8-16-1996

A Business Process Model Based On A Comprehensive Content Specification

Akhilesh Bajaj  
Department of MIS, University of Arizona

Sudha Ram  
Department of MIS, University of Arizona

Follow this and additional works at: http://aisel.aisnet.org/amcis1996

Recommended Citation
http://aisel.aisnet.org/amcis1996/1

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 1996 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A Business Process Model Based On
A Comprehensive Content Specification

Akhilesh Bajaj
Dept. of M.I.S., University of Arizona

Sudha Ram
Dept. of M.I.S., University of Arizona

email: abajaj@misvms.bpa.arizona.edu

mailing address: Akhilesh Bajaj, Dept. of M.I.S., Karl Eller Graduate School of Management
University of Arizona, Tucson, AZ 85721, U.S.A.

ph: (520) 621 - 2748 fax: (520) 621 - 2433

1. Introduction

Process models have traditionally evolved from modeling software processes. Recently, business process modeling has received increased attention in the areas of Business Process Reengineering (BPR) and Workflow Management (WM). Most modeling in these areas has entailed adapting existing process models. Doing this causes 2 problems. First, software process models usually model only one aspect of a process. E.g., Data Flow Diagrams (DFDs) model the flow of information, but leave out timing and spatial issues and do not model non informational entities. Most software modeling paradigms use multiple models to model a process. E.g., [3] advocates the use of class diagrams, object diagrams, state transition diagrams and physical diagrams to model the different aspects of a system. Process models should be easy to use [3,8]. However, having to integrate many models to get a complete view lowers ease of use and user acceptance. Second, business processes present some new concepts, [1,5,6,7,8], not posed by software processes (E.g., physical objects, roles, spatial and temporal aspects, etc.). It is clear that a business process model must support these new concepts.

This work contributes towards solving these problems in 2 ways. First, a comprehensive content specification for a business process model is developed. This specification is used as a framework to analyze and compare existing process models (E.g., DFDs and IDEF0). We show that this framework offers detailed insights into the structure of process models. Next, we use this content specification to design a business process model (BPM) that can represent many aspects of a business process. The structure and axioms of BPM are presented next. Finally, we demonstrate its usage with a business process. Due to space constraints, we present briefly, in section 2, the informal definition of BPM based on the content specification, which has been presented elsewhere [2]. Its usage as a framework, and the formal structure of BPM are presented in a series of technical reports, which can be obtained from the authors. We conclude with plans for future research in section 3.

2. Informal Specification of BPM

BPM is designed to conform with our content specification [2] so that it captures a structural as well as a dynamic view of the business process. We present below an informal definition of BPM. The graphical constructs that are used to model BPM are not shown for lack of space, but can be obtained by contacting the authors.

**Entity Descriptors:**

*Attributes:* These describe an entity semantically. They are a functional mapping from an *entity_type* set to a domain of possible atomic values. A role and *busy_degree* are special attributes, which all entities have. They are defined below.

*Busy_degree:* This describes the extent to which an entity is busy. A value of 1.0 means the entity is completely busy, and unavailable for more work. *Busy_degree* is useful in capturing the semantics of resources as well as agents (who perform an activity). *Busy_degree* is a functional mapping of an *entity_type* set to the domain: \{0.0, 0.1, 0.2, ..., 1.0\}.
Role: This is a description of the part the entity plays in the enterprise. It's only purpose is to convey semantic information to the user. In some implementation of our model, it may be possible to restrict the value to only a certain set of strings, such as {agent, resource, customer, manager}. A set of rules can then be associated with each of these values. E.g. only an agent can start an activity, etc.. However, we consider this to be restrictive. Since our model does not explicitly support rules (although an implementation might), we simply use role to convey semantic information to the user, when developing an instance of our model. Role is a functional mapping of an entity_type to a set of alphanumeric strings, each of which are user defined.

Time_stamps: This fixes the existence of an instance of an entity_type to a certain time value. It is a functional mapping from an entity_type to a time_stamp.

Space_stamp: This fixes the existence of an instance of an entity_type to a certain location in space. It is a functional mapping from an entity_type set to a set where each element is a 3-tuple (for the 3 spatial dimensions).

Time_distance: The value of the arithmetic difference between 2 valid time stamps.

Space_distance: The value of the Euclidean distance between 2 space stamps.

Note that the values of these entity_descriptors will be reported from different sources. E.g. The value of attributes will probably come via a user. The value of time_stamp may come from within the system itself. The value of space_stamp may come from a sensor, or from a user. At this level, we are transparent to these issues.

Entities:

Entity_types are logically distinguishable objects that can be uniquely distinguished based on the entity_descriptors. Entity_instances of the same entity_type are described by the same descriptors. When developing an instance of BPM, entity_types will be user defined.

State:

A state_type is a 12-tuple. One element of this is a set of 3-tuples, such that [entity_type, min_no_instances, max_no_instances] make up each 3-tuple. Each 3-tuple describes which entity_type takes part in the state and how many instances can take part in the state. A state_type also has associated with it:

- a time_stamp, which fixes the existence of an instance of the state_type to a certain time value;

- a max_interval, which determines how long the maximum allowable interval between the time_stamps of entity instances in a state_instance of the state_type can be and;

- a set of eight space_stamps, which define a cuboid, each of whose rectangles is parallel to one of the reference axes. This cuboid represents the spatial span of any instance of the state_type.

In an implementation of BPM, some elements of this 12-tuple may be defined at the onset and then made unchangeable, e.g. the eight space_stamps, the time_span, the max_interval, etc. However, that is a special case, and is not enforced in our specification.

A state_instance is a collection of entities satisfying the state_type, such that the time_stamp of each entity_instance in the set is within + / - max_interval time_units of the time_stamp of the state_instance.

Activities:

An activity_type is a functional transformation on a state_type. An activity_instance takes a state_instance to another state_instance (both of the same state_type). This conceptualization of an activity accounts for activities that move entities through space (functionally transform the values of their space_stamps), through time (functionally transform the values of their time_stamps), descriptively alter entities (functionally transform the values of their attributes) and change the roles and busy_degree of entities.

Decomposition:

Entities:

Entity_types can be decomposed based on the notions of inheritance (sub / super class). The notion of relationships (and aggregation, which is higher order relationships) is modeled by the notion of entities participating in a state_type.
State:

State types can be decomposed along 3 dimensions:

Descriptive decomposition: Here, the decomposed states better describe the original.

Spatial decomposition: Here the decomposed states are spatial decompositions of the original.

Temporal decomposition: Here the decomposed states give the same description as the original, but along a shorter temporal segment (either valid or transaction time).

Activities:

Activity types can be decomposed along 5 dimensions:

The first 3 dimensions correspond exactly to decomposed states. Thus, if an activity_type \( A \) acts on state_type \( S \), then subactivity_types of \( A \) will act on the substate_types of \( S \).

Activity types can also be decomposed along one spatial transport dimension. Thus, if activity_type \( A \) transports a set of entity_types through space distance \( L \), then its subactivity_types will transport the same set of entity_types through distances that are contained in \( L \).

Finally, activity_types can be decomposed along a temporal transport dimension. If activity_type \( A \) transports a state_type \( S \) through time \( T \), its sub_activity_types transport \( S \) through time intervals contained in \( T \) (can be valid or transaction time).

Primitives (end of decomposition):

Entities: These are arbitrary.

States: One entity_type at least. However, it is possible to specify arbitrary states as primitive.

Activities: First 3 decompositions correspond to states. The spatial decomposition stops when a space distance of unit size is reached. The final decomposition stops when a unit time_distance is reached. Again, it is possible to specify arbitrary activities as primitive.

Sequencing and Control Flow:

Sequencing and control flow can only occur between activities. We assume that predicates based on values of entity_descriptors can be evaluated. Constructs to represent sequencing, either \( / \) or branching (predicate based), while \( / \) repeat looping (predicate based), concurrent execution are supported by BPM.

Atomicity is modeled by the atomic relation that acts on a set of activity_types.

Constraints:

These can all be modeled as range and value restrictions of entity_descriptors.

availability of resources: modeled by the condition that busy_degree <= 1.0 for some entity_types in the state.

real-time constraints for activity_types and state_types (e.g., a state_instance can exist only for a certain time period): modeled by a predicate on time_stamp values of state_types.

spatial constraints: modeled by a predicate on the space_distance between entities.

temporal constraints: modeled by a predicate on the time_distance between states.

state_type constraints: modeled by a restriction on values of attributes and roles.

Relations in BPM:
Acts_on: This is a relation between activity_types and state_types. It represents the concept that activity_type A functionally transforms state_type S.

Belongs_to: This is a relation between entity_types and state_types. An entity_type E can participate in many state_types, and an instance E1 of E can belong to more than one state_instance.

Axioms:

If an activity_type A acts_on a state_type S, it acts on all the entity_types contained in S.

Primitive entity_types, state_types and activity_types cannot be decomposed further.

Decomposition of entities, states and activities can be along several orthogonal dimensions. E.g., a state S can be decomposed into S1 and S2 along the descriptive dimension, and also into S3 and S4 along the temporal dimension.

If a set of activity_types is atomic, then either all occur, or none occurs.

A primitive activity_type is necessarily atomic.

Entity Decomposition:

If a superclass participates in a state_type, so do all its subclasses.

If an activity_type A acts on a superclass E, it acts on all subclasses of E.

If an activity_type A acts on an aggregate E, it acts on all components of E.

Note that an activity_type A acts on an entity_type E, iff A acts_on state_type S and E belongs_to S.

State Decomposition:

If an activity_type A acts on a state_type S, it acts on all of the substate_types of S.

Activity Decomposition:

If an activity_type A precedes activity_type B, then all sub_activity_types of A precede all sub_activity_types of B.

If a subactivity_type A1 of activity_type A acts on a state_type S, then A acts on S as well.

If a subactivity_type A1 of activity_type A acts on an entity_type E, then A acts on E as well.

If activity_type A acts on state_type S, an instance of A cannot move any entity that belongs to an instance of S, beyond the cuboidal volume of S.

If activity_type A acts on state_type S, then an instance of A cannot move an instance of S a time distance beyond the time_span of S.

Each level of BPM has two diagrams. The first diagram (static diagram) represents entity_types, the state_types they participate in (with cardinalities) and the activity_types that act on these state_types. The other diagram (dynamic diagram) represents the sequencing and control flow of the activity_types shown in the first diagram. In the dynamic diagram, predicates can be specified, based on entity_descriptors of entity_types in the static diagram. Note that it is possible to show both diagrams in one (since the dynamic diagram only consists of a few additional links between activities which are shown in the static diagram.) However, the static and dynamic information is split into two diagrams for simplicity.

The next level of BPM is built as follows: every entity_type, state_type and activity_type at a lower level are either decompositions of or identical to those at the immediate parent level. Each level has only one static and one dynamic diagram (showing greater detail as we go to lower levels).

Decomposition stops when all 3 types of components of a level are primitives. This is the primitive level.
One model instance can have only one primitive level. At this level, all activities, states and entities are primitives (as defined earlier). The dynamic diagram at the primitive level gives the sequential ordering and control flow of activities at the lowest possible level.

Different workflows can be shown simultaneously in the BPM. Every enterprise has an overall state and an overall activity. This activity can be decomposed into several activities (each of which is a workflow). Each workflow can then be modeled separately, with the other workflows either being decomposed or being arbitrarily specified as primitive (if only one workflow needs to be modeled in detail).

**Advanced Analyses:**

*Reachability:* The process model can be viewed as a directed graph, where the usual reachability algorithms apply.

*Deadlock:* The process model can be mapped to a resource allocation graph, and deadlock can be checked, using common algorithms such as the banker's algorithm.

*Optimality:* The weight of each activity can represent the amount of the resource (the resource we need to optimize) consumed. We can then use a shortest path algorithm to reach from one state to the last.

### 3. Conclusion

BPM is a conceptual model that serves to represent the important aspects of a business process in one model. It is part of a larger project that aims at exploring database support for workflow management.

**References available upon request from A. Bajaj.**