A CONTENT SPECIFICATION FOR BUSINESS PROCESS MODELS

Akhilesh Bajaj & Sudha Ram

Dept. of M.I.S., University of Arizona
Address: email: abajaj@misvms.bpa.arizona.edu
mailing address: Akhilesh Bajaj Dept. of M.I.S.
Karl Eller Graduate School of Management, Mclelland Hall
University of Arizona, Tucson, AZ 85721, U.S.A.
phone: (520) 621 - 2748
fax: (520) 621 - 2433

ABSTRACT

Business process modeling is an essential prerequisite to business process reengineering (BPR), and workflow management (WFM). Process models have been traditionally used to model **software** processes, and many **business** process models are adaptations of these process models. Using these process models to represent business processes results in two problems. First, since these process models usually represent different perspectives of systems (or in this case, businesses) the user needs to **integrate** multiple existing models to completely represent the business processes. This reduces the ease of use, and leads to a lower acceptance by users. Second, business processes contain concepts not found in software processes (*e.g.*, physical objects, roles, etc.). Traditional process models cannot represent these new concepts, and hence traditional process models model business processes inadequately. These two problems can be easily solved if a comprehensive business process model exists, that models all perspectives of a business process, and that allows representation of these new concepts. As a first step towards this goal, we propose a content specification that would need to be satisfied by such a business process model.

The primary contribution of this work is a comprehensive content specification for a business process model that will solve the two problems listed above. This content specification also serves as a framework to analyze process models in detail, and to compare them based on their content (*i.e.*, **what** concepts they model) and the degree to which they model each aspect of a business process (*i.e.*, **how much** of a business process they model).

INTRODUCTION

Process models have traditionally been used to model software processes [Curtis (1992)]. Recently, attention has turned to modeling and reengineering business processes, especially in the emerging area of **workflow management** [Attie (1993), Georgakopoulos (1994), Georgakopoulos (1995), Joosten (1994), Joosten (1995)]. Most work in this area has entailed extending existing process models, which were used earlier primarily for software processes.

While a large number of process models (over a hundred) exist [Olle (1986)], adapting any of these to model business processes poses two problems. The first is that **different models** model different facets of a system [Curtis (1992)]. Thus, [Booch (1994)] advocate the use of class diagrams, object diagrams, state transition diagrams, physical diagrams to model different aspects of a system. [Barker, (1992)] also advocate different models (dependency charts, functional hierarchy charts, data flow diagrams (DFDs), etc.) for modeling different aspects of a system. [Martin (1987)] recommend decomposition diagrams, dependency diagrams, entity-relationship (E-R) diagrams, state transition diagrams, and DFDs to model different aspects of business processes. While many of these models sufficed for software processes, none of these models are considered sufficient by themselves to model a business process comprehensively. Many commercial tools based on one or more of these models exist for modeling business processes ([Georgakopoulos (1995)] provide an extensive listing of these). However, many of these tools either also use different models to model different aspects of business processes (*E.g.*, [Booch (1994)]), or alternately, arbitrarily extend these models in their software, without any attempt at formality. Many of these models were formulated initially in the late 1970's and early 1980's, and researchers have recently tried to add formal extensions to these models [Kung (1991), Tao (1991), Ward (1986)]. However, once again, none of these extensions result in a model that comprehensively captures all aspects of a business process. One of the requirements of a model is that it should be easy to use [Curtis (1992), Kramer (1991)]. The fact that a model can only model some perspectives of a business process leads to a situation where many (often complicated) instances of **different** models have to be integrated to form a complete view of business processes. This has resulted in low ease of use for users, and low acceptance of models by industry. The second problem is that business processes require some new concepts that were not needed for modeling software processes. *E.g.*, physical objects, humans, location, time, etc. need to be modeled in business processes [Georgakopoulos (1995), Joosten (1995)].

In order to solve these 2 problems, a single business process model is needed, that can comprehensively model all aspects of a business process. In this work we take a first step towards this goal, by proposing a **contents specification** for business process models. Each dimension of this framework represents a need that should be fulfilled, in order to model a business process. This specification makes two contributions. First, it serves as a

framework to analyze and compare existing models, based on their content and which perspectives of a business process they model. Second, it is a useful guideline for developing a comprehensive business process model that will solve the two problems mentioned above. The rest of this paper is organized as follows. In the next section, we examine two taxonomies that have been used to study existing process models. We also examine some requirements for process models that have been suggested earlier. After that, we specify the content specification for a comprehensive business process model. We then demonstrate the use of this specification as a framework for detailed analysis and comparison of two existing process models. Finally, we discuss how this content specification can be used for further research.

EXISTING FRAMEWORKS AND REQUIREMENTS FOR PROCESS MODELS

Existing frameworks to study process modeling

Many taxonomies have been proposed to classify existing process models [15-19 in Tolvanen (1994)]. We consider two recently proposed frameworks here. The first classifies models based on the size of the system they try to model, as well as the contents and functionality of the model. The second is based purely on the content of the models.

[Tolvanen (1994)] propose a two-dimensional framework for analyzing business process models. The first dimension is divided along the size of the information system (IS) being modeled, and has 5 values: methods for modeling: specific internal ISs, internally integrated ISs, organizational interface ISs, interorganizational relationships and ISs for business networks (many organizations).

The second dimension is based on what information the model captures. It consists of four sub-dimensions, each of which represents one functionality dimension. The sub-dimensions are:

Scope of analysis: What aspects of the real-world are modeled. *E.g.,* DFDs have processes and dataflows.

System Structuring: How these elements are put into an orderly whole. *E.g.,* the principles of decomposition that govern a model.

Heuristics for candidate designs: The rationale behind generating designs automatically. *E.g.,* A shortest path algorithm to generate the shortest path in a graph.

Standards of Performance Evaluation: How to assess and select the candidate designs or solutions.

[Falkenberg (1991)] propose viewing each model as containing *objects*, with a set *of intuitive axioms* that govern how the designer can use these objects, as well as a set of rules, derivable from the axioms and the rules of the language used to create the model. Thus, DFDs have processes and dataflows as objects, with a set of axioms. *E.g.,* an axiom is: each process must have at least one dataflow connected to it. If the DFD model is specified using predicate logic, then rules can always be derived using the rules of predicate logic, and the axioms of the model. In a sense, this taxonomy corresponds to the "scope of analysis" and "system structuring" sub-dimensions proposed by [Tolvanen (1994)].

These two frameworks reflect the fact that most taxonomies of process models are based on: the content and functionality of the model, the aspects of a process that are captured, and on the size and complexity of the system that the model can represent. However, the dimensions used in most existing taxonomies are broad, and do not allow an extensive analysis of the models. As part of this work, we propose a detailed *content specification* for business process models. The specification represents different aspects of a business process that must be modeled in order to represent the process. Any process model will meet these requirements to varying degrees. This specification improves on existing frameworks for analyzing process models because it allows a much more detailed analysis of process models, based on their content and what aspect of a business process are modeled. It also serves as a good design prescription when developing a business process model.

Previously identified requirements for process models

Several requirements for process models have been identified in past research. [Opdahl (1994)] describe the concepts of **transportation** (through space), **transformation** (of state) and **storage** (transportation through time) as basic aspects of processes. In addition, they distinguish between simple input/output relationships between processes, and the actual flow of data and matter between processes.

[Rubin (1992)] discuss the importance of time ordering in dynamic modeling. Object states, events, and the sequence in which these events occur are all stated to be important. Sequencing is classified as **concurrent, repetitive, selective** or **optional**. They construct the lifecycle of objects as a series of states, with events causing changes to these states. They also specify other desirable attributes of a process model: it should be analyzable for *syntactic correctness, consistency, completeness*. In addition, it should support more advanced analyses of concepts such as *reachability, deadlocks, race conditions and behavioral ambiguities.*

They describe four commonly represented perspectives that a process model should represent: *functional* (what activities are being performed, what is being produced, what is being consumed, etc.), *behavioral* (when activities are performed, as well as how (feedback loops, iteration, entry criteria, exit criteria, etc.)), *organizational* (where and by whom in the organization are activities performed, where artifacts are stored, and the physical communication mechanisms by which they are performed), and finally, *informational* (which informational entities are produced, manipulated or consumed by a process).

[Curtis (1992)] describe process modeling to include the modeling of phenomena that are enacted by humans as well. They enumerate five basic uses of process models: to *facilitate human understanding*, to *automate process descriptions*, *to set a standard for actual process execution* in the organization, to *provide a framework for analyzing processes*, and to *automate actual processes*. Other information that people need from process models is "what is going to be done, who is going to do it, when and where will it be done, how and why will it be done, and who is dependent on its being done." [Curtis (1992)]

[Kramer (1991)] list *adequacy, readability* and *ease of use*, *hierarchical decomposability* and *amenability to formal analysis and reasoning* as typical requirements for process models.

In the workflow literature, [Georgakopoulos (1995)] define a **workflow** as a collection of tasks organized to accomplish some business process. They state that workflow models must describe *processes, subprocesses, dependencies* between these, and required *roles* that can fulfill these tasks. They also list capturing process *objectives* (such as customer satisfaction) in the workflow model.

[Attie (1993)] specify certain dependencies that can exist between various tasks, and propose a finite automata model to enforce these. Some of the dependencies they list are: *existence dependencies* (if task *A* occurs, then *B* must also occur), *conditional existence dependency* (if *A* and *B* occur, then *C* must occur), *temporal dependencies* (if *A* and *B* both occur, then *A* must precede *B*), etc. [Joosten (1994)] describe how *triggers* can be a part of a workflow model.

In addition, there has been extensive work on modeling dependencies between workflow transactions operating on a database (*e.g.*, [Georgakopoulos (1994)]). We do not review this literature here, since it is specific to transaction processing, and hence at a lower level than the process models we are trying to classify here.

The requirements listed above justify the dimensions in our specification, which is described next.

A CONTENT SPECIFICATION FOR BUSINESS PROCESS MODELS

We divide our specification into broad **dimensions** such as **system, state, space, time,** etc. These dimensions are chosen based on previous requirements (described in the previous section). Within each dimension are listed the actual **sub-dimensions** pertinent to that dimension.

System

- The model must define what is an **entity** in the system, and how it is recognized. *E.g.*, an entity could be defined as a logical object that is different from other objects.
- The model must be able to clearly distinguish between entities in the system, and those outside the system. *E.g.*, DFDs use external entities to model entities outside the system.
- Entities must be able to take on one of many different roles. The set of possible roles must be predefined in the system. Examples of roles are: *agent, resource, responsible agent, customer, performer*, etc.
- Entities must be distinguishable into **artifacts** (non-human physical objects), **information entities** (non-human, non-physical objects), or **humans**.

State

- The model must define what it means by *state_type*. *E.g.*, the "attributes" of a set of entities may make up a *state_type*, and values of attributes of all these entities might constitute a *state_instance*.

  The definitions should be verifiable, and unambiguous (*i.e.*, the analyst should be able to tell what *state_type* the system is in, and what *state_instance* it is in).

Space

- The model should be able to capture the **spatial location** of *entities* in the system.
- It should support a measure of **spatial distance**, and be able to model **spatial constraints** based on this measure. *E.g.*, it should be able to model the fact that a milling machine can never be more than 50 meters from a lathe.

Time

- The model should be able to capture the **temporal location** of *state_instances* in the system. Thus, a *state_instance* will occur at a particular time, which will be its location on the temporal dimension.
- It should support a measure of **temporal distance**, and be able to model **temporal constraints** based on this measure. *E.g., state_instance S1* must not lag behind *state_instance S2* by more than 6 hours.

Transformation / Transportation / Storage

- The model should support the notion of three *activity_types*: an *activity-type* that **transforms** the value of a *state_type*, an *activity_type* that **transports** entities through **space** and an *activity_type* that **transports** *state_types* through **time (storage)**. This is based on the idea proposed by [Opdahl (1994)], where *processes, flows* and *stores* respectively support the 3 aspects of transformation, transportation and storage. *E.g., Activity_type A* transforms *state_type S.* An instance *A1* of *A* may then transform instance *S1* of *S* to instance *S2* of *S.*

Sequencing and Control Flow

- The model should support sequencing of a set of *activity_types.*
- It should support the atomic execution of a set of *activitity_types. E.g.,* If an instance of *activity_type A* occurs, then an instance of *activity_type B* **must** also occur.
- It should support concurrent execution of a set of *activity_types.*
- It should support either / or execution between two or more sets of *activity_types*, based on a predicate (similar to the "switch" statement in programming languages).
- It should support while and repeat-until flow on a set of *activity_types*, also based on a predicate.

Note that the above specifications subsume the concept of triggers (execution of an activity based on a predicate).

Decomposition

• *Entities:*

The decomposition of entities must be supported. *E.g.,* a document must be decomposable into its sections, which are themselves entities. The rules for this decomposition must be clearly stated. *E.g.,* In the object-oriented (OO) paradigm, super-classes may be decomposed into sub-classes, following a strict inheritance.

• *States:*

The model must support the decomposition of *state_types.*

Descriptive substates: If an overall *state_type* is used to describe a part of the system, it should be possible to decompose this overall *state_type* along descriptive *substate_types. E.g.,* An automobile assembly line can have an overall *state_type*, say, "state_of_assembly_line" and this overall *state_type* may be further described by the descriptive *substate_types*: "cars_output_rate", "defects_output_rate" and "worker_morale". Thus, *state_types* are decomposed in this dimension on a purely descriptive basis.

Spatial substates: The model should support the decomposition of *state_types* along the space dimension. *E.g.,* it should be able to represent the *state_type* of the whole office, as well as the *substate_type* of different cubicles spatially located in the office. *State_types* are decomposed in this dimension on a purely spatial basis.

Temporal substates: The model should support the decomposition of *state_types* along the time dimension. *E.g.,* it should be able to represent the *state_type* of the assembly line across a week (maybe by using an **average** measure for the week), and this *state_type* can be decomposed into *substate_types* that represent the state of the assembly line for a day. These *substate_types* can be further decomposed into *substate_types* that represent the state of the assembly line for an hour, etc. *State_types* are decomposed in this dimension on a purely temporal basis. Note that this is different from transporting a *state_type* across time, as described earlier. In the latter case, no decomposition of *state_types* takes place.

• *Activity_types:*

The model must support the decomposition of *activity_types.*

--Decomposition of *activity_types* that transform *state_types*:
Subactivity types acting on descriptive *substate types:* This would be a decomposition of *activity_types* on a descriptive dimension. *E.g.*, An overall *activity_type* might transform the *state_type* "state_of_assembly_line" and its descriptive *subactivity_types* will transform the descriptive *substate_types* "cars_output_rate", "defects_output_rate" and "worker_morale".

Subactivity types acting on spatial *substate types:* This is a decomposition of *activity_types* on a spatial dimension. Thus, if an *activity_type A* transformed a *state_type S*, the spatial *subactivity_types* of *A* would transform the spatial *substate_types* of *S*.

Subactivity types acting on temporal *substate types:* This is a decomposition of *activity_types* on a temporal dimension. Thus, if an *activity_type A* transformed a *state_type S*, its temporal *subactivity_types* would transform the temporal *substate_types* of *S*.

Note that the decomposition of *activity_types* that transform *state_types* corresponds to the decomposition of *state_types* described earlier. In addition to *activity_types* that transform *state_types*, we have two more *activity_types*, whose decomposition is described below.

--Decomposition of *activity_types* that transport entities through space:
If an *activity_type A* transports a set of entities $\{E_1...E_n\}$ through space distance L, then its *subactivity_types* may transport a subset of these entities through L.

Another type of decomposition would be *subactivity_types* of *A* that transport the entire entity set $\{E_1...E_n\}$ through distances L_1 ... L_m such that $L_1 + L_2 + ... + L_m = L$.

--Decomposition of *activity_types* that transport *state_types* through time:
If an *activity_type A* transports a *state_type S* through a time distance T, then its *subactivity_types* transport *S* through time distances $T_1...T_n$ such that $T_1 + T_2 ... + T_n = T$. Note that *A* can also be decomposed into *subactivity_types* that act on *substate_types* of *S*, but this has already been described earlier.

- The model should have atomic constructs that signify the end of a decomposition. *E.g.*, primitive *activity_types* might signify the end of *activity_types* decomposition for a system.

Constraints and Axioms
The model should support the following constraints:

- availability of resources;
- real-time constraints for *activity_types* and *state_types* (*e.g.*, a *state_instance* can exist only for a certain time period);
- spatial constraints, as described earlier;
- temporal constraints, as defined earlier; and
- *state_type* constraints. *E.g.*, The *state_type*: "working_state" for a researcher can never have a "vacation" *state_instance*.
- The model must explicitly use axioms that make any instance of the model unambiguous, and that show if the instance of the model is correct.
- The model must be specified in a language with well defined rules, that allow the derivation of new rules from axioms. *E.g.*, the model may be specified using a language based on predicate logic.

Advanced Analyses
The model should support analyses that show:

- if a *state_type* is reachable;
- whether deadlock is possible;
- whether or not a particular model specification is optimal, based on pre-defined criteria (*e.g.*, does the model
  specification consume the minimum resources to achieve the final state?);
- whether a process will ever go into an infinite loop; and
- whether a certain sequence of activities will ever cause a race condition.

The dimensions and sub-dimensions for the content specification are summarized in figure 1.

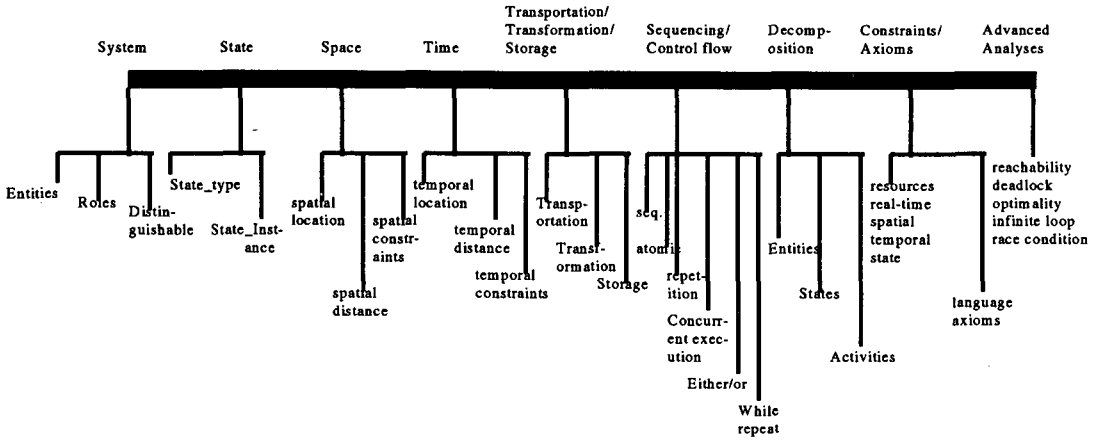

Fig. 1. Dimensions and subdimensions of the content specification for business process models

USING THE CONTENT SPECIFICATION TO CLASSIFY EXISTING MODELS

We now show how our content specification can be used to analyze and compare existing process models. DFDs are considered intuitively appealing and are popular [Opdahl (1994)]. There are also a lot of commercial models available based on the IDEF0 model [Laamanen (1994)]. We examine both these process models. The intention is not to criticize either of these models, but to demonstrate the usage of our specification as a framework for analysis and comparison.

Data Flow Diagrams

In their basic form, DFDs [DeMarco (1978), Gane (1982)] consist of **activators** (processes, external entities and data-stores) and **dataflows**. A short, and by no means complete list of axioms that are used to ensure consistency in the DFD model are shown in table 1.

- Each dataflow must have one source and one destination.
- DFDs may have cycles but no point cycles.
- DFDs cannot have isolated activators, which have no flows connected.
- Data stores and external entities have to be mutually distinct (as do processes).
- Each dataflow must have a process at at least one end of it
- Any activator connected with only one flow has to be an external entity
- Processes can be hierarchically decomposed, but it has to be a strict tree hierarchy (i.e. partial order + only one parent)
- If a dataflow exists at different levels, it must follow the hierarchy, i.e. the dataflow at a lower level can occur between two activators iff it occurs at a higher level between ancestors of the activators.

Table 1. Some axioms in the DFD process model

Let us see how our framework can be applied to this model:

System: we see that DFDs support the notion of entities outside the system (external entities) and inside the system (dataflows). There is no concept of roles. They do not distinguish between entities (all internal entities are considered to be *information entities*).
State: The state is represented by dataflows. Thus, a dataflow plays the part of both, an internal entity and the *state_type* of the entity. *E.g.*, a dataflow called "unsigned_purchase_order" may go to a process called "purchaser", who may sign it and a dataflow called "signed_purchase_order" (the same entity in a new state) will emanate from the "purchaser".
Space and Time: None of the sub-dimensions of the space or time dimensions are explicitly supported by DFDs. However, they could be informally mentioned in the descriptions of the dataflows. *E.g.*, "unsigned_order_in_purchase_dept." captures the spatial location of the purchase order.
Transformation/transportation/storage: In DFDs, processes transform flows. They could also potentially be used to transport flows through space. *E.g.*, A process called "move_information" may transport a flow called "customer_data" across a computer network. Data-stores are used to transport flows through time. Note that it is not possible for a flow to be simultaneously transported through time and be transformed or transported through space (since data-stores are not processes). It *is*, however, possible for a flow to be transported through space, and transformed at the same time (since both these functionalities are served by processes).
Sequencing / Control Flow: DFDs only support the notion of sequencing. The sub-dimensions of *atomic execution, repetition, either/or, while/repeat* and *concurrent* execution are not supported.
Decomposition: Traditionally DFDs allow only the decomposition of processes. Since dataflows model entities and states, the decomposition of these is not explicitly supported. However, the decomposition of *activity_types* (processes) is very well supported. The only *activity_type* whose decomposition is not well supported is one that transports *state_types* through time (since these are modeled using data-stores). Thus, processes can be decomposed in ways that support all the decomposition requirements of activities described earlier, except for the decomposition of *activity_types* that transport *state_types* through time. *E.g.*, an overall process may transform a dataflow "insurance claim". Its decompositions may transform the dataflows: "accident_description" and "repair_cost_estimate". Thus, in a sense, the fact that the decomposition of most *activity_types* is supported and strictly hierarchical, implies that the decomposition of *state_types* is implicitly supported and is also strictly hierarchical. Finally, DFDs do support the notion of primitive processes, that signify the end of a decomposition.
Constraints and Axioms: DFDS do not explicitly support constraints modeling resource scarcity, real-time constraints, spatial constraints, temporal constraints or state_type constraints. The axioms in DFDs do make instances of DFDs unambiguous. However, traditionally DFDs were not formulated in a language that allowed derivation of new rules from axioms.

Advanced Analyses: DFDs do support reachability (by modeling a DFD instance as a directed graph, where every dataflow is a node, and every process an arc, it is possible to determine reachability of a dataflow). They do not support analyses for deadlock, optimality, infinite loops or race conditions.
We now examine some extensions and enhancements to the DFD model, and see how they fit in our framework. [Falkenberg (1991)] proposed a formal specification of DFDs in a language based on predicate logic. This clearly meets the previously unfulfilled requirement that a model must be specified in a language that is formal and allows derivation of rules from axioms. [Kung (1991)] proposed a formal decomposition for processes in DFDs, again using predicate logic as the base language. [Tao (1991)] also proposed a formal specification for DFDs, based on set theory. [Ward (1986)] proposed extensions to DFDs that met requirements in the *sequencing / control flow* and the *time* dimensions of our framework. Our examination of DFDs has shown that the dataflow concept (as well as the process concept, to a lesser degree) is overloaded. [Opdahl (1994)] describe extensions to DFDs that reduce the overloading of the dataflow concept somewhat, in that dataflows and predicate conditions are modeled separately.

The IDEF0 Model

The IDEF0 model [Laamanen (1994)] consists of **boxes**, that represent the system's functions and **arrows** that represent one of: *inputs, outputs, controls* or *mechanisms*, to support the function. A short, and incomplete, list of axioms that are used to ensure consistency in the IDEF0 model is shown in table 2.

• Input, output and control arrows can only have one direction (input and control arrows go into the box, while output arrows come out of the box.
• There can be multiple arrows of any one type for a function.
• Mechanism arrows that come into the box identify the means by which a function performs inputs to outputs. Outward arrows (call arrows) signify that the decomposition at the present level of abstraction is complete, but that further details can be found in lower levels.
• All child diagrams must have at least 3 and at most 6 sub functions.
• Boxes (functions) can be decomposed, following a strict tree hierarchy.
• Arrows are viewed as conduits containing other arrows. At any point, the appropriate detail can be shown, by decomposing or grouping or both.
• Once an arrow belongs to a function, it can only belong to the function, a child or an ancestor.
• Arrows can belong to multiple functions (e.g. input to one function can also be input to another function).
• Arrows can play multiple roles, e.g. output from a function can be input to another function.

Table 2. Some axioms in the IDEF0 model.

Let us see how our framework can be applied to this model:

System: Entities in IDEF0 are modeled using arrows. They can take on one of 4 roles: inputs, outputs, controls and mechanisms. Entities external to the system are not modeled. Entities may be artifacts, information entities or human artifacts, as long as they play one of the 4 roles.
State: The state is represented by arrows. Thus, arrows represent both the entity and the *state_type* of the entity in IDEF0. *E.g.*, "unresolved_operational_issues" may be an input to a box, and the output may be "resolved_operational_issues", which represents a new *state_instance* of the *implicit state_type* "operational_issues_status".
Space and Time: The sub-dimensions in the space and time dimensions are not explicitly modeled in IDEF0. However, as in DFDs, it is possible to capture some aspects of space and time in IDEF0. *E.g.*, the input to a box may be "document_at_location_X", the box may be: "move_document_to_Y", and the output may be "document_at_location_Y".

Transformation / transportation / storage: In IDEF0, **boxes** can be used to transform input arrows. They can also be used to transport input arrows through space (if the input arrows represent entities), or through time (if the input arrows represent states of entities).
Sequencing and control flow: IDEF0 models sequencing of functions. However, *atomic execution, repetition, concurrent, either / or* and *while / repeat* execution are not supported.
Decomposition: In IDEF0, decomposition is primarily done along the box (function) dimension. This means that *activity_type* decomposition is the primary decomposition. In addition, decomposition and aggregation of arrows (in the form of "forking") is supported, although "it is still unclear as to the proper implementation of a fork arrow" [Laamanen (1994)]. Thus, the decomposition of entities and states is not well supported (since arrows represent these). We do not mention examples here, but it should be clear that, since boxes can represent all the *activity_types* in our framework, and since their decomposition is well supported, *activity_type* decomposition is supported along all the sub-dimensions in that dimension, in our framework. Also, as we saw for DFDs, it is possible to model *state_type* decomposition *implicitly*, by decomposing activity_types. The IDEF0 model does model the end of decomposition (boxes that do not have downward arrows emanating from them are not further decomposed).
Constraints and axioms: IDEF0 does not explicitly support constraints modeling resource scarcity, real-time constraints, spatial constraints, temporal constraints or *state_type* constraints. The axioms in IDEF0 make an instance of IDEF0 unambiguous. However, like DFDs, IDEF0 is not formulated in a language that allow derivation of rules from axioms.
Advanced Analyses: IDEF0 does support reachability (by modeling an IDEF0 instance as a directed graph, where every input and output is a node, and every box an arc it is possible to determine reachability of an input or output). It does not support analyses for deadlock, optimality, infinite loops or race conditions.

Differentiating between the 2 models

System: IDEF0 supports roles and different types of entities. DFDs support external entities.
State: We find that dataflows in DFDs and arrows in IDEF0 are overloaded to model both entities and states.
Space and *Time:* This is not formally supported in either model.
Transformation / transportation / storage: In DFDs, processes are used to transform and transport, while data-stores are used for storage. In IDEF0, boxes are used for all 3 aspects.
Sequencing / Control Flow: This is similar in both models.
Decomposition: Both models support *activity_type* decomposition. Neither supports the decomposition of entities or states.
Constraints and axioms: Both models have axioms that signify correctness, and unambiguity of an instance of the model.
Advanced Analyses: Both models allow analyses for reachability of state, but none of the other analyses are supported.

CONCLUSION

The content specification proposed here can be used as a framework for a detailed analysis of business process models, based on the content and the aspects of a business process they model. Two widely used process models: DFDs and IDEF0 were analyzed as a demonstration of the use of this framework.
The content specification defined here is part of a larger research project whose overall goal is to provide support to workflow management using a database management system. We plan to use this content specification to formally define a business process model that will meet all the requirements listed here. We anticipate the new model will contribute to solving the two problems identified earlier: the problem of difficulty of use amongst end-users and the problem of capturing all the concepts required by a business process.

REFERENCES

Attie, P.C., Singh, M.P. Sheth, A. & Rusinkiewicz, M. **Specifying and Enforcing Intertask Dependencies**. Proceedings of the 19th VLDB, Dublin, Ireland. pp. 134-145, 1993.
Barker, R. & Longman, C. **CASE*Method: Function and Process Modelling**. Oracle Corp., UK Ltd.
Booch, G. **Object Oriented Design with Applications**. Benjamin-Cummings, 1994.
Curtis, W., Kellner, M.I. & Over, J. **Process Modeling**. Communcns. of the ACM. vol. 35, pp. 75-90, 1992.
DeMarco, T. **Structured Analysis and System Specification**. Yourdon, Inc., 1978.

Falkenberg, E.D., van der Pols, R. & van der Weide, Th. P. **Understanding Process Structure Diagrams**. Information Systems, vol. 16, no. 4, pp. 417-428 (1991).

Gane, C. and Sarson, T. **Structured System Analysis**. McDonnell Douglas, 1982.

Georgakopoulos, D. & Hornick, M. **A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows**. International Journal of Intelligent and Cooperative Information Systems. vol. 3, pp. 225-253, 1994.

Georgakopoulos, D., Hornick, M. & Sheth, A. **An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.** Distributed and Parallel Databases. vol. 3, pp. 119 - 153, 1995.

Joosten, S. **Trigger Modeling for Workflow Analysis**. Proc. CON 1994: Workflow Management, pp. 236-247, 1994.

Joosten, S. **Conceptual Theory for Workflow Managament Support Systems**. Tech. Report. Center for Telematics and Information Technology, Univ. of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

Kramer, B. and Luqi. **Towards Formal Models of Software Engineering Processes**. J. Systems and Software, vol. 15, pp. 63-74, 1991.

Kung, C. **Process Interface Modeling and Consistency Checking**. J. Systems and Software. vol. 15, pp. 185 - 191, 1991.

Laamanen, M.T. **The IDEF Standards. Methods and Associated Tools for the Information Systems Life Cycle** IFIP (A-55) A.A. Verrjin-Stuart and T.W. Olle (Ed.). Elsevier Science B.V. (North-Holland), 1994.

Martin, J. **Recommended Diagramming Standards for Analysts and Programmers: A Basis for Automation**. Prentice-Hall, Inc. (1985).

Olle, T.W. et al. (Eds.) **Proceeding of the IFIP WG 8.1 Working Conference on Comparative Review of ISD methodologies: Improving the Practice,** Borth-Holland, The Netherlands, 1986.

Opdahl, A.L. & Sindre, G. **A Taxonomy for Real-World Modelling Concepts**. Information Systems, vol. 19, no. 3, pp. 229 - 241, 1994.

Rubin, K.S. & Goldberg, A. **Object Behavior Analysis**. Commncns. of the ACM. vol. 35, pp. 48-62, 1992.

Tao, Y and Kung, C. **Formal Definition and Verification of Data Flow Diagrams**. J. Systems and Software, vol. 16, pp. 29-36, 1991.

Tolvanen, J. & Lyytinen, K. **Modeling Information Systems in Business Development: Alternative perspectives on business process re-engineering**. IFIP, (A-54) B.C. Glasson, et al. (Ed.). Elsevier Science B.V. (North-Holland), 1994.

Ward, P.T. **The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing**. IEEE Trans. Software Engg. vol. SE-12, pp. 198-210, Feb. 1986.