SEAM: A State-Entity-Activity-Model for a Well-Defined Workflow Development Methodology

Akhilesh Bajaj and Sudha Ram, *Member*, *IEEE Computer Society*

Abstract—Current conceptual workflow models use either informally defined conceptual models or several formally defined conceptual models that capture different aspects of the workflow, e.g., the data, process, and organizational aspects of the workflow. To the best of our knowledge, there are no algorithms that can amalgamate these models to yield a single view of reality. A fragmented conceptual view is useful for systems analysis and documentation. However, it fails to realize the potential of conceptual models to provide a convenient interface to automate the design and management of workflows. **First**, as a step toward accomplishing this objective, we propose SEAM (State-Entity-Activity-Model), a conceptual workflow model defined in terms of set theory. **Second**, no attempt has been made, to the best of our knowledge, to incorporate time into a conceptual workflow model. SEAM incorporates the temporal aspect of workflows. **Third,** we apply SEAM to a real-life organizational unit's workflows. In this work, we show a subset of the workflows modeled for this organization using SEAM. We also demonstrate, via a prototype application, how the SEAM schema can be implemented on a relational database management system. We present the lessons we learned about the advantages obtained for the organization and, for developers who choose to use SEAM, we also present potential pitfalls in using the SEAM methodology to build workflow systems on relational platforms. The information contained in this work is sufficient enough to allow application developers to utilize SEAM as a methodology to analyze, design, and construct workflow applications on current relational database management systems. The definition of SEAM as a context-free grammar, definition of its semantics, and its mapping to relational platforms should be sufficient also, to allow the construction of an automated workflow design and construction tool with SEAM as the user interface.

Index Terms—Workflow systems, data modeling, process modeling, relational databases, temporal models, requirements engineering.

1 Introduction

Recently, the design and construction of Workflow Management Systems (WFMS) has emerged as an important area in both theory and practice (e.g., [1], [2], [3], [4], [5]). We borrow our definition of a workflow from [1] who defines a **workflow** as a collection of tasks organized to accomplish some business objective. This definition conforms to a widely accepted understanding of a workflow in the literature (e.g., [6], [7]). Hence, in this work, the notion of a "business process" is synonymous with that of a "workflow."

Current methodologies that are used to specify workflows at a conceptual level involve using several informal or semiformal models to model different aspects of the workflow, such as the data aspect, the process aspect, and the organizational aspect. This fragmented conceptual view is useful for the purpose of systems analysis and documentation, but it does not enable the model to formally assist in the construction of the workflow or in its management. In this work, we propose a solution to this problem, in the form of SEAM (State-Entity-Activity-Model), a single conceptual workflow model that we formally define and specify as a context-free grammar. We also provide mapping rules that map SEAM schemas to abstractions that are supported by commercial relational database management systems (RDBMS). We discuss the feasibility of using SEAM for modeling real-world workflows, based on a real-life case study. Finally, to demonstrate the feasibility of support for SEAM concepts on commercial RDBMSs, we present a prototype workflow application built on a commercially available RDBMS.

The primary motivation for this work is to present sufficient information so that automated tools (in the form of compilers) can be constructed that will map a SEAM model schema to abstractions supported by a RDBMS and, thus, aid in the construction of workflow applications. A secondary motivation is providing guidelines for the development of RDBMSs themselves, as they evolve to support workflows. Thus, some of the concepts in SEAM (e.g., the temporal concepts) may be modeled directly as abstractions in future RDBMSs, or implemented separately as cartridges in object relational databases.

Having briefly described the motivation for this work, we next present the research context of this work.

Several issues have been identified in workflow research, and specialized implementations have been proposed and, in some cases prototyped, to resolve these issues. The issues can be broadly divided into *implementation* issues and *business process modeling* issues. Implementation issues are

- *A. Bajaj is with the Heinz School, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: akhilesh@andrew.cmu.edu.*
- *S. Ram is with the Department of MIS, Eller College of Business, 430J McClelland Hall, University of Tuscon, AZ 85721. E-mail: ram@bpa.arizona.edu.*

Manuscript received 9 July 1998; revised 22 Mar. 2000; accepted 7 Sept. 2000. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 107025.

the low-level changes that need to be made to applications that deal directly with the operating system and/or hardware. Business process modeling issues start at the level of capturing descriptions of end-user requirements and then go down to varying levels of actually implementing the system.

Implementation issues include newer transaction models, interworkflow communication architectures, and managing workflows in heterogeneous and distributed (HAD) environments. For example, [8] proposes a data model and nonisolated transaction model as part of their WFMS. An active, object-oriented database management system (DBMS) is used by [6] to build their WFMS. A WFMS is constructed and presented in [9] using FlowMark [10], which is a specialized workflow system product. An architecture for exception handling in interorganizational workflows is proposed in [11]. A specialized messaging bus architecture to manage workflows, and allow for better communication is described in [12]. An argument for why serializability is too strict a criterion for workflow transactions is presented in [13], who describe a distributed agent architecture that can be used to develop workflow applications. A synthesis of different transaction models is presented in [14]. The management of workflows in distributed environments is demonstrated in [15]. A light workflow system based on Petri Nets is demonstrated in [16]. The materialization of object-oriented views and their application to workflow systems is presented in [17]. A transaction oriented workflow environment is presented as a class library by [18]. The class library provides features for the construction of long-lived, concurrent, nested, multithreaded activities. A dynamic and distributed environment for task scheduling called FlowAgent is presented in [19]. Specialized implementations offer the advantages of better performance and efficiency, and in some cases, facilities that are simply not offered by existing systems.

In **business process modeling**, several conceptual models (e.g., [20], [21], [22], [23]) have been proposed. Many of these models share common concepts such as the data, activities, controls, and organizational responsibilities involved in the workflow. Some models are informally defined [24], [25]. Thus, user workflows are modeled using operational user profiles in [26], when evaluating the quality of service in Web-based education systems. The incidence and causes of human and organizational error are captured in a model proposed in [27]. This model is used to suggest strategies for business process reengineering that minimize specific errors. A knowledge-based approach to depicting workflow exceptions is presented in [28], wherein the goal is to assist users in managing exceptions that may arise. Operational parameterized building blocks are used to construct workflow schema in [29].

Other modeling environments (e.g., [20], [22], [30]) use several semiformally defined models to model different aspects of the business process, such as the data aspects, the activity aspects, and the organizational aspects. In [31], workflows are modeled as reactive objects that only obey ECA (event-condition-action) rules. These models help in the analysis phase, as well as in documenting user requirements. However, as pointed out in [16], the potential of conceptual models to formally *assist* in the construction of a workflow system and help *manage* the workflows has not been realized. In this work, we attempt to address this problem.

In order to make workflow systems that can be specified and managed at the conceptual level, we propose that the following requirements need to be satisfied: First, a single conceptual model is needed to represent the entire system. Second, this conceptual model must be formally specified and be able to map to a reasonably low level of workflow design. Third, the conceptual model must be easy to use and scalable for large real-world applications. Now, we explain each of these issues.

A single conceptual model is required because, to the best of our knowledge, no algorithms exist to amalgamate diverse data and process models to form a *single view of the reality* being modeled. A single view of the reality is needed at the conceptual level before any mapping to lower levels can be done. Support for this argument can be found in [32].

The model needs to be formal so that its semantics are well-understood, and a tool (such as a compiler) can be constructed to map the model to lower levels. Thus, a single, mappable conceptual workflow model will facilitate the construction of a workflow design tool and also make it possible to manage workflows by making changes via the conceptual level.

Finally, the model needs to be easy to use so that it scales well to modeling real-world workflows. Thus, a model that captures information about workflows down to the level of the actual code will result in extremely complex schemas when applied to modeling real-world workflows. Note the trade-off here between the depth of the implementation level to which the conceptual model can be mapped and the ease of use of the model. Thus, a model A that supports the process titles with verbal descriptions for describing the processes will be easier to use and more scalable at the conceptual level than another model B that formally models the entire logic of the processes using, say, finite state machines. However, model B will map to a lower level of implementation than model A.

In the well-established area of database application construction, such a model already exists and is widely used. The methodology based on this model is briefly summarized in Fig. 1a. In the methodology, a single conceptual data model (the Entity Relationship Model (ERM) [33]) is used to capture the end-user's reality. The mapping from the conceptual to the implementation model (the relational data model [34]) is well-defined and can be automated. We call such a methodology a "well-defined" methodology. Such a methodology is necessary in order to be able to construct and manage database applications from the conceptual level itself. Several prototypes of automated design tools using this well-defined methodology (e.g., [35], [36]) have been constructed.

In Fig. 1b, we depict the primary contribution of this work: **a well-defined workflow systems development methodology** that can be used in the analysis, design, and construction of workflow systems using current relational database management system platforms. To achieve this, we present SEAM, a single conceptual workflow model. We define its semantics in terms of set theory, and define the set

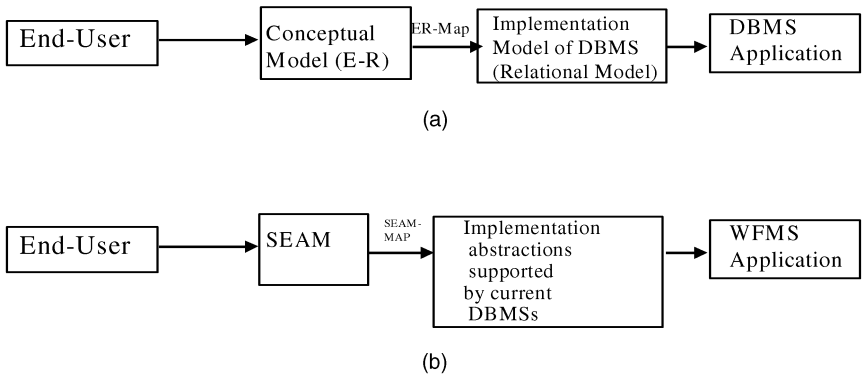

Fig. 1. The analogy between an established, well-defined database methodology and the proposed well-defined workflow methodology. (a) An existing well-defined methodology to create workflow applications using the relational model. (b) A possible well-defined methodology to create workflow applications using implementation abstractions supported by current DBMSs.

of legal model schemas as a formal context-free grammar. We also present a set of rules that map from SEAM to existing RDBMS platforms, such as Sybase [37] and Oracle [38]. While SEAM may similarly be mapped to other specialized workflow implementations, we have two reasons for selecting commercially available relational implementations as the target in this work. **First**, specialized implementations require substantial financial investment, which is beyond the purview of most organizations [39]. **Second**, most organizations have personnel who are familiar with the environments of these commercial systems [40], [41] and, hence, have a greater incentive to utilize these systems to the fullest, instead of moving to more specialized environments. Hence, in this work, we present an unambiguous mapping of a SEAM schema to abstractions supported by current RDBMS platforms. A similar mapping to other, more specialized implementation platforms may be similarly done.

To the best of our knowledge, SEAM is the first conceptual workflow model that explicitly incorporates the representation of time. While the representation of time in conceptual data models is a well-researched area (e.g., [42]), our work makes a first attempt at representing time in a conceptual workflow model (that represents **both data and processes**).

The rest of this paper is organized as follows: In Section 2, we define SEAM. The mapping from SEAM to abstractions supported by current RDBMSs is presented in Section 3. In Section 4, we illustrate, with a simple prototype application, how SEAM can be used to construct a workflow application using an existing RDBMS, and also discuss how it compares with other models. In Section 5, we conclude with a discussion of the contributions of this work and directions for future research.

2 The State-Entity-Activity-Model (SEAM)

First, we present the intuition behind creating SEAM. Our design philosophy was to create a model that captures data and processes in well-understood ways. Hence, data in SEAM is based heavily on the ERM, which is a well-understood model in the academic and practitioner communities. Data in SEAM is captured using the notion of $entity\_types$ and $state\_types$. Both have attributes, and $entity\_types$ have a $primary\_key$ susbet of attributes. Each instance of an $entity\_type$ and a $state\_type$ has time stamps associated with it, which fix its existence in time. $Entity\_instances$ from one or more $entity\_types$ belong to $state\_instances$ (instances of $state\_types$). The issue of temporal consistency, so that the $entity\_instances$ that belong to a $state\_instance$ do not differ by more than a certain time interval, is handled by the $max\_interval$ value of the $state\_type$.

Processes in SEAM are modeled using a simple activity decomposition model that is similar to models like the data flow diagram model [43], which is a well-understood process model. $Activity\_types$ in SEAM can be decomposed down to the level of primitive $activity\_types$. Each $activity\_type$ acts_on a $state\_type$ (and the $entity\_types$ that belong to the $state\_type$). Primitive activities are further described using a verbal description, similar to mini specs in the data flow diagram model.

SEAM differs from other models in that, first, it explicitly links activities and data, thus enforcing one consistent view of reality. Second, it introduces temporal concepts in a workflow model, which we believe is novel. Third, SEAM is formally defined in this work, thereby allowing the construction of software that can be used to manage workflows from the SEAM level. In Section 5, we discuss in greater detail the differences between SEAM and other modeling methods.

Now, we define the components and semantics of SEAM using set theory in Section 2.1. Fig. 2 lists many of the symbols we use. Next, SEAM is specified as a context-free grammar in Section 2.2. A graphical notation to depict SEAM is defined in Section 2.3. An example, taken from a real-world case study to illustrate the usage of SEAM is shown in Section 2.4.

2.1 The Definition of SEAM

2.1.1 $Entity\_types$ and $Entity\_type$ Descriptors in SEAM

Entity_type. An $entity\_type$ is a set of $entity\_instances$, each of which is described by the same set of mappings (entity descriptors):

$$E = \{e_i \; \forall \; i = 1 \ldots n\}.$$

E: an entity_type
e: an entity_instance
S: a state_type
s: a state_instance
A: an activity_type
a: an activity_instance
W: a workflow_type
w: a workflow_instance
E: the set of all entity_types in a SEAM scheme
S: the set of all state_types in a SEAM scheme
A: the set of all activity_types in a SEAM scheme
R^+: set of positive real numbers

Fig. 2. Symbols used in the SEAM specification.

Entity_Attribute. An entity_attribute is a functional mapping from an *entity_type* into a value set or a Cartesian product of value sets:

$$Eat : E_i \rightarrow V_i \text{ or } V_{i1}XV_{i2}X \ldots XV_{in}.$$

Entity_time_stamp. The time_stamp of an *entity_instance* fixes its location in time (relative to the start of the workflow). It is defined as a functional mapping of an *entity_type* to the value set consisting of positive real numbers:

$$ets : E_i \rightarrow V_{ti} \text{ where } V_{ti} = \{\text{x}|\text{x} \in \mathbf{R}^+\}.$$

Primary_key. Each *entity_type* has at least one minimal subset of entity_attributes, say, P, such that the mapping from this subset to the Cartesian product of the corresponding value sets is a one-one mapping. The primary key for the *entity_type* is given by:

$$P \cup \text{ets } (E_i).$$

2.1.2 *State_types* and *State_type* Descriptors in SEAM

State_type. A *state_type* is a mathematical relation between n *entity_types*. It is defined as an n-tuple of these entity types, with the constraint of **temporal consistency**: The time_stamps of all the *entity_instances* that belong_to a *state_instance*, as well as the time_stamp of the *state_instance*, should fall within a particular length of time:

$$S_i = \{[e_1, e_2, \ldots, e_n]|e_1 \in E_1, e_2 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n \wedge \forall\, j = 1 \ldots n, sts(s_j) - ets(e_j) <= +/- smi(s_j)\}.$$

We say that $E_1, \ldots, E_n$ belong_to *state_type* S_i. Each tuple of the *state_type* relation is a *state_instance*.

State_attributes. A state_attribute maps from a *state_type* into a value set or a Cartesian product of value sets:

$$Sat : S_i \rightarrow V_i \text{ or } V_{i1}XV_{i2}X \ldots XV_{in}.$$

State_time_stamp. The state_time_stamp fixes the location of a *state_instance* in time (relative to the start of a workflow). It is defined as a functional mapping of a *state_type* to a value set consisting of positive real numbers:

$$sts : S_i \rightarrow V_{ti} \text{ where } V_{ti} = \{\text{x}|\text{x} \in \mathbf{R}^+\}.$$

Max_interval. The max_interval for a *state_type* is the length of the temporal interval within which the entity_time_stamps of all *entity_instances*, as well as the state_time_stamp of the corresponding *state_instance* have to fall, so that the *state_instance* represents a temporally consistent view of reality. It is a functional mapping from a *state_type* to the value set V_{ti}:

$$smi : S_i \rightarrow V_{ti} \text{ where } V_{ti} = \{x|x \in \mathbf{R}^+\}.$$

2.1.3 *Activity_types* and *Activity_type* Descriptors in SEAM

Activity_type. An *activity_type* is a transformation that maps from a *state_type* to the *state_type* itself. It is an ordered binary relation on the *state_type*, where the binary relation represents the possible initial and final *state_instance* pairs that can occur, for an *activity_instance* of the *activity_type*:

$A_i = \{s_j, , s_k, \in S_l|s_j$ is transformed to s_k following a predefined logic of $A_i : [s_j, , s_k,]\}$.

Each element $a_m \in A_i$ is a tuple in this ordered, binary relation. We say that A_i acts_on S_l.

A_Type. The a_type of an *activity_type* describes whether the *activity_type* is fully automatable, nonautomatable, or mixed. A mixed *activity_type* is one whose sub_*activity_types* are either automatable, nonautomatable, or mixed. A_type is a functional mapping from an *activity_type* to the set set

$$V_{at} = \{\text{automatable, nonautomatable, mixed}\}.$$

If $a_t(a_m) =''$ automatable,'' then a_m has to be a functional relation:

$$at : A_i \rightarrow V_{at}.$$

2.1.4 *Workflow_types* in SEAM

In SEAM, a *workflow_type* is defined as every *activity_type* that occurs at the highest level of a SEAM scheme. The *activity_instances* of the *activity_type* are the *workflow_instances*. We say that all the component *activity_types* of a *workflow_type* participate_in the *workflow_type*.

2.1.5 Cardinalities

Similar to the ERM, the cardinality of a *state_type* is a constraint on the mappings from one *entity_type* that belongs_to the *state_type*, to all other *entity_types* that belong_to the *state_type*. Thus, a 1 to n mapping means that each *entity_instance* in the second *entity_type* can only belong_to a *state_instance* with one *entity_instance* of the first *entity_type*.

2.1.6 Decomposition and Precedence in SEAM

State_type **Decomposition**. A *state_type* can be decomposed into two or more *state_types* that describe it in more detail. *State_type* decomposition is defined as a relation between two *state_types*, where the first is a component of the second:

$$sdc \subseteq S_1 \times S_2.$$

We say that S_2 *superstate* S_1 iff S_1 *sdc* $S_2 \vee S_1 = S2$.

Activity_type **Decomposition**. An *activity_type* can be decomposed into two or more *activity_types* that describe it in more detail. *Activity_type* decomposition is defined as a

relation between two *activity_types*, where the first is a component of the second:

$$adc \subseteq A_1 \times A_2$$

A_1 *adc* A_2 iff $\exists\ S_1$, S_2 such that A_1 acts_on $S_1 \wedge A_2$ acts_on $S_2 \wedge S_1$ *sdc* S_2.

We say that A_2 *superactivity* A_1 iff $A_1\ \mathrm{adc} A_2 \vee A_1 = A_2$.

***Entity_type* Abstractions**. SEAM supports the well-known *entity_type* abstractions of generalization and aggregation. These abstractions have been defined and extensively discussed in the literature [44], [45], [46].

Generalization occurs when similar *entity_types* are abstracted to form a higher-order, generic *entity_type*. Let all the entity_atributes of an entity_type E_i be $Attr_i$. From the definition of *entity_types* earlier, the *entity_type* E_i has at least one minimal subset of $Attr_i$, say, P_i, such that the mapping from this subset to the Cartesian product of the corresponding value sets is a one-one mapping.[1] We define generalization as follows:

$$E_1\ sub\_class\_of\ E_2 \text{ iff } P_1 = P_2 \wedge Attr_2 \subseteq Attr_1.$$

Aggregation occurs when two or more *entity_types* and the *state_type* that they belong_to are abstracted to form a higher level *entity_type*. We define aggregation as follows:

A *state_type* S_1 and all the entity_types $E_1, \ldots, E_n$ that belong_to *state_type* S_1 form an aggregate entity_type

$$E_{aggr} => P_{aggr} = (P_1 \cup P_2 \cup \ldots \cup P_n) \wedge Attr_{aggr} \subseteq Attr_{S1} \wedge ets(E_{aggr}) = sts(S_1).$$

2.1.7 Decomposition Primitives

A *state_type* S_1 is a primitive *state_type* if it does not have any component *state_types* in the SEAM scheme. Thus, S_1 is a primitive *state_type* iff only one entity _type E_1 belongs_to $S_1 \vee \neg\exists$ *state_type* S_i such that S_1 *superstate* S_i.

An *activity_type* A_1 is a primitive if it does not have any component *activity_types* in the SEAM scheme. Thus, A_1 is a primitive iff $\neg\exists\ A_i$ such that A_1 superactivity A_i.

Each primitive *activity_type* is further described by a pseudocode of operations on the descriptors of the *state_type* that it acts_on, and the *entity_types* that belong_to that *state_type*.

This is exactly analogous to the practice of using minispecs in process models like the Data Flow Diagram [43], and is used to complete the description of the primitive *activity_types*. Note that this pseudocode is not part of the SEAM specification, but is an add-on that can help in application construction.

2.1.8 Precedence

Precedence is defined as a relation between two *activity_types*:

$$precedes \subseteq \mathbf{A} \times \mathbf{A}.$$

A_i *precedes* A_j iff $\forall w_k, a_i \in A_i \wedge a_j \in A_j \wedge a_i$ participates_in $w_k \wedge a_i$ acts_on $s_m \Rightarrow \mathrm{sts}(a_i(s_l)) < \mathrm{sts}(s_m)$, where $a_i(s_l)$ represents the *state_instance* after a_i has acted_on s_l.

1. In a nontemporal model, this would be the primary key of the *entity_type*.

2.1.9 Axioms and the Construction of SEAM Schema

Axioms. In order to make every SEAM scheme logically consistent, we define the following axioms:

1. **Entity Generalization Axiom.** $\forall\ E_i$, E_j, S_k, E_j *sub_class_of* $E_i \wedge E_i$ *belongs_to* $S_k \Rightarrow E_j$ *belong_to* S_k.
2. **State Decomposition Axioms.**
   a. $\forall\ S_i$, S_j, A_i, S_i *superstate* $S_j \wedge A_i$ *acts_on* $S_i \Rightarrow A_i$ *acts_on* S_j.
   b. S_1 *sdc* $S_2 \Rightarrow$ $(\forall\ E_i, E_i$ *belongs_to* $S_1 \Rightarrow$ $(E_i$ *belongs_to* $S_2))$.
3. ***Activity_type* Decomposition Axiom.** $\forall\ A_i$, A_j, A_k, A_m, A_i *superactivity* $A_j \wedge A_k$ *superactivity* $A_m \wedge A_i$ *precedes* $A_k \Rightarrow A_j$ *precedes* A_m.

The Construction of SEAM Schema. Each SEAM schema consists of different levels. Each level consists of a static scheme and a dynamic scheme. The static scheme depicts the *state_types*, *entity_types*, and *activity_types*, along with their descriptors. The dynamic scheme depicts the sequencing of the *activity_types* depicted in the static scheme, along with predicates.

A static scheme is defined as a 5-tuple

$$St_i = [\mathbf{O}, acts\_on, belongs\_to, sub\_class\_of, aggregate\_of],$$

where $\mathbf{O} \subseteq \mathbf{E} \cup \mathbf{A} \cup \mathbf{S}$ is a nonempty, finite set of components consisting of pairwise disjoint sets: **E**, the set of *entity_types*, **S**, the set of *state_types*, and **A**, the set of *activity_types*;

1. $acts\_on \subseteq \mathbf{A} \times \mathbf{E}$;
2. $belongs\_to \subseteq \mathbf{E} \times \mathbf{S}$;
3. $sub\_class\_of \subseteq \mathbf{E} \times \mathbf{E}$;
4. $aggregate\_of \subseteq \mathbf{E} \times \mathbf{E}$.

A dynamic SEAM scheme is defined as a 2-tuple $Dy_i = [\mathbf{A}, precedes]$.

St_j is lower_than St_i iff

- $\forall\ A_j$ in St_j, $\exists\ A_i$ in St_i, such that A_i *superactivity* $A_j \wedge$
- $\forall\ S_j$ in St_j, $\exists S_i$ in St_i such that S_i *superstate* S_j.

Dy_j is lower_than Dy_i iff

- $\forall\ A_j$ in Dy_j, $\exists\ A_i$ in Dy_i, such that A_i *superactivity* A_j.

2.2 Defining SEAM Schema as a Context-Free Language

We define the set of all possible SEAM schemas as a context free language [47], defined by a nonambiguous, context-free grammar G. The grammar is presented in Fig. 3.

2.3 The Graphical Representation of SEAM

SEAM is a conceptual workflow model that can be used to capture the users' descriptions of their business realities. SEAM schemas are created using graphical symbols (that correspond to the components of SEAM). The graphical symbols we use in SEAM, and the components they represent, are shown in Fig. 4. Next, we present a simple example to illustrate how SEAM can be used to capture user requirements.

G = [S,T,V,P] where S is the start symbol;
T is a set of terminals;
V is a set of non-terminals;
P is a set of production rules.

Extended Backus-Naur Form notation is used to define the grammar. For the sake of clarity, nonterminals are shown in italics while terminals are in boldface and, in some cases, put in quotes for clarity.

S = *model*

T = {**entity_attribute, entity_time_stamp, e_type, sub_class_of, aggregate_of, state_attribute, state_time_stamp, max_interval, a_type, belongs_to, acts_on, precedes, lower_than, ‘,’**}

V = {*entity_type, primary_key, activity_type, state_type, static_scheme, dynamic_scheme, subclass, aggregate, model*}

P = {

model	:	*static_scheme* **’,’** *dynamic_scheme* | *static_scheme* **’,’** *dynamic_scheme* **lower_than** *model*
static_scheme		*state_type* | *entity_type* | *activity_type* **acts_on** *state_type* | *entity_type* **belongs_to** *state_type* | *subclass* | *aggregate* | {*static diagram*}$^+$
subclass	:	*entity_type* **sub_class_of** *entity_type*
aggregate	:	*entity_type* **aggregate_of** *entity_type*
dynamic_scheme	:	*activity_type* | *activity_type* **precedes** *dynamic_scheme*
entity_type	:	{**entity_attribute**}$^+$ **‘,’ entity_time_stamp ’,’ e_type ’,’** *primary_key*
primary_key	:	{**entity_attribute**}$^+$
state_type	:	{**state_attribute**}$^+$ **‘,’ state_time_stamp ‘,’ max_interval**
activity_type	:	**a_type**

}

Fig. 3. Context-free grammar for the SEAM model.

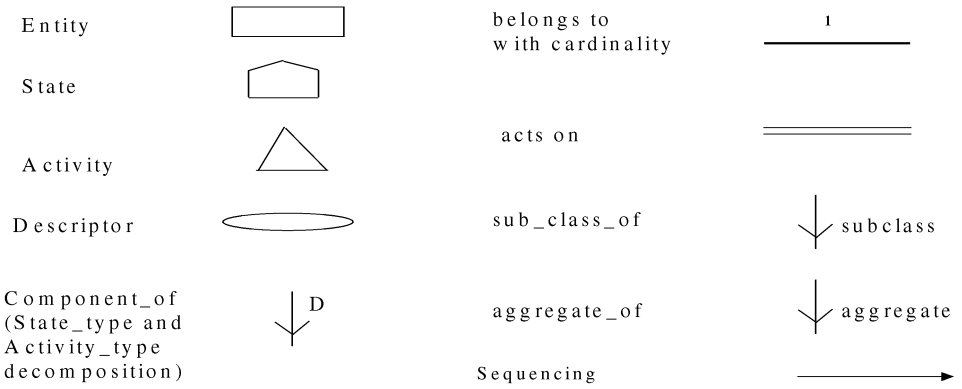

Fig. 4. Constructs in SEAM.

2.4 An Example SEAM Scheme

This example is part of an actual case study that has been conducted where the SEAM schema was created for the workflows of the quality control unit of a medium sized international software development company headquartered in the USA [48]. The case study took three months and involved the intensive participation of four employees of the unit, representing the four different job functions of that unit. For reasons of space, we show a subset of the SEAM schema obtained for the company. We show a SEAM scheme for the following situation: In a software development process, the design specification document (DSD) is created before coding begins. Members of the following departments participate in this creation: implementation (IMP), customer services (CS), technical publications (PUB), development (DEV), marketing (MKTG), and quality control (QC). The QC department consists of a director, several managers, senior analysts, and testers. The DEV department consists of a lead developer and

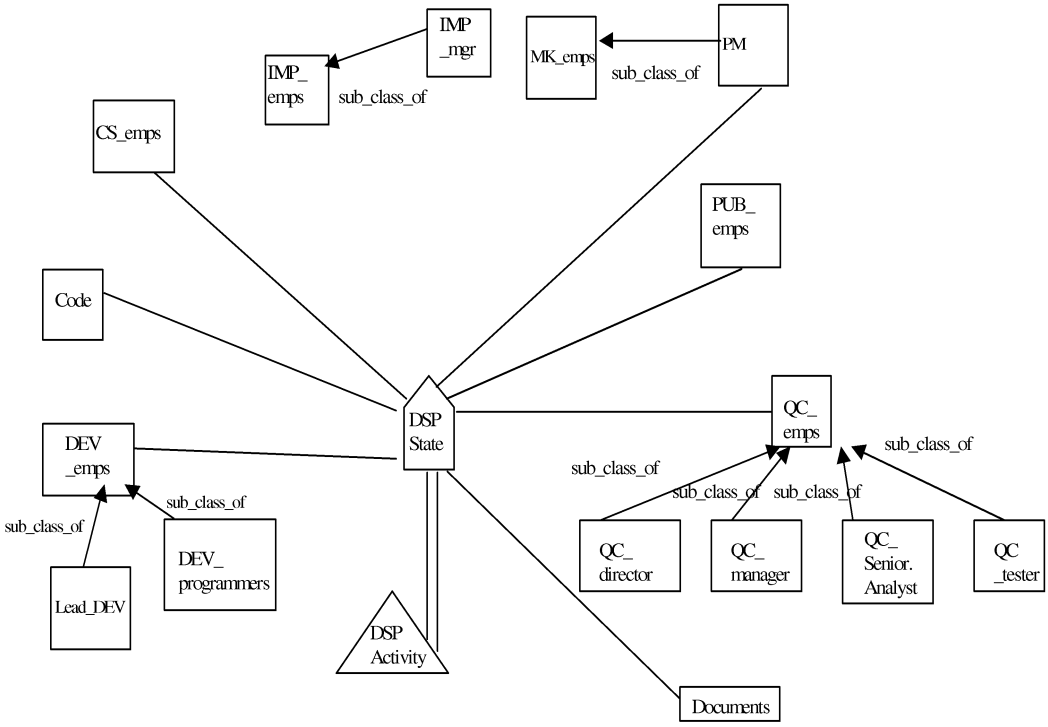

Fig. 5. Level 1 static scheme of DSD Construction.

programmers. The MKTG department includes (among other employees) product managers (PM), one of who participates in the DSD construction process. IMP consists of several managers and employees. While the DSD is being developed, DEV may sometimes demonstrate prototypes of the final system. Prior to developing the DSD, the product planning document (PPD) has already been created. In the final phases of DSD construction, QC starts developing an initial test plan document that sets forth testing plans and criteria for the new system. At this time, QC also identifies any special requirements they may have for testing the new system. The construction of the DSD proceeds in the form of meetings between all of the parties. At a certain point, the preliminary DSD code is approved, and the coding and unit testing (CUT) starts. At this point, all code is written and the units are individually tested. After all the code has been written, the functional integrated testing of the code (FIT) starts. The DSD may be modified during the CUT phase, and appendices may be added to it in the FIT phase.

A simplified SEAM scheme for this situation is shown in Figs. 5, 6, 7, 8, and 9. *The purpose is to* ***illustrate*** *SEAM usage; a SEAM scheme for a real-life system to support this process would be considerably more complex*. Descriptors and cardinalities are not shown for reasons of legibility. Fig. 5 is the top-level static scheme and shows the $workflow\_type$ "DSP activity." Note how the departments are modeled using the generalization concept, just as in the Extended ERM. There is no dynamic scheme at the top level, since there is only one $activity\_type$ (the $workflow\_type$).

At the second level, shown in Fig. 6, the $state\_type$ "DSP_state" is decomposed into three $state\_types$:

1. "$design\_group$,"
2. "$develop\_DSD$," and
3. "$initial\_test\_plan\_state$."

The decomposition relation between $state\_type$ "DSP_state" and these three $state\_types$ is not shown for reasons of legibility. The $workflow\_type$ "DSP activity" has been decomposed into five $activity\_types$:

1. "$develop\_dsd\_activity$,"
2. "$meet\_activity$,"
3. "$provide\_feedback\_on\_DSD$,"
4. "$develop\_plan$," and
5. "$list\_special\_requirements$."

The level 2 dynamic scheme in Fig. 7 shows the sequence of these $activity\_types$.

In level 3, shown in Fig. 8, the $activity\_type$ "develop_DSD_activity" is decomposed into four $activity\_types$. The sequence of these four $activity\_types$ is shown in the level 3 dynamic scheme in Fig. 9. Note that the dynamic schemes at levels 2 (Fig. 7) and 3 (Fig. 9) will need to be superimposed to get the complete picture since only new decompositions are shown at each lower level.

The primitive $state\_types$ in this example are: "design_group," "initial_test_plan_state," and "develop_DSD." The primitive $activity\_types$ are: "develop_plan," "list_special_requirements," "meet_activity," "provide_feedback_on_DSD," "approve_DSD," "draft_preliminary_DSD," "approve_DSD," "modify_DSD_CUT_phase," and "add_appendices_DSD_FIT_phase." It is relatively straightforward to conceptualize further decomposition beyond these three levels. We also do not show the pseudocode for any of the

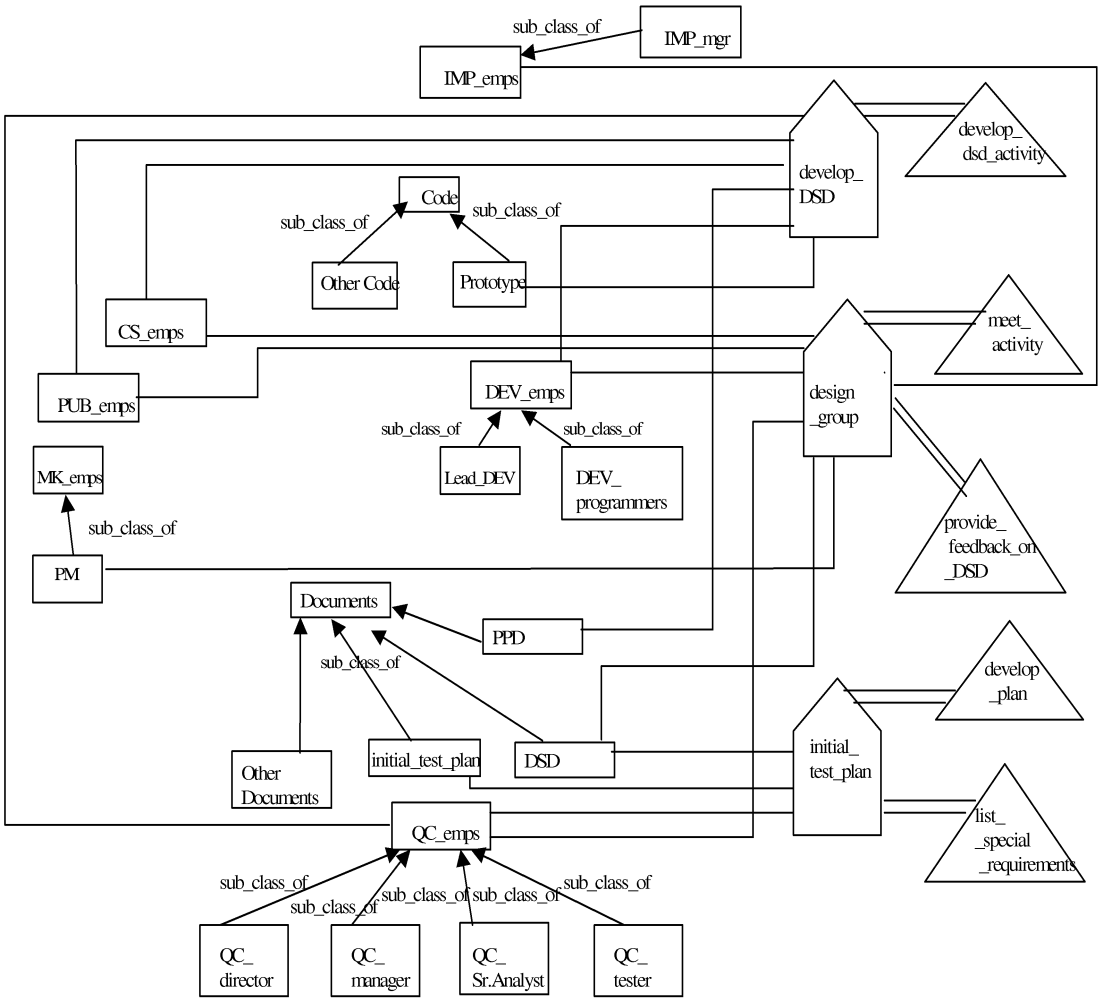

Fig. 6. Level 2 static scheme for DSD Construction.

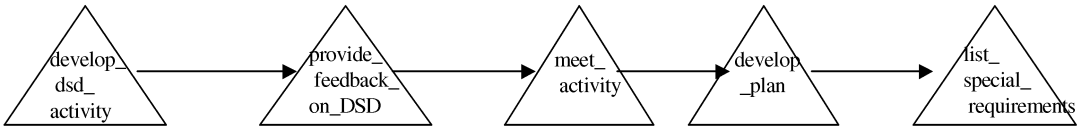

Fig. 7. Level 2 dynamic scheme for DSD Construction.

primitive $activity\_types$ since that is not part of the SEAM specification. However, this would almost certainly be required for completing the documentation in a real-life application.

Having described how user requirements can be captured in a SEAM scheme, we next complete the link shown in Fig. 1b by defining a concise set of rules to map the SEAM schema to abstractions supported by commercial RDBMSs: The relational data model, triggers, a computationally complete (possibly embedded) SQL (Structural Query Language), and the ACID (Atomic, Consistent, Isolated, Durable) transaction model for execution, concurrency and recovery.

3 Mapping SEAM Schema to Widely Supported Abstractions

In the following discussion, we shall refer to relations in the relational data model as "tables," and to the well-known foreign key concept as "referencing." In order to apply these rules to a particular workflow, the term wf_name should be substituted for the actual name of the $workflow\_type$.

3.1 Rules to Map SEAM Schema

1. **Metadata Associated with Each $Workflow\_type$.**

   a. Each $workflow\_type$ has a table,

   $$wf\_name\_wf\_info,$$

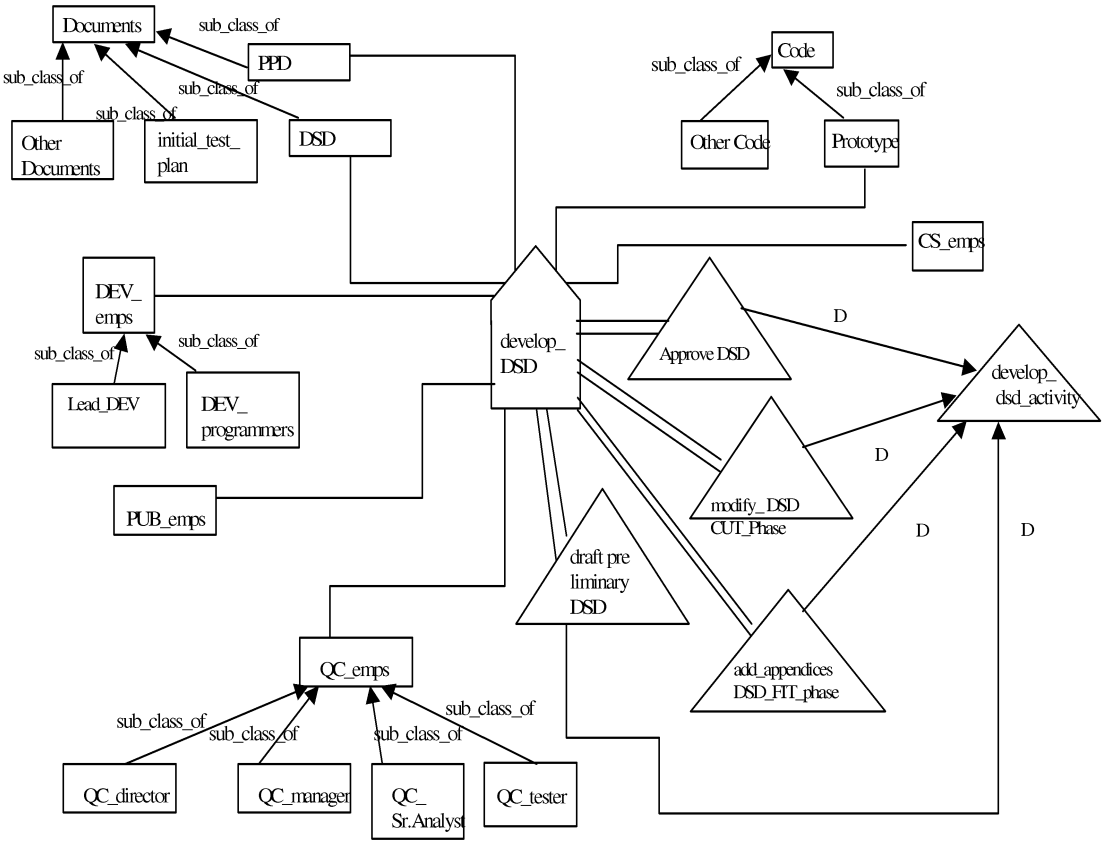

Fig. 8 Level 3 static scheme for DSD Construction.

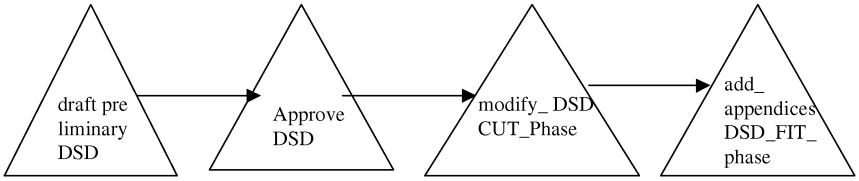

Fig. 9. Level 3 dynamic scheme for DSD Construction.

with the following attributes:

$$(\text{wf\_description}, \text{wf\_id}, \text{wf\_time\_began}).$$

The primary key for this table is wf_id.

b. A table called $wf\_name\_activities\_info$ is used to record information about all the possible primitive $activity\_types$ that participate_in the $workflow\_type$. The table has the following attributes:

$$(\text{activity\_description}, \text{activity\_id}).$$

The primary key is activity_id.

c. A table called $wf\_name\_state\_max\_ints$ is used to record the max_intervals of all the $state\_types$. It has the following attributes:

$$(\text{state\_max\_int\_id}, \text{state\_name}, \text{time1}, \text{time2}).$$

The primary key is the state_max_int_id.

d. A table called $wf\_name\_progress$ is used to coordinate the workflow. It has the following attributes:

$$(\text{activity\_id}, \text{wf\_id}, \text{activity\_time\_began}, \text{activity\_time\_ended}, \text{completed\_status}, \text{executing\_status}).$$

The primary key of the table is (wf_id, activity_id). The wf_id references the wf_name_wf_info table, while the activity_id references the wf_name_activities_info table.

2. **Mapping SEAM** $Entity\_types$**.** Each $entity\_type$ is represented as a table, whose attributes are the same as the $entity\_type$'s descriptors. The primary key of the table is the primary key of the $entity\_type$ (which includes the entity_time_stamp). Since current systems do not support temporal data, time_stamps have to be explicitly modeled as attributes in the table, and each table represents values across time for different $entity\_instances$.

3. **Mapping SEAM Subclass** *Entity_types* **and Aggregate** *Entity_types***.** The mapping from subclass *entity_types* and aggregate *entity_types* to the relational data model has been extensively covered in the literature, and is identical to the mapping of SEAM subclass and aggregate *entity_types*. For reasons of space, we do not discuss the mapping here, but refer the reader to [49].
4. **Mapping SEAM** *State_types***.** Nonprimitive *state_types* are not mapped. Each primitive *state_type* is represented by a table. The attributes of the table include: (state_attributes, state_time_stamp, (primary keys—entity_time_stamp) of all *entity_types* that belong_to the *state_type*).

   In addition, the table has attributes: (state_type_*id*, state_max_int_id, wf_id*1*, activity_id*1*... activity_id*n*, wf_id*2*, activity_id*1*,..., activity_id*p*, ...wf_id*m*, activity_id*1*,... activity_id*q*). Thus, attributes exist for each *workflow_type* that acts_on the *state_type*, and within that, for each *activity_type* that acts_on the *state_type* and that also participates_in the *workflow_type*.

   The primary key for the

   $$\text{table} = \textit{state\_type}\_\text{id} \cup \text{state\_time\_stamp}.$$

   The reason is that, like *entity_instances* in the *entity_type* tables, *state_instances* are also represented across time.

   The state_max_int_id references the

   $$wf\_name\_state\_max\_ints$$

   table, the wf_id references the relevant

   $$wf\_name\_wf\_info,$$

   table, and the activity_ids reference the relevant $wf\_name\_activities\_info$ table.
5. **Generic Triggers to Enforce SEAM Reality.** We identify only the essential triggers that are needed to implement the SEAM conception of reality. In addition, of course, there may be several application-specific triggers, which are not discussed here.

   a. **Triggers to guarantee temporal consistency of data.** Foreign keys are insufficient by themselves to guarantee temporal consistency, *since they do not check to see if the referenced value falls within a particular range of the referencing value*. Hence, we need triggers to ensure temporal consistency.

   - Each *state_type* table has an insert trigger associated with it, to ensure that *entity_instances* exist whose time_stamps lie within the max_interval of the state_time_stamp value. The logical structure of this trigger is shown in Fig. 10a.
   - Each *entity_type* table has a delete trigger associated with it. The trigger ensures that if an *entity_instance* is deleted, then the tables of all the *state_types* to which the *entity_type* belongs are checked. If any *state_instances* are found that depend on this *entity_instance*, and for which no other *entity_instance* can be found, then those *state_instances* must be deleted as well. The logical structure for this trigger is shown in Fig. 10b.

   b. **Workflow Initiation Trigger.** This is associated with insertions into the $wf\_name\_wf\_info$ table. When a row is inserted into this table, it means that wf_name has been instantiated. This trigger then inserts all *activity_types* that participate_in the *workflow_type* in the $wf\_name\_progress$ table, and marks them as incomplete. The logical structure of this trigger is shown in Fig. 10c.
6. **Mapping SEAM Primitive** *Activity_types***.** If the a_type of a primitive *activity_type* is nonautomatable, it may be written as a form.[2] If it is automatable, it is written as a trigger. Each primitive *activity_type* in the SEAM scheme may be written as an embedded SQL module of the form:

```
activity_name() {
   get_input(); /*ask the user for
        possible input that may be
        required
   if it is non-automatable*/
   start_transaction();
      /*
      identify relevant row in the
      state_type table based on
      state_time_stamp and
      transform only the
      entity_instances that lie
      within +/- max_int of the
      state_time_stamp value. After
      relevant entity_instances are
      modified, insert a new row in
      the state_type table that has
      the same wf_id and activity_id
      as the original row
      */
      update_wf_progress(); /*
        updates the relevant record
        in the wf_name_progress
        table, that the activity has
        been completed*/
   commit_transaction();
}
```

This structure implies that each primitive *activity_type* is one transaction. It also supports the SEAM conception of reality: An *activity_instance* acts_on a *state_instance* and all the *entity_instances* that belong_to that *state_instance*. The common wf_id and activity_id in the *state_type* table are sufficient enough to generate the *state_instance* that was input to the *activity_instance*, as well as the

2. A *form* is an event driven interface into the database, which allows end-users access to the data. Many fourth generation languages are widely available to create forms to a database.

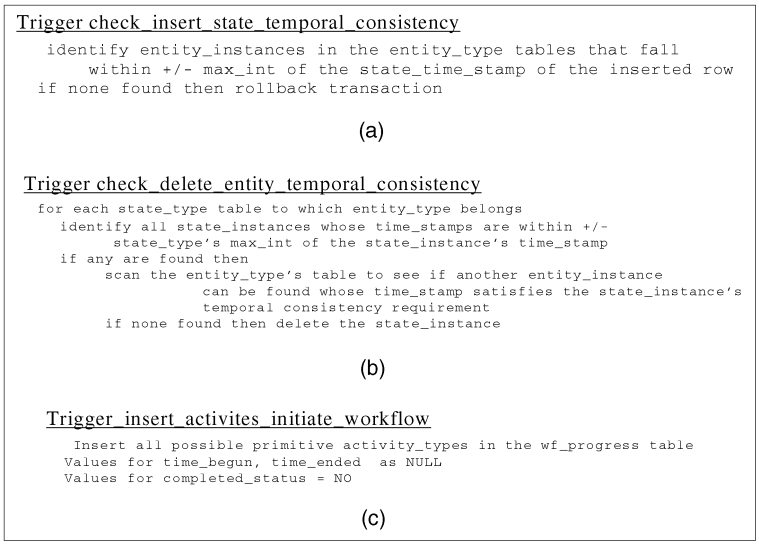

Fig. 10. The logical structure of generic triggers in WF-M. (a) Logical structure of insert trigger associated with each state_table. (b) Logical structure of delete trigger associated with each entity_table. (c) Logical structure of insert trigger associated with wf_name_wf_info table.

state_instance after the *activity_instance* has transformed it.

7. **Mapping SEAM Nonprimitive** *Activity_types***.** Nonprimitive *activity_types* in SEAM are not relevant at the implementation level. Similar to other tools like the DFD, nonprimitive *activity_types* in SEAM serve as a software design aid for decomposing *activity_types*, until primitive *activity_types* that can be coded are reached, e.g., the develop_DSD *activity_type* will not be supported at the implementation level. However, it was useful in arriving at the four primitive *activity_types* shown in Fig. 9.
8. **Managing the Instantiation of SEAM *Workflow_types*.** Each *workflow_type* is managed by a server program (*wf_name_server*) whose structure is as follows:

```
main() {
 while(1) {
        decide_next_activity();
          /*scans the
          wf_name_progress table,
          and picks out an
          activity_type that needs to be
          executed*/
       execute_activity_handler();
          /*a wrapper function that
          accepts the handle of the
          next activity to be
          executed, forks a process
          to execute it &
          immediately returns
          SUCCESS or FAILURE*/
 }
}
```

wf_name_server runs continually. The decide_next_activity function scans the *wf_name_progress* table, sees which *activity_type* has been most recently executed, and understands which *activity_type* needs to be executed next.[3] This information is passed to the execute_activity_handler, which will simply fork a process to execute that *activity_type*'s module and return.

3.2 Workflow Execution, Recovery, and Concurrency Issues

Fig. 11 shows the architecture of a WFMS that can be constructed using SEAM. For each *workflow_type*, *wf_name_server* runs continually. A *workflow_instance* starts when a row is inserted into the *wf_name_info* table. This triggers the workflow initiation trigger (in Fig. 10c), which inserts relevant rows in the *wf_name_progress* table. *wf_name_server* scans this table continually, and decides on the next primitive *activity_type* to be executed. When an *activity_type* is found, it forks a process for the activity and returns. Note that each *activity_type* is written as one transaction. This utilizes the ACID transaction abstraction to ensure recovery from a crash. If the system crashes when a workflow is in progress, the current executing activities are all rolled back (automatically by the DBMS), since they are written as transactions. Since the *wf_name_progress* tables have not been updated (they are the last table an *activity_instance* updates), they remain marked NOT COMPLETE. When the system is restarted, only the *wf_name_server* needs to be restarted for each *workflow_type*. The rolled back *activity_types* are then automatically handled by the servers.

In order to facilitate recovery, information on all *wf_name_server* programs may be stored in a *wf_server_registry* (metametadata). On recovering from a crash, a recovery program could scan this table and start all the servers running again.

Using the transaction abstraction provides the same degree of concurrency to *activity_types* that the DBMS provides to transactions. The key is to decompose

3. There are several ways to implement this. One possible way would be to write a function for each *activity_type* that participate_in the *workflow_type*, that returns true if the *activity_type* is ready for execution.

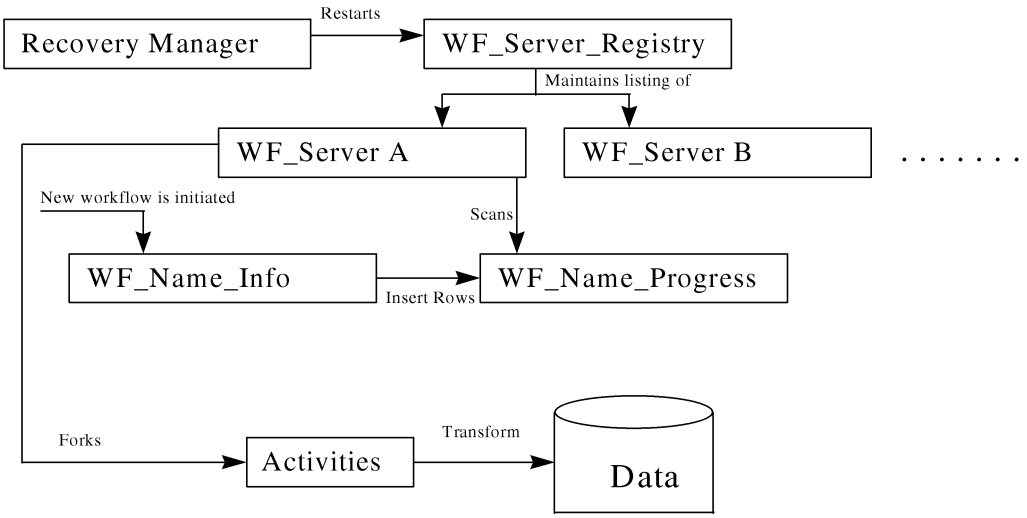

Fig. 11. Architecture of the proposed WFMS.

activity_types so that, finally, each primitive *activity_type* is of "reasonable" duration (i.e., not too long) and can be implemented as a transaction with a reasonable degree of concurrency in the system. SEAM clearly helps in this process by facilitating decomposition.

4 Prototype Workflow Application

Next, we describe a prototype application that we constructed for a SEAM schema. The prototype was developed on Sybase 10 [37], a client-server database management system developed by SYBASE.

Sybase 10 largely supports the relational model, triggers, embedded SQL code, extended SQL, as well as stored procedures. The aim of implementing the prototype was not to implement a real-life application, but rather a proof-of-concept demonstration, to show how a real application can be developed. The top-level static SEAM schema used is shown in Fig. 12. There is one *entity_type*: employees, and one *state_type*: the DSP state. Two *activity_types* transform the DSP state and employees. The first *activity_type* is called Activity 7, and this increases the salary of an employee by 10 percent. It takes between two and four minutes to execute, depending on the system load, and requires input from the user. The second *activity_type* is called Activity 8, and gets the ID of the employee whose salary was increased, and prints out a message to the user. It also takes between two and four minutes to execute, depending on system load. The workflow consists of these two activities executing sequentially. In the prototype, the *employee_id* represents shared information between the two activities in the *workflow_type*. Information is shared by passing along the workflow_id of the *workflow_instance* to each *activity_instance*. Based on this, it can obtain all the tuples that were accessed by previous *activity_instances* in the same *workflow_instance*.

Next, we applied the mapping rules presented in Section 3 to implement this SEAM schema. The relational schema and the constraints on the relational schema (in terms of primary and foreign keys) were derived. The relational schema implemented for the prototype is shown in Fig. 13. The triggers (and stored procedures) that were used to impose SEAM's axioms on the prototype schema were coded next, based on the mapping rules described in Section 3. The programming language Transact-SQL, which is part of the Sybase 10 system, had sufficient features to create these triggers.

Next, the code for the two activities and the code for the dsp_server (the workflow server) was written. The build environment for the activities and dsp_server was created. The code was written in C with embedded SQL calls.

The execution pattern of the prototype is shown in Fig. 14.

The prototype was tested as follows: Triggers were tested by inserting good and bad data (data that violated SEAM axioms) into the tables. An example of bad data is inserting *state_instances* where temporally consistent *entity_instances* do not exist. The *check_insert_state_temporal_consistency* trigger prevented insertion of this data. The *check_delete_entity_temporal_consistency* trigger was tested by deleting *entity_instances* and then verifying that all *state_instances* that were dependent on only that *entity_instance* for temporal consistency were deleted. The *insert_actvities_initiate_workflow* trigger was tested by inserting a row in the dsp_info_table and then verifying that rows for primitive *activity_types* Activity 7 and Activity 8 were created in the dsp_progress table.

The actual workflow (consisting of a sequential execution of the two activities) was executed several times. Each workflow_instance was inititated by inserting a row into the dsp_wf_info table. Several workflow_instances were thus created, and the resulting tables were examined. In all cases, the processes were forked and the triggers worked as intended. Thus, the prototype showed that existing RDBMS platforms offer sufficient abstractions to allow the implementation of a WFMS system using SEAM.

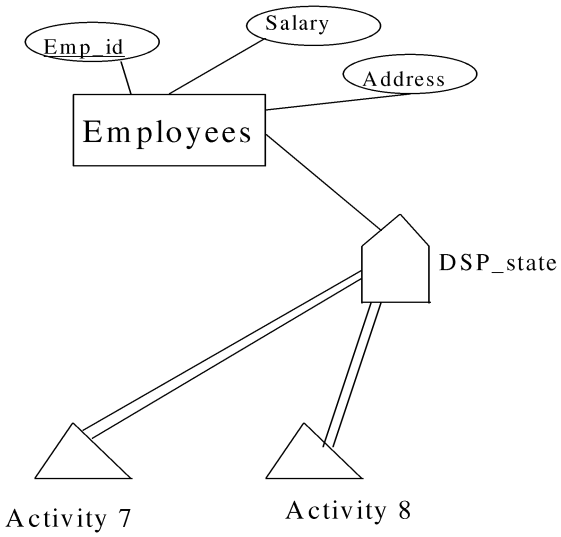

Fig. 12. SEAM schema for the prototype construction.

EMPLOYEES (emp_id, time_stamp, salary, address)

DSP_STATE (emp_id, time_stamp, state_max_int_id, act1_id, act2_id, dsp_wf_id)

DSP_WF_INFO(wf_desc, wf_id, wf_time_begun)

DSP_ACTIVTITES_INFO(act_id, act_desc)

DSP_STATE_MAX_INTS(state_max_int_id, state_name, date1, date2)

DSP_PROGRESS(act_id, wf_id, time_began, time_ended, completed, executing)

Fig. 13. Relational schema for the prototype.

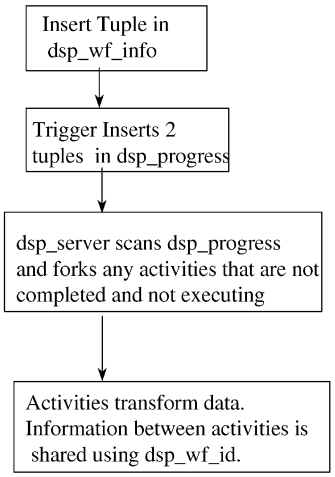

Fig. 14. Execution pattern of the prototype.

While the prototype used a simple example, it is valuable because it demonstrates the end-to-end methodology, of taking a SEAM schema, at the conceptual level, and implementing it as a workflow application on a commercial RDBMS. The prototype covers most of the concepts we anticipate would be required for building a large workflow application, using the SEAM methodology, on a commercial relational platform.

4.1 Lessons Learned about Use the SEAM Methodology in Real-World Situations

We conducted a three month long case study in the quality control unit of a medium-sized software development company based in the USA [50]. There were four job functions in the unit, and the study involved the active participation of four employees, one from each function. At the end of the case study, we had a SEAM diagram of the workflows of the unit. Thirteen levels of decomposition were created. In all, there were 55 *entity_type*s, 60 *state_type*s, and 76 *activity_type*s. A subset of these diagrams has been shown in Figs. 5, 6, 7, 8, and 9. Based on our experience with the case study and the coding of the prototype application, next we summarize important points for developers who choose to use the SEAM methodology.

From the perspective of the **organization**, **first**, SEAM provided, in one diagram, a documentation of the processes and the data. According to the employees who participated in the case study, this was preferable to using several different diagrams and manuals. **Second**, the decomposition in SEAM meant that different levels of abstraction could be represented for processes, making a SEAM diagram suitable for display for a variety of levels of management. **Third**, participating in the creation of a SEAM schema clarified their own workflows for the employees, and increased their knowledge of their own unit.

From the perspective of the **modeler**, SEAM required a lot of effort to use, as compared to simpler diagrams that represent either data or processes. It took longer to both learn how to use SEAM and to actually create the SEAM schema. Thus, it took three weeks to create the SEAM schema, whereas it took only one week to create an IDEF0 [51] schema, which only models processes, for the organization, using the same modeler. However, the consistency between data and processes inherent in SEAM makes it easier, in the opinion of the modeler, than creating separate data and process models, and trying to make them consistent.[4] Thus, developing one SEAM schema is more **scalable** than developing separate, consistent schemas in other modeling methods.

SEAM captures information on *activity_types* at the conceptual level, using verbal descriptions similar to minispecs. There are other models that capture activity logic formally, and are thus similar to programming languages. Based on this case study, we note that extending SEAM to formally capture the detailed logic of each primitive *activity_type* would have resulted in far more complex schemas, and seriously affected the scalability of SEAM when modeling real-world requirements.

One important quality in a conceptual model is completeness, which is the degree to which the model can represent the real-world domain, as perceived by the humans in the domain. The completeness of SEAM was formally evaluated as part of this case study [48], and SEAM was found to be more complete than the IDEF0 model.

4. An example of such a consistency check would be: flows in a data flow diagram have to be represented as data in the data diagram, and all data in the data diagram has to be represented as a flow in the dataflow diagram.

Lessons from implementing the prototype are, **first**, the coding of triggers to enforce the temporal model and the workflow server, (shown in Fig. 10) is initially tedious. This is because it has to be done for each *entity_type* and *state_type*. However, we find that it becomes fairly routine after creating the first set of triggers. We anticipate that any development team choosing to use SEAM will need to quickly establish a template for coding these triggers. **Second**, the actual implementation on an RDBMS platform is relatively simple, once the design is complete. The SEAM diagram serves as an adequate documentation for the workflow system. Thus, this work provides a well-defined methodology for constructing workflow applications, as shown in Fig. 1b.

4.2 Comparing SEAM to Other Models

As mentioned in Section 1, several conceptual models exist to model systems. In this section, we compare SEAM to two popular models: the Software Requirements Engineering Methodology (SREM) [52] and to the Systems Analysis and Design Technique (SADT) [53], [54]. These two models are fairly representative of two broad classes of software engineering tools. SREM models system behavior at low levels and in great detail, while SADT is a higher level conceptual model that model activities and data.

SREM along with the Systems Requirements Engineering methodology (SYSREM) is used to model systems from a higher level of abstraction to a state machine level. SYSREM uses time functions which have inputs, outputs, invariants, completion criteria, and performance. Each time function lasts a finite period of time. It can be decomposed into concurrent and/or sequential activities. At the lowest level, each activity can be modeled as an R-net, which is a graph with several different node types that model the abstractions found in programming languages. R-nets in turn are just one of 21 elements that constitute the Requirements Statement language (RSL). This language allows the specification of all the functions of a system, as well as the conditions under which each one occurs. It also allows the specification of performance criteria and provides for verification of data flow consistency. Table 1 compares SEAM with SREM/SYSREM.

The SADT methodology is representative of a large class of conceptual models that have been used in systems analysis. The SADT technique also spawned the IDEF0 model [51]. The SADT technique focuses on activity diagrams. Activities are boxes, which can be decomposed into subactivities. Arrows represent the inputs, outputs, controls, and mechanisms for the execution of activities. The decomposition of activities is formally defined to ensure preservation of input and output data, but the decomposition of arrows is informal. SADT stresses communication between the developers, analysts, and the customers, when capturing system requirements. Table 2 compares SADT and SEAM.

SEAM appears to successfully bridge the conceptual modeling and implementation layers of workflow application development. Many of the SEAM implementation level concepts (such as time stamps) are not used when conceptually modeling requirements. Based on our experience with using SEAM in a real-world situation, at the

TABLE 1
Comparison between SEAM and SYSREM/SREM

SREM/SYSREM	SEAM
Used primarily to specify any system, at a high level (SYSREM) and a low level (SREM)	Used to conceptually model requirements for business workflows at a high level, similar to data and process conceptual models.
Captures precedence and concurrence at higher levels (SYSREM) , and models behavior in great detail at lower levels (SREM)	Models precedence of activities, and supports decomposition of activities. Primitive activities are modeled using Mini Specs.
Models are large and very detailed since the low level schemas essentially model the complete program behavior	Models will be smaller since modeling is at the conceptual level.
Can be used for any system	Primarily used for business workflows.
Complex model with 21 element types and 23 relationship types.	Simpler model with 3 element types and less than 10 relationship types.
Time functions are decomposed into activities, but temporality of data is not supported.	Explicitly supports the temporal aspects of data in the form of time stamps for *state_types* and *entity_types*, and the notion of temporal consistency.
Time functions are linked to a data processing component, formal support for data modeling is absent	SEAM explicitly models data as in the extended ER model.

conceptual modeling level, SEAM is not more complex to use than well-established models like the ERM and the data flow diagram model. SEAM is a lot easier to use than models that capture detailed logic of activities, such as the SYSREM/SREM.

SEAM is also more scalable than other methodologies that use multiple models. This is because these methodologies have no consistency checks and this allows modelers to create divergent views of the same reality (such as creating data flows that have no counterpart in the data

TABLE 2
Comparison between SEAM and SADT

SADT	SEAM
Is activity centric and data is modeled as arrows.	Is neither activity nor data centric. Data is modeled as *entity_types* and *state_types*, activities are modeled as *activity_types*.
Activity decomposition is formally supported, while data decomposition is not.	Activity and data decomposition is supported formally. Data decomposition is temporal (substates).
No temporal modeling	Explicitly supports the temporal aspects of data in the form of time stamps for *state_types* and *entity_types*, and the notion of temporal consistency.
Captures controls, mechanisms, activities and data	Captures activities and data.
Since data is not modeled formally, mapping to RDBMS abstractions is not supported.	SEAM explicitly models data as in the extended ER model. Mapping to RDBMS abstractions is supported.

model, or creating inconsistent decompositions of data-flows) especially as the size of requirements grows. In SEAM, we have explicit links between activities and data, and there is no redundant depiction of data or process elements since multiple models are not used. This prevents the creation of divergent views by the modeler, when modeling large scale realities.

5 Conclusion

This work contributes to both theory and practice. **First**, SEAM is a rigorously defined conceptual workflow model, that incorporates time. It represents a first step toward a conceptual model driven workflow environment, i.e., an environment where the design and management of workflows is controlled from the conceptual level. Although SEAM can be mapped to any implementation, we have mapped it here to widely used and understood implementation abstractions. The information contained in this work should be sufficient enough to allow application developers to utilize SEAM to model and design workflow applications. The mapping rules presented in Section 3 provide the integrated methodology shown in Fig. 1b. This integrated methodology is the primary contribution of this work.

Second, the formal definition of SEAM semantics and the grammar presented here allow for the construction of an automated design tool consisting of a SEAM compiler that understands the mapping rules. The construction of such a compiler, which is part of our future research, will further automate the construction of workflow systems from SEAM schemas. **Third**, the case study discussed in this work validates the scalability of SEAM in real-world systems. **Fourth**, the mapping rules in this work have also identified abstractions that need to be supported by widely used implementations, in order to model the SEAM conception of reality, e.g., the well-known foreign-key concept can be extended to include checking ranges of values, in order to automate the temporal consistency trigger described in Fig. 10a.

This work also has limitations. The biggest one in our view is the curve associated with learning how to create SEAM diagrams, and actually creating SEAM diagrams in the complexity of a real-life setting. A modeler will need to invest significant time in learning SEAM. However, based on the case study, we feel the advantages of automatic consistency (since everything is in one diagram) and the ability to represent different levels of abstraction will be a sufficient pay-off for modelers who do make the investment in learning SEAM.

This work is part of a larger project that aims at providing a conceptual model driven workflow design and management environment. We have used SEAM to model the workflows of a medium-sized organization. Our future research aims at the construction of an automated workflow design tool prototype, with SEAM as the user interface.

Acknowledgments

The authors would like to acknowledge the comments of Dr. Peter Venable of the School of Computer Science, Carnegie Mellon University, the associate editor and, three anonymous reviewers. All the comments greatly improved the quality of this paper.

References

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

[2] M. Rusinkiewicz, P. Krychniak, and A. Cichocki, "Toward a Model for Multidatabase Transactions," *Int'l J. Intelligent and Cooperative Information Systems,* vol. 1, pp. 579-617, 1992.

[3] J. Davis, W. Du, and M.-C. Shan, "Open-PM: An Enterprise Process Management System," *Bull. Technical Committee on Data Eng.,* vol. 18, pp. 27-32, 1995.

[4] A. Bernstein, D. Chrysanthos, T.W. Malone, and J. Quimby, "Software Tools for a Process Handbook," *Bull. Technical Committee on Data Eng.,* vol. 18, pp. 41-47, 1995.

[5] T. Winograd and R. Flores, *Understanding Computers and Cognition.* Addison Wesley, 1987.

[6] G. Kappel, P. Lang, S. Rausch-Schott, and W. Retschitzegger, "Workflow Management Based on Objects, Rules and Roles," *Bull. of the Technical Committee on Data Eng.,* vol. 18, pp. 11-18, 1995.

[7] S. Joosten, "Trigger Modeling for Workflow Analysis," *Proc. CON: Workflow Management,* 1994.

[8] A. Reuter and F. Schwenkreis, "Contracts: A Low Level Mechanism for Building General Purpose Workflow Management Systems," *Bull. Technical Committee on Data Eng.,* vol. 18, pp. 4-10, 1995.

[9] C. Mohan, G. Alonso, R. Gunthor, and M. Kamath, "Exotica: A Research Perspective on Workflow Management Systems," *Bull. Technical Committee on Data Eng.,* vol. 18, pp. 19-26, 1995.

[10] IBM, "Flowmark: Managing Your Workflow,"Report SH-19-8176-01, 1994.

[11] H. Ludwig, "Termination Handling in Inter-Organizational Workflows-An Exception Managemen Approach," IBM Research Division, Zurich Research Laboratory, Rueschlikon RZ 3042 (#93088), Aug. 1998.

[12] A. Chan and K. Harty, "Building Flexible Applications with the Teknekron Enterprise Toolkit," *Bull. Technical Committee on Data Eng.,* vol. 18, pp. 33-40, 1995.

[13] M.P. Singh and M.N. Huhns, "Automating Workflows for Service Order Processing," *IEEE Expert,* pp. 19-23, 1994.

[14] P.K. Chrysanthis and K. Ramamritham, "Sysnthesis of Extended Transaction Models Using ACTA," *ACM Trans. Database Systems,* vol. 119, pp. 450-491, 1994.

[15] D. Barbara, S. Mehrotra, and M. Rusinkiewicz, "INCAs: Managing Dynamic Workflows in Distributed Environments," *J. Database Management,* vol. 7, pp. 5-15, 1996.

[16] G.D. Michelis, "Net Theory and Workflow Models," *Application and Theory of Petri Nets,* S. Donatelli and J. Kleijn, eds., pp. 282-283, 1999.

[17] H.A. Kuno and E.A. Rundensteiner, "Incremental Maintenance of Materialized Object-Oriented Views in Multiview: Strategies and Performance Evaluation," *IEEE Trans. Knowledge and Data Eng.,* vol. 10, no. 5, pp. 768-793, Sept./Oct. 1998.

[18] M. Papazoglou, A. Delis, A. Bougettaya, and M. Haghjoo, "Class Library Support for Workflow Environments and Applications," *IEEE Trans. Computers,* vol. 46, pp. 673-687, 1997.

[19] W. Wang and C. Zhong, "The Distributed Workflow Management System-FlowAgent," *J. Computer Science and Technology,* vol. 15, pp. 376-382, 2000.

[20] G. Dinkhoff, V. Gruhn, A. Saalmann, and M. Zielonka, "Business Process Modeling in the Workflow Management Environment: Leu," *Proc. 13th Int'l Conf. Entity Relationship Approach,* 1994.

[21] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Conceptual Modeling of Workflows," *Proc. 14th Int'l Conf. Object Oriented and Entity Relationship Approach,* 1995.

[22] A.-W. Scheer, *Architecture of Integrated Information Systems.* Berlin: Springer-Verlag, 1992.

[23] O. Zukunft and F. Rump, "From Business Process Modeling to Workflow Management: An Integrated Approach," *Business Process Modeling,* B.S. Reiter and E. Stickel, eds., 1996.
[24] L. Yu, "A Coordination Based Approach to Modelling Office Workflow," *Business Process Modeling,* B. Scholz-Reiter and E. Stickel, eds., 1996.
[25] M. Rohloff, "An Object Oriented Approach to Business Process Modeling," *Business Process Modeling,* B. Scholz-Reiter and E. Stickel, eds., 1996.
[26] M.A. Vouk, D.L. Bitzer, and R.L. Klevans, "Workflow and End-User Quality of Service Issues in Web-Based Education," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 4, p. 673-687, July/Aug. 1999.
[27] J.M. Nieves and A.P. Sage, "Human and Organizational Error as a Basis for Process Reengineering: With Applications to Systems Integration Planning and Marketing," *IEEE Trans. Systems, Man, and Cybernetics—Part A: Systems and Humans,* vol. 28, pp. 742-744, 1998.
[28] M. Klein and C. Dellarocas, "A Knowledge-Based Approach to Handling Exceptions in Workflow Systems," *Computer Supported Cooperative Work: The J. Collaborative Computing,* vol. 9, pp. 399-412, 2000.
[29] R. Agarwal, G. Bruno, and M. Torchiano, "An Operational Approach to the Design of Workflow Systems," *Information and Software Technology,* vol. 42, pp. 547-555, 2000.
[30] J.A. Miller, A.P. Sheth, and K.J. Kochut, "Perspectives in Modeling: Simulation, Database and Workflow," *Conceptual Modeling: Current Issues and Future Directions,* P.P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, eds., pp. 154-167, 1999.
[31] L. Shuzhou and A.G.E. Soong, " Modeling Workflows with Reactive Objects," *Int'l J. Flexible Automation and Integrated Manufacturing,* vol. 7, pp. 343-353, 1999.
[32] S. Jablonski, "On the Complementarity of Workflow Management and Business Prrocess Modeling," *SIGOIS Bul.,* vol. 16, pp. 33-38, 1995.
[33] P.P. Chen, "The Entity-Relationship Model: Towards a Unified Model of Data," *ACM Trans. Database Systems,* vol. 1, pp. 9-36, 1976.
[34] E.F. Codd, "A Relational Model for Large Shared Databanks," *Comm. ACM,* vol. 13, pp. 377-387, 1970.
[35] K.L. Siau, H.C. Chan, and K.P. Tan, "A CASE Tool for Conceptual Database Design," *Information and Software Technology,* vol. 34, pp. 779-786, 1992.
[36] W. Kozaczynski and L. Lilien, "An Extended Entity-Relationship Database Specification and Its Automatic Verification and Transformation into the Logical and Relational Design," *Proc. Sixth Int'l Conf. E-R Approach,* 1987.
[37] D. McGovern and C.J. Date, *A Guide to Sybase and SQL Server.* Addison Wesley, 1993.
[38] K.T. Owens, *Building Intelligent Databases with Oracle PL/SQL, Triggers and Stored Procedures.* New Jersey: Prentice Hall, 1996.
[39] J.Y.L. Thong, C.-S. Yap, and K.S. Raman, "Engagement of External Expertise in Information Systems Implementation," *J. Management Information Systems,* vol. 11, pp. 209-231, 1994.
[40] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database ReSearch: Achievements and Opportunities into the 21st Century," *SIGMOD Record,* vol. 25, pp. 52-63, 1996.
[41] E.K. Clemons and P.R. Kleindorfer, "An Economic Analysis of Interorganizational Information Technology," *Decision Support Systems,* vol. 8, pp. 431-446, 1992.
[42] H. Gregersen and C.S. Jensen, "Temporal Entity Relationship Models: A Survey," AalBorg University, Denmark TIMECENTER TR-3, Jan. 1997.
[43] T. deMarco, *Structured Analysis and System Specification.* Yourdon, Inc., 1978.
[44] J.M. Smith and D.C.P. Smith, "Database Abstractions: Aggregation and Generalization," *ACM Trans. Database Systems,* vol. 2, pp. 105-133, 1977.
[45] S. Ram and V. Storey, "Composites and Grouping: Extending the Realm of Semantic Modeling," *Proc. Hawaiian Int'l Conf. System Sciences,* 1993.
[46] T.J. Teorey, D. Yang, and J.P. Fry, "A Logical Design Methodology for Relational Databases Using the Extended ER Model," *Computing Surveys,* vol. 18, pp. 197-222, 1986.
[47] R.W. Floyd and R. Beigel, *The Language of Machines.* New York: Computer Science Press, 1994.
[48] A. Bajaj, "Managing Business Workflows Using a Database Approach: A Formal Model, A Case Study, and A Prototype," *MIS,* Univ. of Arizona, Tucson, 1997.
[49] H. Korth and A. Silberschatz, *Database Systems Concepts.* New York: McGraw Hill, 1991.
[50] A. Bajaj and S. Ram, "An Empirical Methodology to Evaluate the Completeness of Conceptual Business Process Models," *J. Information Technology Cases and Applications,* vol. 1, pp. 5-30, 1999.
[51] M.T. Laamanen, "The IDEF Standards: Methods and Associated Tools for the Information Systems Life Cycle," *Proc. Int'l Federation for Information Processing,* 1994.
[52] M.W. Alford, "SREM At The Age Of Eight: The Distributed Computing Design System," *Computer,* vol. 18, pp. 36-46, 1985.
[53] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Trans. Software Eng.,* vol. 3, 1977.
[54] D.T. Ross, "Applications and Extensions of SADT," *Computer,* vol. 18, pp. 25-35, 1985.

Akhilesh Bajaj received a BTech degree in chemical engineering from the Indian Institute of Technology, Bombay, an MBA degree from Cornell University, and a PhD degree in MIS (minor in computer science) from the University of Arizona. He is an assistant professor of Information Systems Management at the H. John Heinz III School of Public Policy and Management, at Carnegie Mellon University. Dr. Bajaj's research deals with the construction and testing of tools and methodologies that facilitate the construction of large organizational systems, as well as studying the decision models of the actual consumers of these information systems. He has published articles in several academic journals and conferences. He is on the editorial board of the *Journal of Database Management*. He teaches graduate courses on basic and advanced database systems, as well as enterprise wide systems.

Sudha Ram received the BS degree in chemistry (with physics and mathematics) from the University of Madras in 1979, the PGDM degree from the Indian Institute of Management, Calcutta in 1981 and the PhD degree from the University of Illinois at Urbana-Champaign in 1985. She is a professor of management information systems and the director of the Advanced Database Research Group at the University of Arizona. Dr. Ram's research deals with the modeling and analysis of database- and knowledge-based systems for business, manufacturing, and scientific applications. Her research has been funded by IBM, NCR, US ARMY, NIST, US National Science Foundation, NASA, ORD (CIA), and Raytheon. She has published articles in such journals as *Communications of the ACM*, *IEEE Expert*, *IEEE Transactions on Knowledge and Data Engineering*, *ACM Transactions on Information Systems*, *Information Systems*, and *Management Science*. Dr. Ram serves on the editorial boards for the *Journal of Database Management*, *Information Systems Frontiers*, *Journal of Information Technology and Management*, and as associate editor for the *Journal of Systems and Software* and *INFORMS Journal on Computing*. She teaches graduate courses on database design, E-business, and advanced information technology. She also serves as an information systems consultant to several national and international corporations. Dr. Ram recently received the Anderson Consulting Professor of the Year award in recognition of her contributions as professor, scholar, educator, and community leader. She is a member of the IEEE Computer Society.