execution Entity may be a module, system, or indeed an agent in a virtual environment. Figure 6 shows how a workflow system may become an agent itself and operate in a distributed network, making use of internal or external process execution capabilities that are available in the network.

We have implemented a scenario where two copies of the workflow system reside and operate in two different organisations, each equipped with its own processes and workflow management capabilities. At run time, when a workflow system has appropriate process execution entities for its tasks at hand, it may choose to allocate the tasks directly to them. Such execution entities may be an agent inside or outside of its own organisation. When it is less clear who the appropriate execution entities may be, the workflow system may choose to send a request for task execution to known brokers which will try to find appropriate agents to carry out the task. The efficient and effective task requesting and automatic match making with agents requires appropriate descriptions of requirements for tasks, agent's capabilities and the different types of task requester and provider. This can be done using a knowledge-based approach.

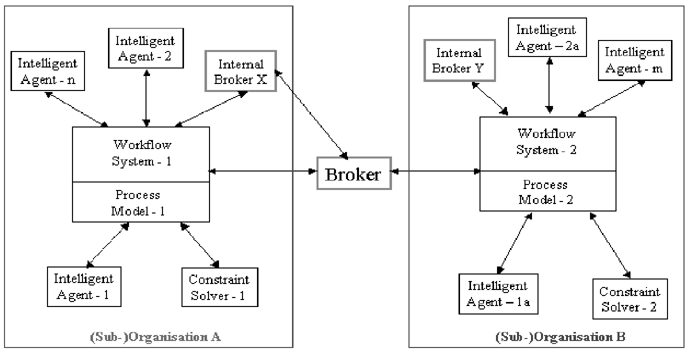

Figure 6 Workflow in the Context of a Distributed Environment

Our *Knowledge-based* capability matching that refers to the more sophisticated matching that takes into account knowledge about capabilities themselves and relationships between them.[Jarvis et al, 1999] The reason for using any such matching function in a workflow context is that it is impossible to predict the exact environment in which a task is executed. Similarly, specific agents may not be available at the time of execution (people take holidays or leave the organisation), or more suitable agents may have become available (people are hired and new software systems are developed). Similarly, activities may not be required in the specific context of a task's execution. Availability of agents not only has an impact on assigning activities to agents, but also on the decision of which method is chosen to achieve a given task. If a method for carrying out a task requires a particular capability but there are currently no agents available with that capability, then the task must be achieved using an alternative method.

Using a specification schema based on the hierarchical capabilities ontology in a matching function, the workflow support can not only determine which agents match the capability requirements of an activity exactly, but it can rank all agents available at the time of execution according to how closely they match the capability requirements. Exact matches of a capability specification are best, but agents that can apply the required capability in a wider area than required are nearly as suitable. Similarly, agents that have a more general capability are suitable, although more specialised agents would be preferred because they are likely to perform the activity more effectively.

Workflow Collaboration and Conceptual Mapping

Figure 7 depicts a conceptual overview of an implementation of two workflow systems that are based on FBPML. Their implementation is part of a joint initiative with Aberdeen University, UK, under the AKT project. The two systems reside in two different organisations, but they rely on each other’s collaboration to accomplish organizational goals. They operate within the context of a distributed environment as described previously in Figure 6.

The two workflow systems in this implementation, the Edinburgh Costing Site and the Aberdeen Technical Site, use the same workflow engine. Each system, however, has its own business process model that governs its behaviours (BPM-1 and BPM-2). Each business process model consists of two parts: the internal processes and communication processes. The internal processes deal with operations that do not require external capabilities, whereas communication processes (ComP-1 and ComP-2) request services from other agents and provide solutions and information to other agents.

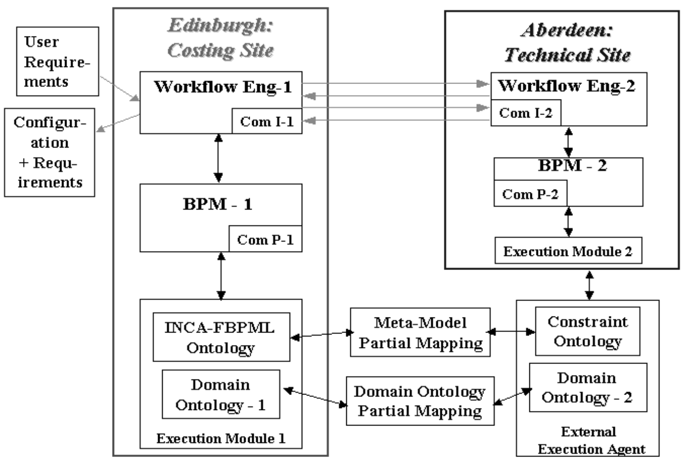

Figure 7: An Implementation of Workflow Communication

The two execution components are the Execution Module that is internal to the Edinburgh workflow system and Execution Agent that is external to the Aberdeen workflow system. The meta-Ontology for the Edinburgh and Aberdeen workflow system are the same, as they are the same workflow system. The domain ontologies for the two systems, however, are different as they are governed by the local module and agent capabilities. To enable collaboration, the meta-ontology of the two process execution entities have been partially mapped at design time. It is a partial mapping, because only those concepts that are significant for communication need to be matched. Such concepts are the primitives that are used by the corresponding methods to describe and organise domain information. In this case, the two sites use different frameworks to organise their processes: the Edinburgh Execution Module uses the modelling primitives of the I-N-C-A framework [Tate, 2002], whereas the Aberdeen Agent uses the semantics and structure of constraint sentences that are used by its Constraint Satisfaction Solver, KRAFT [Preece, 2000].

As it is a reasonable assumption that each execution entity may have specialised expertise as well as local knowledge, the Edinburgh execution module has a domain ontology that is different from that of the Aberdeen agent. The two different ontologies therefore require to be mapped before information may be shared correctly. The mapping covers all the domain information that is to be exchanged during the task execution and problem solving process. The Aberdeen agent, KRAFT, is a generic Constraint Satisfaction Solver whose domain data can be described using any Relational Data Model. The domain ontology at the Edinburgh site is FBPML-based. One of the advantages of FBPML is that its data model can be fully translated to an ER Data Model. Therefore the mapping of data between the two domain ontologies is straightforward. The constraint sentences in FBPML are mapped to KFAFT constraints based on the mapping of the two meta-ontologies.

Using the partial mappings for the meta-models and domain ontologies, information can be shared between different workflow systems and tasks may be carried out by internal and/or external problem solving capabilities when appropriate. Figure 4 shows the first level process decomposition for the Edinburgh Site which plays the role of "Sales and Marketing Agent". After obtaining new customer requirements (the event occurrence "Customer_request_for_pc_specification") and storing this information, the Edinburgh workflow system carries out two activities: "Fulfil Technical Requirements" and "Fulfil Cost Requirements". As Edinburgh is the marketing and sales agent, it does not have knowledge on machine assembly. However, it knows that the Aberdeen Technical Site is capable of solving such problems, it therefore may choose to dispatch the "Fulfil Technical requirements" activity to the Aberdeen site. Upon receiving the new task, the Aberdeen workflow system invokes an appropriate problem solver, the KRAFT system, which may provide several possible specifications.[2] The Aberdeen workflow will transfer those specifications back to Edinburgh. In parallel, the Edinburgh site may invoke the "Fulfil Cost Requirements" process to derive specifications that satisfy customer's cost constraints. If any suitable configuration has been found to satisfy both requirements, i.e. a conjunction of the above two results, the solution is given to the customer and the process is finished.

However, if there is no suitable specification, the Edinburgh site may choose to alter some user requirements (either heuristically or randomly) and repeat the above two parallel configuration processes until a solution has been found and can be given to the customer. As there is no constraint on the execution sequence between these two configuration processes (because an or-split has been used), the two processes may be executed in parallel or one after the other. Both processes, however, must be finished before the "Identify Configuration" process may be performed; this is because an and-join has been used.

This example demonstrates how a core WfMS may be replicated to perform different actions in different organisations, how it collaborates with each other on tasks based on conceptual mapping, and how external execution capabilities may be used to extend the capabilities of a WfMS. In this case, only one external

[2] Due to limited space, the Aberdeen process model is not displayed.

knowledge-based problem solver has been used, but the approach is generic and can be used to include other facilities.

Conclusion

The work described in this paper provides a valuable contribution to the workflow community for several reasons. By underpinning Enterprise Modelling techniques with a formal basis, it allows those techniques to be used for building models that are suitable for workflow support. Not only does this make process models suitable for enactment, but with the formal grounding it is also possible to use the models directly for workflow system design. In addition, the formal grounding can be used to detect errors and inconsistencies, and to support the user in dealing with such problems. The clear and unambiguous nature of the formal process models forms a reliable basis for formal automatic analysis, verification, validation, simulation and giving error-correction and exception-handling advice.

The layered approach proposed in this paper separates business and technical decisions while keeping track of design rationale for any technical decisions being made. This increases confidence in the model and provides good reference for any necessary future changes. The loose-coupling mechanism between the formal language and the actual WfMS, when applied correctly, allows rapid WfMS prototyping and development based on the newest technologies.

We believe that our approach can bring much more flexibility to WfMS, which leads to a new generation of WfMS, which will open up new application areas for which previously WfMS were too restrictive or did not provide enough support. All those abilities aim to make a WfMSs more flexible and to enable them to adapt rapidly to accommodate an organisation's fast-changing and evolving needs. This is a timely contribution because Web services are becoming more and more ubiquitous and businesses expect more and more from these services. Increasingly, the Web services are described in terms of process models and realised in WfMS [Ankolekar, 2002]. This has lead to pressure on the workflow community to produce WfMSs that provide versatile functionality to support diverse systems and devices, role-oriented operations, rapid adaptation for frequent changes of business practice, and error-free, high quality services in a dynamic and distributed environment. These are exactly the kind of requirements that our approach covers.

However, dynamic workflow behaviour can be quite complex and unpredictable, especially when parallel cooperative processes are being carried out by non-deterministic agents in a distributed environment. Despite the ability of formal approaches to help the understanding of dynamics and to provide adaptability and stability, there is lack of evidences that formal approaches have been widely used. It is therefore interesting and urgent to investigate how formal languages can help to specify behaviour and improve performance of a WfMS in all stages of its design-build-test-deployment life cycle.

Acknowledgements

This paper describes work carried out during several of AIAI's projects, including the Task-Based Process Management (TBPM) project, the Advanced Knowledge Technologies (AKT) project, and several commercial knowledge transfer projects. The authors would like to thank the members of the relevant project teams for their contributions to this work.

The TBPM project has been carried out jointly between the Artificial Intelligence Applications Institute at the University of Edinburgh and Loughborough University Chemical Engineering Department, under EPSRC research grants numbers GR/L42179 and GR/L42346 on the "Systems Engineering for Business Process Change" program.

The AK) (IRC) project 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. Alonso G, Agrawal D, El Abbadi A, and Mohan C, “Functionality and Limitations of Current Workflow Management Systems,” IEEE Expert, 12(5), 1997.
2. Ankolenkar A., Burstein M., Hobbs J., Lassila O., Martin D., McDermott D., McIlraith S., Narayanan S., Paolucci M., Payne T. and Sycara K., The DAML Services Coalition (alphabetically), "DAML-S: Web Service Description for the Semantic Web", The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.
3. AOEM: Air Operation Enterprise Modelling Project, Joint Force Air Component Commander (JFACC), Defense Advanced Research Projects Agency Program, http://www.darpa.mil, 2001.
4. Bundy A, “The Computer Modelling of Mathematical Reasoning”, Academic Press Inc Ltd. 1983.
5. Chen-Burger Y., “Formal Support For An Informal Business Modelling Method”. PhD Thesis, Informatics, The University of Edinburgh, 2001a.
6. Chen-Burger Y., “Sharing and Checking Organisation Knowledge”. Chapter of book: Knowledge Management and Organizational Memories. Editors: Rose Dieng-Kuntz, Nada Matta. Publisher: Kluwer Academic Publishers, Boston, ISBN 0-7923-7659-5, July 2002c.
7. Chen-Burger, Y., “Informal Semantics for the FBPML Data Language”, technical manual ED-INF-RR-154, Informatics, The University of Edinburgh, 2002b.
8. Chen-Burger Y., Tate A., Robertson D., “Enterprise Modelling: A Declarative Approach for FBPML”, European Conference of Artificial Intelligence, proceedings of Knowledge Management and Organisational Memories Workshop, 2002a.
9. Delphi Group, “BPM 2002: Market Milestone Report”, www.delphigroup.com, Feb 2002.
10. Dobson J. and Blyth A., Chudge J. and Strens M., “The ORDIT Approach to Organisational Requirements”, Requirements Engineering: Social and Technical Issues, London, ed. Jirotka and J.A.Goguen, Academic Press, 1994.
11. Fox M. and Gruninger M., “Enterprise Modelling”, AI Magazine, AAAI press, Fall 1998, pp.109-121.
12. Frank U., “Multi-Perspective Enterprise Models as a Conceptual Foundation for Knowledge Management”, Proceedings of Hawaii International Conference on System Sciences, Honolulu, 2000.
13. IBM, “Business System Development Method, Business Mapping, Part1: Entities; and Part 2: Processes”, 2nd ed, IBM England, May 1992.
14. Jarvis, P., Stader, J., Macintosh, A., Moore, J., Chung P., “What Right Do You Have To Do That?: Infusing adaptive workflow technology with knowledge about the organizational and authority context of a task”. First International Conference on Enterprise Information Systems (ICEIS-99), Setubal, Portugal,1999.

15. Junginger S., Kuhn H., Heidenfeld M., Karagiannis D., "Building Complex Workflow Applications: How to Overcome the Limitations for the Waterfall Model", Workflow Handbook 2001, Ed. Layna Fischer, Published in association with the Workflow Management Coalition (WfMC), 2000, pp.191-224.
16. Lee J. and Gruninger M. and Jin Y. and Malone T. and Tate A. and Yost G. and other members of the PIF working group, "The PIF Interchange Format and Framework", The Knowledge Engineering Review, Vol. 13, March 1998.
17. Mayer R, Cullinane T, deWitte P, Knappenberger W, Parakath B, & Wells S, "IICE IDEF3 process description capture method report (al/tr-1992-0057)". Technical Report, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, 1992. See also http://www.idef.com/
18. Moore J, Inder R, Chung P, Macintosh A, and Stader J, "Who Does What? Matching Agents to Tasks in Adaptive Workflow," proceedings of International Conference on Enterprise Information Systems, Stafford, UK, July 2000.
19. NIST, "Integration Definition for Function Modelling (IDEF0)", Federal Information Processing Standards Publication 183, National Institute of Standards and Technology (NIST), Dec 1993.
20. Ould M, "Business Processes: Modelling and Analysis for Re-engineering and Improvement", John Wiley and Sons, 1995.
21. Preece A., Hui K., Gray A., Marti P., Bench-Capon T., Cui Z., Jones D., International Journal on Intelligent Cooperative Information Systems (IJCIS), 2000.
22. Robertson D, "An Introduction to Logic", Lecture Note, Informatics, The University of Edinburgh. September 1992.
23. Robertson D. and Augusti J., "Software Blueprints: Lightweight Uses of Logic in Conceptual Modelling", Addison Wesley, 1999.
24. Schlenoff C. and Gruninger M. and Tissot F. and Valois J. and Lubell J. and Lee J., "The Process Specification Language (PSL): Overview and Version 1.0 Specification", ISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD (2000), http://www.nist.gov/psl/, 2000.
25. Sheth A., "From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration". Workshop on Workflows in Scientific and Engineering Applications, France, September 1997.
26. Stader J., Moore J., Chung P., McBriar I., Ravinranathan M., and Macintosh A., "Applying Intelligent Workflow Management in the Chemicals Industries", Workflow Handbook 2001, L. Fisher (ed), Published in association with the Workflow Management Coalition (WfMC), 2000, pp.161-181.
27. Stader J, "Results of the Enterprise Project", Proceedings of the 16th International Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK, 1996.
28. Tate A, "I-X: Technology for Intelligent Systems", www.i-x.info, AIAI, The University of Edinburgh, 2002.
29. Uschold M, and Gruninger M, "Ontologies: Principles, Methods and Applications", The Knowledge Engineering Review, 11(2), 1996, pp.93–136.
30. Uschold, M., King, M., Moralee, S., and Zorgios, Y., 1998, "The Enterprise Ontology". The Knowledge Engineering Review, Vol. 13., p 31-89. Also available at http://www.aiai.ed.ac.uk/~entprise/enterprise/ontology.html.
31. Waern A. and Hook K. and Gustavsson R. and Holm P., "The Common-KADS Communication Model", KADS-II/M3/SICS/TR/006/V2.0, Swedish Institute of Computer Science, Stockholm, Sweden, Dec 1993.