Business Process Modeling Languages: Sorting Through the Alphabet Soup

Hafedh Mili[1,2], Guitta Bou Jaoude[1,2], Éric Lefebvre[1,3], Guy Tremblay[1,2], Alex Petrenko[1,4]
[1]Laboratoire de Recherche sur les Technologies du Commerce Électronique (www.latece.uqam.ca)
[2]Département d'informatique, Université du Québec à Montréal, Montréal, Canada
[3]École de Technologie Supérieure, Montréal, Canada
[4]Centre de Recherche Informatique de Montréal, Montréal, Canada

Abstract

Requirements capture is arguably the most important step of software engineering and yet the most difficult and the least formalized one [Phalp & Shepperd,2000]. Enterprises build information systems to support their *business processes*. Software engineering research has typically focused on the development process, starting with user requirements—if that—with *business modeling* often confused with *software system modeling* [Isoda, 2001]. Researchers and practitioners in management information systems have long recognized that understanding the business processes that an information system must support is key to eliciting the needs of its users (see e.g. [reference]), but lacked the tools to model such business processes or to relate such models to software requirements. Researchers and practitioners in business administration have long been interested in modeling the processes of organizations for the purposes of understanding, analyzing, and improving such processes [reference], but their models were often too coarse to be of use to software engineers. The advent of eCommerce and workflow management systems, among other things, have led to a convergence of interests and tools, within the broad IT community, for modeling and enabling business processes. In this paper, we present an overview of business process modeling languages. We first propose a categorization of the various languages. Then, we describe representative languages from each family.

1 Introduction

> *"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements . . . No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."* [Brooks, 1987]

Despite great advances in software engineering research, requirements capture remains the most difficult and least formalized development step [Phalp & Shepperd, 2000]. To understand what a software system is supposed to do, we need to put it into the context of the *business processes* that it is supposed to support. This support can range from storage and retrieval of business data to decision support or even to full automation. In all cases, an understanding of the underlying business processes is required. Researchers and practitioners in MIS have long recognized that understanding the business processes that an IS is supposed to support is key to eliciting the needs of its users. However, they lacked the conceptual tools to represent such processes, and to relate descriptions of such processes to requirements of the information systems that support them. Proponents of object-oriented modeling have argued that object models enable us to model the “real world” [Isoda, 2001] in a way that all stakeholders can understand. However, a number of experts agree that UML lacked the vocabulary to express business processes in a natural and intuitive way. UML’s built-in extension mechanism, however, has been used to define business process modeling concepts (*e.g.,* EDOC [OMG,2001]). Business process modeling constructs have also been added to UML 2 [OMG,2003].

Experts in business administration have long been interested in the business processes of organizations. Understanding such processes enables us to analyze them, to identify potential weaknesses or inefficiencies, and to “re-engineer them” to address those weaknesses [Hammer, 1990],[Hammer & Champy, 1993],[Ould,1995]. A number of process modeling languages have emerged, but the languages are typically too abstract, and the models too coarse, to support the elicitation of precise functional specifications for information systems.

Workflow systems were developed specifically to orchestrate business processes involving long interaction sequences (or *transactions*) [Jackson & Twaddle, 1997], [Dayal et al., 2001]. A number of workflow modeling languages have emerged, along with attempts to standardize them [WFMC,1999]. While workflow modeling languages are precise enough to be *executable*, the information systems that we are typically interested in end up as individual tasks within the workflow, and we are no closer to modeling the processes *embodied* by these systems.

Electronic commerce has reinforced interest in business process modeling for two reasons. First, enterprises that wish to partake in electronic commerce need to redesign their internal processes so that complex inter-enterprise transactions may be fully automated [Turban et al., 1999]. Second, they need to expose those parts of their internal processes that are needed to inter-operate with their peers. A number of languages for describing business processes and for business process coordination have been developed as a result, e.g., BPML [BPMI,2003], BPEL4WS[Andrews et al.,2003], ebXML [OASIS,2001]. Most of these languages use XML as a serialization format, but otherwise, have different foci—e.g., describing process semantics versus inter-process coordination—and may be more or less technology-dependent (e.g. BPEL4WS vs. ebXML).

In this paper, we provide a survey of the major business process modeling languages. We first present basic notions about business processes. Next, we propose a classification of business process description languages. In section 4, we present the traditional process modeling languages, coming mostly from the MIS tradition. Workflow and process integration languages are presented in section 5. In section 6, we discuss how object-oriented languages may be used for process modeling. We conclude in section 7 by a brief comparison of the various languages together with guidelines for selecting such a language.

2 Business process basics

2.1 What is a business process

The word "process" is defined in the dictionary as "a series of actions, changes, or functions bringing about a result". Bill Curtis defined a process as a partially ordered set of *tasks* or *steps* undertaken towards a specific goal [Curtis et al., 1992]. Hammer and Champy define *business processes* as a set of *activities* that, together, produce a result of value to the customer [Hammer & Champy, 1993]. The workflow management coalition defines business processes as "a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships." [WfMC,1999].

Different authors provide variations on the same set of themes. What emerges from these definitions is the following. The **activities** of a business process are performed by **actors** playing particular **roles**, consuming some **resources** and producing others**.** Activities may be triggered by **events** and may, in turn, generate events of their own. The **activities** of a process may be linked through **resource dependencies** (producer-consumer dependencies) or **control dependencies** (one activity triggering an other). The **actors** operate within the context of **organizational** boundaries. Organizations perform specific business **functions**. Roles can support **functions**. Figure 1 shows a first-cut UML-like model (metamodel) of what a process is.

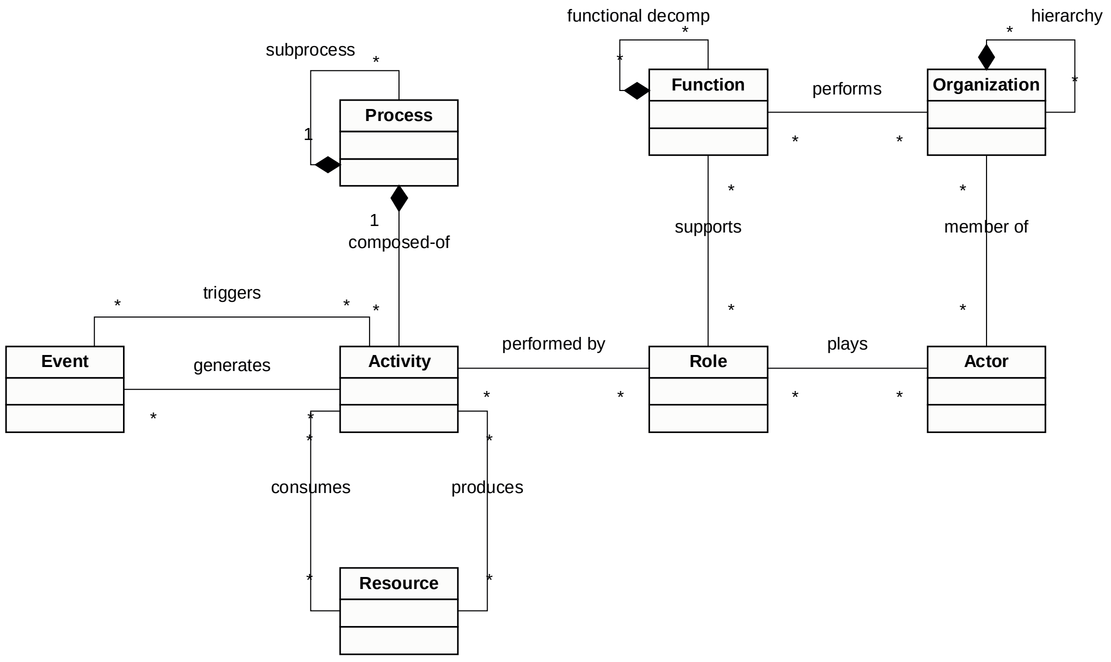


Figure 1. A first-cut business process metamodel.

Before we go on any further, we should make a distinction between *process definitions* and *process instances*. A process definition deals with *types* of business data or resources.

A process instance deals with *specific instances* of business data. For example, we could define what happens to a customer order in terms of triggering events (a call to customer service, or the submission of a web form), in terms of *types* of resources (we don't know which items the customer will order), and in terms of *roles* (*a* customer server representative). A process instance deals with a specific customer ("John Smith"), specific order contents, and a specific *actor* ("Beverly") playing the role *customer service representative*. The metamodel in Figure 1 is mostly about *process definitions*. Only the class **Actor** and its associations carry instance semantics.

2.2 Why business processes

In the traditional view, a business is considered as a hierarchical organization that reflects both the functional decomposition of the enterprise and the chain of command. Different *departments* specialize in specific business functions (e.g., marketing or production or accounting), and within each department, sub-departments, teams and individuals specialize in sub-functions. The processing of a customer order generally cross the boundaries of various departments: sales (to take the order), planning (to plan the manufacture of the product or the replenishment of the inventory), production, shipping, and accounting. Early management theory focused on the workings of the hierarchy and on managing its branches effectively (chain of command, workflow, accountability, communication, etc.), but focusing on each branch in isolation [Hans,2000]. With the so-called Business Process Re-engineering trend, a revolution took place: instead of focusing on each business function separately—and thus not *questioning* the overall structure of the underlying business processes—researchers started to advocate that one should look at the entire business process that is enacted to handle a business transaction from end to end, looking for ways to optimize the business process in its entirety [Hammer, 1990], [Hammer & Champy, 1993]. This meant that different business processes did not necessarily cross the same organizational boundaries, or did not cross them in the same way. From the practical side, this trend led to exotic organizational structures such as *project-oriented management* whereby project teams are assembled from different functional units to handle all aspects of a business transaction, or *matrix organizations* where *business units* (vertical) offer services to outside customers by relying on internal *service units* (horizontal).

Business process re-engineering has renewed interest in business process modeling as a pre-requisite for process analysis and improvement. A sample of research on business process analysis include work done at MIT to develop a business process repository [Malone et al., 1999]. Other research deals with assessing quality properties of business processes using structural metrics similar to those used in software [Phalp, 1998], [Phalp & Shepperd, 2000]. A number of researchers have tried to verify the dynamic properties of processes such as liveness and absence of deadlock using formal methods [Glykas & Valiris,1999], [Gruhn & Wellen,2001]. The advent of electronic commerce has further amplified interest in process modeling languages.

2.3 An ontology of enterprises

Businesses have a wide variety of processes going on concurrently. The board of directors has it own predefined processes for decision making, for nominating officers, for designating members of the board. The executive will have its own decision processes and distribution of responsibilities. Each functional department (accounting, marketing, production, customer service, etc.) or business division (e.g., a bank might have separate divisions for personal financial services, corporate financial services, fund management, etc.) will have its own processes, etc. Each level of the organization, down to making security rounds or mail delivery, will have its own objectives, its own processes, its own performance measurements to measure the extent to which a given process helps attain the desired objectives. When we talk about "business process modeling", we must identify which processes we are interested in, at what level of detail, and what are the relationships between these processes, if any.

Assume that a company aims at increasing its market share for its products. There are several ways to achieve this goal, including product innovation, competitive pricing, targeted marketing, building customer loyalty, responsive customer service, etc. Assume that the company chooses to focus on competitive pricing and targeted marketing. This would be *the process* followed by strategists at the enterprise level to achieve the goal of increased market share. Competitive pricing entails cost reduction, including engineering costs, production costs, distribution costs and marketing costs. The organization may use a product development strategy based on the concept of product lines to share the development of costs between several products. Product-line engineering itself uses a specific engineering process, with its own sequence of steps, sub-steps, resources, etc. Come the end of the year, suppose we compare the current market share of the company with that of the previous year … and it has actually gone down. The attainment of the initial goal has triggered a number of processes at different levels of the organization, and any one of them, or combination thereof, might be at fault. It could be the focus on competitive pricing and targeted marketing: perhaps the product is labor intensive with no room for automation and the company operates in a labor market with high labor costs. It could also be that the product-line engineering approach was not appropriate, or that the specific product-line engineering *process* was not effective, etc.

The previous example shows that in order to gain a thorough understanding of how an organization works, we need to look at the processes at different levels of the organization, and how they relate to each other. In the above example, we were not concerned with the process used by security guards. Clearly, some processes will have little bearing on the analysis at hand. Ould distinguishes between *core processes*, *support processes*, and *management processes* [Ould, 1995]. The *core processes* are concerned with addressing external requests from the enterprise—its customers. The *support processes* support its employees internally in executing the core processes. *Management processes* manage both the core processes and the support processes. Very broadly, we analyze core processes to enhance customer satisfaction. We analyze support processes to enhance the enterprise efficiency. We analyze management processes to enhance the enterprise structure.

Gale and Eldred view an enterprise as a *hierarchical multi-layer system*, thus combining the characteristics of *hierarchical systems* and *multi-layer systems* [Gale & Eldred,1996]. A *hierarchical system* is an aggregate system where each node in the aggregation hierarchy controls and coordinates its immediate components. Figure 2-a illustrates an example of a hierarchical system. Control and coordination happens through two-way communication between an aggregate and its components: i) the components "send" information about their performance to the aggregate, and ii) the aggregate "sends" control information back to the components. An example of a hierarchical system is the lower-levels of organization of work in a factory: a foreman controls the work of individual workers. An assembly-line manager controls the work of teams. An operations manager controls the work of various assembly lines. In a hierarchical system, the work of an aggregate is *defined* as the aggregation of the work of its components. In a *multi-layer system*, work happens at every layer of the system, which are self-contained from a definitional point of view, but each layer provides *services* to the layer above it. The best illustration is the OSI protocol hierarchy for telecommunication networks: each layer of the OSI hierarchy has its own protocol, but within each layer, some functions of the protocol are *implemented* using services from the layers below it. Another example is a computer, which can be seen as a stack of "machines", with the physical machine at the bottom and a high-level language at the top, and various "virtual machines" in-between. Within a hierarchical system, a level of the hierarchy *controls* the levels below it. Within a multi-layer system, a layer *implements* some of the functionalities of the layer above it, but the layers use different "vocabularies".

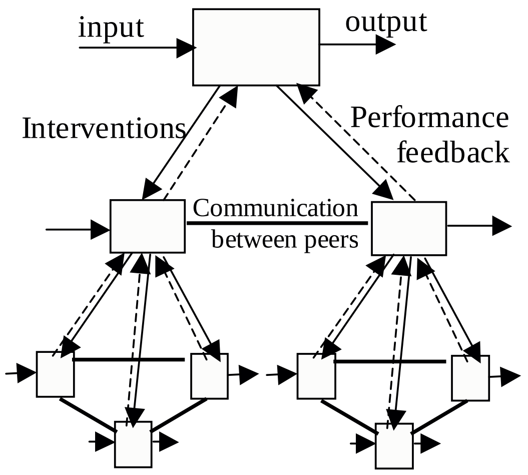


Figure 2-a. A *hierarchical system*

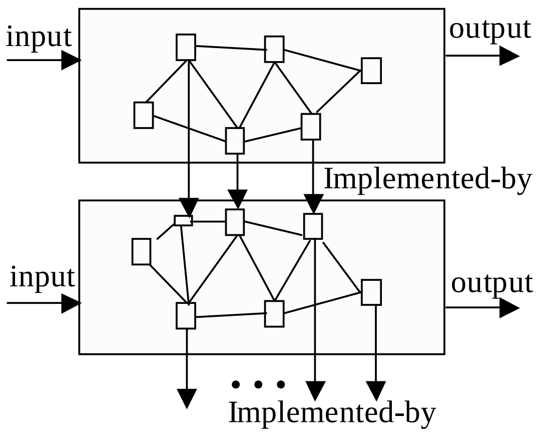


Figure 2-b. A *multi-layer system*

According to Gale and Eldred, organizations combine the characteristics of both *hierarchical systems* and *multi-layer systems*, where the layers often correspond to levels of the hierarchy [Gale & Eldred,1996]. Within an organization, people at the executive level set the strategy. People at the managerial level plan a course of action to realize the strategy. People at the operations level execute the strategy. Incidentally, managers also control operations staff, and executives control managers, but the skills required for strategizing, managing or producing are different.

From a business process modeling point of view, it appears that we can document a process at various levels of the organization, and we often should. As our previous example showed, things can go wrong because of various reasons (poor strategy, poor planning, poor execution), and to fully diagnose problems and fix them, we need to understand processes at different levels. Clearly, the domain vocabularies for the processes will be different, depending on the layer of the organization at which the processes take place. At the operations or factory floor level, we may talk about individual workers, machine tools, conveyor belts, and bins. Higher up, we may *still* talk about workers, assembly lines, temporary storage areas, and warehouses. At the highest levels of the enterprise, we may talk about *yet* more people (decision makers, upper managers), business units, functional units, regions, etc. Process modeling languages have to be able accommodate these various levels of discourse about the enterprise.

3 A classification of business process modeling languages

3.1 Kinds of modeling languages

As the history of process modeling languages showed (sections 1 and 2), existing business process modeling languages come from different scientific traditions and, as such, serve different purposes and represent different things [Curtis et al., 1992]. Ould argues that business process modeling is useful for three basic reasons, which may in turn support several business goals [Ould,1995]:

1) *Describing a process*: we model a process to be able to describe it. We could have different target audiences for these descriptions, for instance, humans, in which case understandability is important [Curtis et al., 1992], or machines, in which case formality is important.
2) *Analyzing a process*: simply put, process analysis consists of assessing the *properties* of a process. Process re-engineering and improvement relies on an analysis of existing processes to identify redundant or sub-optimal steps. If the process is described *formally*, we can *verify mechanically* structural properties such as coupling and cohesion [Phalp & Shepperd, 2000] or dynamic properties such as the absence of deadlock, liveness properties, etc.
3) *Enacting a process*: we may enact a process for simulation purposes or to provide some level of support for process execution. Depending on the language, this support can take different forms : reacting to events triggered by the execution of the process, t checking that specific constraints are satisfied, driving the execution of the process [Curtis et al., 1992]. Only formal languages[1] make process enactment possible.

Language designers may put the emphasis on one of these basic usages, often at the expense of others.

[1] A language is considered to be formal if both its syntax and semantics can be precisely defined. When the semantics if formally defined, sentences in the language then have a unique interpretation.

Because business processes are complex, language designers generally provide different modeling *views*, each focusing on one aspect of the process. Curtis identified four views, summarized below [Curtis et al., 1992]:

1) *The functional view*: this view presents the functional dependencies between the process elements (activities, sub-processes, etc.). These dependencies are typically embodied in the fact that some process elements consume (or need) data (or resources) produced by others. Typical notations used in the functional view include *data flow diagrams*.
2) The *dynamic (behavioral) view*: the dynamic view provides sequencing and control information about the process, that is, when certain activities are performed (timing, pre-conditions) and how they are performed (e.g., by describing the control logic).
3) *The informational view* includes the description of the entities that are produced, consumed or otherwise manipulated by the process. These entities include pure data, artifacts, products, etc.
4) *The organizational view* describes *who* performs each task or function, and *where* in the organization (functionally and physically).

As is common with modeling methods, different notations may be appropriate for different views.

Most object-oriented modeling methods include the first three views mentioned by Curtis et al., namely, the functional view, the dynamic view, and the informational view (e.g., OOA/OOD (OMT [Rumbaugh et al., 1991], OOSE [Jacobson et al.,1992], Fusion [Coleman et al., 1994], Shlaer & Mellor [Shlaer & Mellor, 1993], UML [Booch et al., 1999]). Some methods may amalgamate the functional and dynamic view[2], but in the end, the important concepts from these views are properly represented. What is new, from an ontological point of view, is the *organizational view*, which includes a description of the *participants* in the process as well as a description of the physical (location) and organizational context within which this process is conducted. Furthermore, whereas the informational view in object-oriented modeling represents only data entities, the informational view of business processes modeling may represent tangible resources and artifacts that are used and produced by processes. We will elaborate more on this distinction in section 3.2.

Another interesting distinction is the one, introduced more specifically in the context of Web services, between the *orchestration* view and the *choreography* view [Peltz, 2003]. Whereas orchestration refers to a specific process—i.e., how this process is to be performed, solely from that process perspective—, choreography refers to the exchange of messages between the various parties and sources—i.e., orchestration tracks the message sequences among the parties involved in the various transactions. While some languages can express only one of those views, generally the orchestration one, others (e.g., BPEL4WS) can express both.

[2] OMT makes the distinction very clear [Rumbaugh et al., 1991], although it has been criticized for the difficulty with which the functional model could be integrated into the object (informational) model.

3.2 Kinds of modeling

Modeling may be seen as a mapping between two *worlds*, the *modeled world* and the *modeling world*. A modeling technique is defined by the way different things, concepts or constructs in the *modeled* world are mapped to things, concepts or constructs in the *modeling* world. Typically, the things in the modeled world map to simpler things in the modeling world. This mapping is done in order to perform some operations on objects in the *modeling world* that would not be possible on objects of the modeled world—too costly, difficult to perform, etc. Different modeling techniques embody different *perspectives* as they abstract different details from the modeling world, leaving only those aspects that are relevant for the tasks at hand. Most of us in the software field are familiar with modeling techniques, and more specifically, object-oriented modeling techniques. Is *business process modeling* any different from the kind of modeling we do with OMT or Fusion or RUP, and if so, in what ways?

Isoda distinguishes between two kinds of object-oriented modeling, what he called *real-world modeling*, and *pseudo –real-world modeling* [Isoda, 2001]. He identifies three possible uses of object-oriented modeling — 1) to gain an understanding of how the real world functions, 2) to run a *simulation* of the real world, and 3) to write an application to automate parts of a business process — and shows that they call for different kinds of modeling [Isoda,2001]. He argues that the first two types of use call for *real-world modeling,* requiring a faithful representation of the *modeled world*, in this case, the *real world*. In *real-world modeling*, entities of the real world and their properties are mapped to classes and attributes. The functions of the real world entities are mapped to operations on those classes, and static relationships between the real world entities are mapped to associations.

The third use of modeling, i.e., its use to develop software that automates parts of the business process, requires both real-world modeling and pseudo real-world modeling. We need *real-world modeling* to model the behavior of the business as it would exist *with the automated system in place*, in order to elicit the functionalities of the system. In other words, we imagine what the process would be with the automated system to better delineate the required functionalities [Isoda,2001]. "Real-world" models of the business would show the automated system as a black box[3] that interacts with the other actors of the process. We also need pseudo-real world modeling to model the software artifacts that are inside the black box. With pseudo real-world modeling, entities of the real world are modeled by entities that *represent the information about the real world that an automated system would manipulate*. Generally speaking, pseudo-real world modeling maps those entities of the real world whose information is manipulated by the software application to *classes*. The pieces of information required by the software then become attributes. Methods are assigned to classes based on software packaging issues—the *encapsulation principle* according to Isoda—rather than on functional responsibilities in the real world, as is the case for real-world modeling.

[3] Shows up as "**System**" is use cases or sequence diagrams.

The following example, based on [Isoda,2001], illustrates the difference between real-world modeling and pseudo real-world modeling. Assume we want to build a library automation system. The real world includes books, users, user files, librarians, subject areas, physical indices, and physical locations (sections, racks, shelves). If we were to use a computerized library system, the "automated real world" would include users, books, librarians, physical locations, and would include "the system", which manages indices, user files, and which tracks books. Figure 2-a shows the static real-world model of the functioning of the library *with* a computerized system. Figure 2-b shows a pseudo real-world model of the computerized system itself.

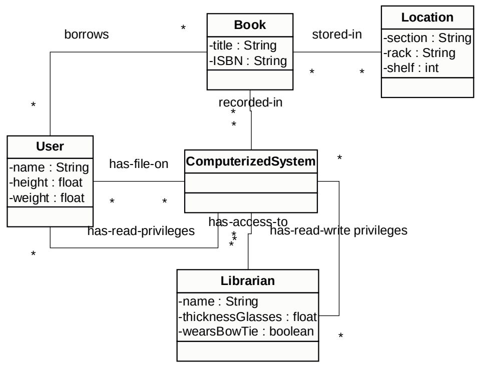


Figure 2-a. Real world model of library with computerized system

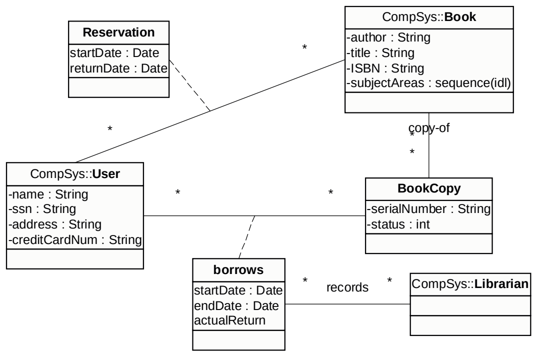


Figure 2-b. Pseudo-real world model of the computerized system

While some of the attributes of the classes in Figure 2-a may appear *facetious*, we wanted to make clear the distinction between the two kinds of modeling. With real-world modeling, we model the actual real world entity (height, weight). With pseudo real-world modeling, we model the information that a computerized system needs to manipulate. Obviously, the class "User" appearing in the model of Figure 2-a is different from the one appearing in Figure 2-b: the first represents the actual user while the second represents the user's file. The same is true for the classes "Librarian" and "Book". Isoda describes a number of modeling errors that result from mixing real-world modeling with pseudo real-world modeling [Isoda,2001].

From the above, it appears that **business process modeling** requires ***real-world modeling* of the business** and its processes, while **software modeling** requires **pseudo real-world modeling**. Eliciting **user requirements** for a software application requires **real-world modeling** of **the business *with* the automated system**. Business process modeling does not require the presence of automated components. In the case of the library, we could describe the borrowing process for an entirely manual library system, from searching index cards, to browsing through the shelves to find the book, to recording the book loan with a librarian. The interesting thing here is that the *internal process* of the computerized library system mimics rather closely the manual process: instead of paper files and records and manual look-up, we have computerized records, and computer-based lookup. This will often *not* be the case: the process implemented by the computer component of the business process will often be different from the corresponding manual process[4]. In other words, the model of the process *without* the computerized component—i.e. the manual process—will *not* simply consist of the merging of the model of the business *with* the computerized component together with the model of the computerized component. This idea was the premise behind the BPR movement: instead of blindly automating manual processes, we should re-engineer the processes while taking advantages of the possibilities for automation [Hammer, 1990].

3.3 An overview of business process modeling languages

In the remainder of this paper, we will be studying a number of languages that were developed with different objectives in mind, but that have all been used to describe business processes. The languages that we will study address different facets of business processes (dynamic, functional, informational, organizational), and may be more or less formal, depending on the intended use and audience. There is no easy way of categorizing these languages along a single dimension, as they cut across several of the dimensions discussed earlier. However, the languages do fall under four broad but distinct scientific traditions:

1) *Traditional process modeling languages*: these languages mostly come from the MIS tradition of information engineering and from work on business process engineering. With one notable exception (Petri Nets), they share concerns for understandability by people. These languages are typically not formal, but may lend themselves to various informal or heuristic analyses. Languages in this category include IDEF, Petri Nets, Event Process Chains (EPC) [], Role Activity Diagrams [Ould, 1995], Resource-Event-Agent (REA) [], and the recently minted *Business Process Modeling Language* [BPMI,2003].
2) *Workflow modeling languages*: roughly speaking, a workflow management system is a computer system that manages a business process by assigning activities of the process to the right resources, by "moving" work items (e.g., documents, orders, etc.) from one processing step to the next, and by tracking the progress of the process [WfMC,2002]. Roughly speaking, workflow modeling languages are scripting languages for describing workflows so that they may be supported by a workflow management system. These languages are, for the most part, formal and executable. We will talk about the Workflow Process Description Language (WPDL) [WfMC,1999] and proposed interchange formats such as PIF [Lee et al., 1996] and PSL [NIST,2002].
3) *Process integration languages*: the advent of inter-enterprise electronic business (B2B) has spurred interest in process modeling languages for the purposes of integrating the processes of two or more business partners. Such languages typically focus on the mechanics of the integration in terms of abstract, technology independent, programming interfaces and data exchange formats. Languages in this category may also capture different levels of the *semantics* of the underlying processes. Three such languages will be presented: RosettaNet

[4] The two processes may be similar in the library system because libraries have been around for centuries, and we were able, as a civilization, to optimize the underlying processes.
[5] Petri Nets, which admittedly, had a separate lineage.

[RosettaNet,2003], ebXML [ebXML,2003], and BPEL4WS [Andrews et al., 2003].

4) *Object-oriented languages*: despite its programming ancestry, object-oriented modeling has been vaunted from the beginning as a *natural* way of representing the world in a way that both domain experts and IT experts can relate to (e.g., see [Coad & Yourdon, 1989], [Rumbaugh et al., 1991]). The question we raised in section 3.2 above was : "Which world"? After the "naiveté" of the early years [Isoda, 2001], the boundary between the *problem domain* (modeling the *business*) and the solution domain (modeling the *software*) has become well enough defined for us to realize that object-oriented modeling languages are, for the most part, geared more towards representing the solution (software) domain rather than the problem (business) domain, either because of inherent shortcomings or because of focus. In the section on OO languages, we will talk about UML 1.x, its extension mechanisms, as well as various extensions that were proposed in the literature to handle enterprise modeling, including EDOC [OMG,2001]. UML2 has also incorporated a number of these extensions in its metamodel, and we will conclude that section by presenting the new metamodel-level constructs introduced by UML2 [OMG,2003].

4 Traditional process modeling languages

4.1 The IDEF family

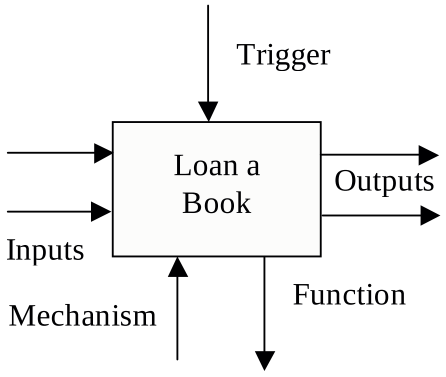
IDEF is family of methods for enterprise modeling and analysis sponsored by the US Air Force within the context of its long-running *Integrated Computer-Aided Manufacturing* (ICAM) program. The program, launched in the mid-seventies, sought to increase manufacturing productivity through the systematic application of computer technology. The program recognized (manufacturing) process analysis as an important tool, and identified the need for better communication techniques to describe such processes. A family of modeling methods was introduced, referred to collectively as *IDEF* (*I*CAM *Def*inition). Initially, three methods were planned, IDEF0, for functional modeling, IDEF1 for information modeling, and IDEF2, for dynamic modeling. The methods have since been updated and maintained under the stewardship of *Knowledge Based Systems Inc*. under the sponsorship of the US Commerce Department. New methods have joined the family, IDEF4, which is an object-oriented modeling methodology, and IDEF5, which is a methodology for developing *ontologies*. In this section, we look at IDEF0 (functional modeling), IDEF1X (an extension of IDEF1), and IDEF3 (dynamic modeling), which supercedes IDEF2.

4.1.1 IDEF0: functional modeling

IDEF0 was based on the *Structured Analysis and Design Technique* (SADT) [Ross, 1977], a *software* analysis and design technique developed in the late seventies. It includes both a definition of a graphical notation, as well as a comprehensive methodology for developing functional models.

An IDEF0 model describes what a system does—its *function*—what controls it, what are its inputs, its outputs, and which services or other functions it needs to perform its function. An IDEF0 *diagram* consists of a graph where nodes, represented by boxes, represent *functions*, and where *directed* edges represent *data flows* and *control flows*. Figure 4-a shows what the function looks like. Figure 4-b shows a sample diagram.

An IDEF0 model is a hierarchy of diagrams where a “box” at a given level may be expanded into a graph (another diagram) at the lower level, enabling us to represent



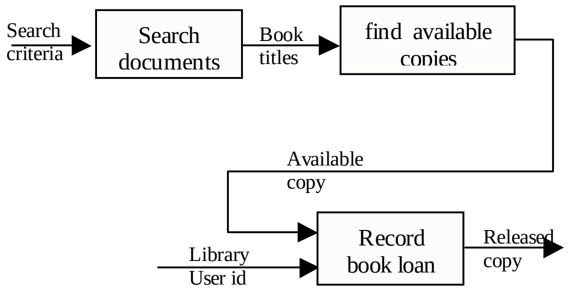


Figure 4-a. Anatomy of a function in IDEF0   Figure 4-b. A functional diagram in IDEF0

functions at increasing levels of detail. Data flows between functions lead to implicit control: a function can only start when all of its inputs are available. Data flows also have richer semantics than is shown in Figure 4-b (forks, joins, aggregation/decomposition, etc.).

Clearly, IDEF0 covers only the *functional* view of a business process. Furthermore, it is not as rich as the data flow diagram notation used in the twin *structured analysis/structured design* techniques (e.g., [Yourdon, 1989]) which includes things such as data stores, actors, sinks, etc.

4.1.2 IDEF1/IDEF1X: informational modeling

IDEF1X is a technique for representing *semantic data models* of the enterprise. The initial modeling technique (IDEF1, 1981) was based on the relational data model and the entity-relationship (E/R) approach. In addition to the modeling notation, IDEF1 proposed guidelines and procedures for eliciting such data models. IDEF1 was later extended (1985) to incorporate notions of *semantic data models*, including much richer semantics for relations, and new concepts such as aggregation and *categorization*—organizing entities in generalization/specialization hierarchies.

IDEF1X supports the notion of *views*, but in a way slightly different from that used in traditional databases. Roughly speaking, a view is a partial, possibly restricted model of the data. The model is partial in the sense that not all entities are represented, and for a given entity, only a subset of the attributes may be included in the view. The model is also possibly restricted because the domain for an attribute within a given view may also be restricted.

In both intent and form, IDEF1X is geared towards representing data that will ultimately be manipulated by a computer program. The specification defines entities as "things ***about which data is kept***". Things about which no data is kept are not represented. Like we showed in section 3.2, this is a *subset* of the real world that we would normally want to model in an enterprise.

4.1.3 IDEF3: the behavioral view

IDEF3 was developed specifically to describe the dynamic aspect of business processes [Mayer et al., 1995]. One of the motivations of IDEF3 was to facilitate the system requirements definition/elicitation of computer systems. IDEF3 was designed to support descriptions of the following aspects: 1) scenarios of organizational activities, 2) *roles* of user types in the organizational activities, 3) use cases (UML's), 4) user classes, 5) timing, sequencing, and resource constraints, and 6) user interface objects. Referring to our typology of modeling approaches (see section 3.2), items 1), 2) and 3) above correspond to *real-world modeling of the business with the automated system*. Aspects 4) and 6) correspond to software modeling (*pseudo real-world modeling*) whereas aspect 5) involves a bit of both.

IDEF3 models are, to use today's terminology, scenario-driven. A scenario describes a typical occurrence of a business process. Scenarios are described using two complementary models: *process-centered* models are variations on UML's activity diagrams[6], which focus on sequencing or *workflow*, and *object-centered* models, which focus on state changes for objects throughout the process. There are several flavors of object-centered models:

- *Transition schematics*, which describe a single object state change through a single scenario,
- *Extended transition schematics*, which describe state changes for *several* objects through a scenario, and
- *Object schematics*, which describe state change information across *several scenarios*.

Figure 5 shows an example of a simple process for purchase ordering. It shows two alternative paths of processing depending on whether there is a known supplier for the part or not. Each activity, called *Unit of Behavior* (UOB), is represented by a numbered box. The join box (the X) denotes an exclusive or, and has similar semantics at the fork and join points: only one path is followed leading out of "Request material", or leading into "Order material".

[6] Although work on IDEF3 preceded the inception of UML, we will keep comparing its diagrams to the more recent UML notations, simply because more people are familiar with UML.

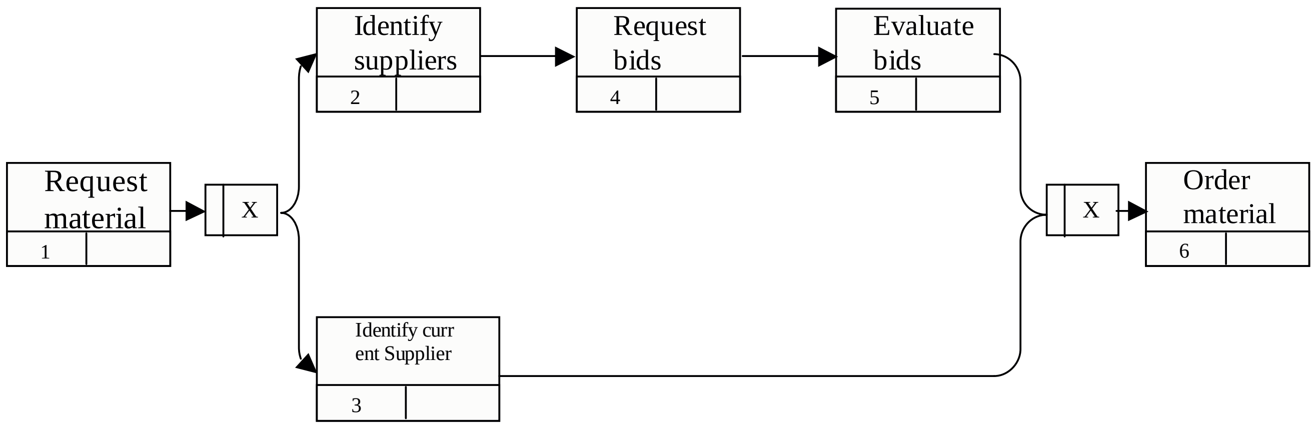


Figure 5. A process diagram in IDEF3. Adapted from [Mayer et al., 1995].

Figure 6 shows excerpts of the *transition schematic* for the purchase order object for the "Order material" process represented in Figure 5. Note that all the states are related to the same object (Purchase Request, PR), and that transitions are labeled by the activities (UOBs) that cause them.

The full syntax of IDEF3 includes rich semantics for process junctions (synchronous and asynchronous forks and joins), and richer semantics for links than what we have shown. The same holds true for object-centered diagrams, with rich semantics for states, transitions as well as for the properties that can be associated with a transition. In *extended transition schematics*, the same diagram shows state transitions of different objects, along with relations between those objects. The relations may include part-of relations, arbitrary associations, as well as typing relations between objects. We find this addition slightly confusing as the *informational view* seems to be creeping into the dynamic view. Finally, the notation has a language for *elaborations*, which are semantic annotations written in a logical language that describe static or dynamic constraints that must hold during the process.

In summary, IDEF3 includes two important aspects of behavioral modeling:, process execution and state changes of the entities during the execution of the process. One of its interesting features is the *elaboration language*, which can express logical constraints about the processes. The mixing of informational aspects and dynamic aspects may lead to confusion, and does not appear a helpful feature. What is missing from the family of—supposedly—complementary methods are notions related to *actors*, *roles*, and *organizations*.

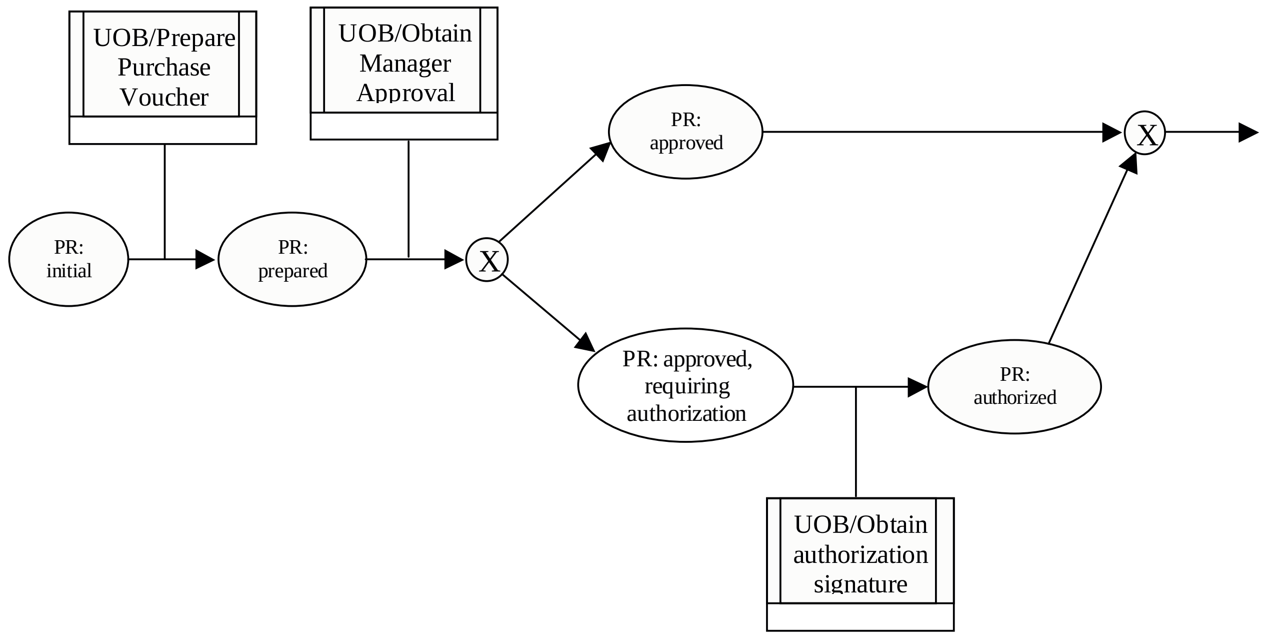

UOB/Prepare Purchase Voucher
UOB/Obtain Manager Approval
PR: approved
X
PR: initial
PR: prepared
X
PR: approved, requiring authorization
PR: authorized
UOB/Obtain authorization signature

Figure 6. Transition schematic showing the states undergone by a purchase request

4.2 Petri nets

A Petri net is a special kind of graph aimed at representing the behavior of dynamic systems [Murata, 1988]. Petri nets have been developed by Carl Adam Petri, in the context of his doctoral dissertation, in 1962. They have been used to model the behavior of systems that are concurrent, asynchronous, distributed, parallel, non-deterministic, and even stochastic. The power of Petri nets resides in the fact that a small number of constructs (tokens, places, and transitions) can represent fairly complex behaviors, in a fairly concise fashion. Furthermore, models expressed using Petri nets lend themselves to formal analysis and validation [Murata, 1988].

Simply put, a Petri net is a special kind of graph with two kinds of nodes, *places* and *transitions*, and directed weighted arcs between them. The places that are upstream from a transition are called its *input places*. The places that are downstream are its *output places*. A transition is said to be *enabled* if each input place contains at least as many tokens as the weight of the arc linking it to the transition. Figure 7-a shows a transition that is *not* enabled; Figure 7-b shows a transition that *is* enabled.

A transition that is *enabled* may *fire*, in which case it sends tokens along its outbound arcs to the output places. The number of tokens sent by a firing transition to an output place equals the weight of the arc leading from the transition to the output place. Figure 7-c shows the state of the Petri net after the firing of the transition.

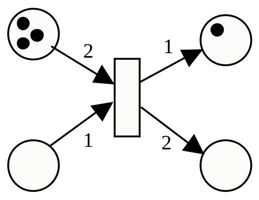


Figure 7-a. A petri net. The transition isn't triggered because the bottom left place does not have a token.

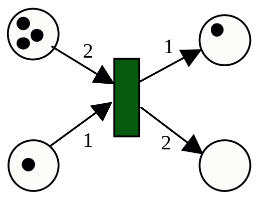


Figure 7-b. The transition is now triggered with a token moving in the bottom left place

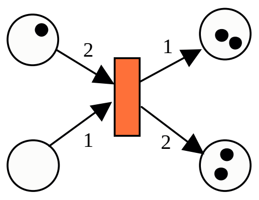


Figure 7-c. After firing, each input (output) place loses (gains) an equal number of tokens to the weight of arc.

Petri nets have been used in many application areas. The following table (from [Murata, 1988]) shows the many possible interpretations of places, tokens, and transitions.

Input places	**Transition**	**Output place**
Preconditions	Event	Post-conditions
Input data	Computation step	Output data
Input signals	Signal processor	Output signals
Resources needed	Task	Resources released
Conditions	Clause in logic	Conclusions
Buffers	Processor	Buffers

Throughout the years, different variations on the simple Petri net models were developed to account for the complexity of the systems being modeled. One basic variation assumes *finite capacity places*, adding a condition for firing transitions: output places should not overflow. It can be shown that a finite capacity Petri can be transformed into an equivalent infinite capacity net [Murata, 1988]. The introduction of pure control arcs (*inhibitor arcs*) increases the expressive power of Petri nets to that of Turing machines [Peterson, 1981].

Petri nets lend themselves to formal analyses. One can formally validate a number of properties, including the followings:

1) *Reachability*: The state of a Petri net is described by the contents of the different places (*marking*). A marking M_n is reachable from an initial marking M_0 if there exist a number of transition firings that can lead to M_n from M_0.
2) *Boundedness*: A Petri net is bounded if, for a given initial marking, the number of tokens in any given place never exceeds a finite number k.

3) *Liveness*: A Petri net with some initial marking is live if every transition is firable/reachable from the initial marking.
4) *Reversibility and home state*: A Petri net is reversible if for each marking M that is reachable from some initial marking M_0, there exists a finite number of transitions that would take the net from M back to M_0 (or some other state, referred to as the *home state*).

The fact that these properties are theoretically verifiable does not mean that they are computationally tractable. A number of transformations need to take place before (e.g., see [Murata, 1988]). Current research with Petri nets deals with new application areas as well as reduction techniques that help make Petri nets computationally tractable.

Referring back to our classification of section 3.1, Petri nets are clearly geared towards the modeling of the behavioral view. All of the other perspectives are literally missing, although researchers have developed extensions with richer "token models" to account for structured data. Additionally, Petri nets are clearly not meant for communicating models to business people, although many researchers have valued their analyzability, which makes it possible to validate properties about the processes they model (e.g., see [Phalp, 1998],[Glikas & Valiris, 1999]). In summary, the Petri net formalism can be seen as a low-level language, almost a kind of assembly language level modeling language. In fact, this is how Petri nets have been used in some formal specification language toolboxes, e.g., in the CADP toolbox [Fernandez et al., 1996], where process algebraic specification written in the LOTOS language [Bolognesi & Brinksam, 1987] are compiled into Petri nets in order to be analyzed.

4.3 Role Activity Diagrams (RAD)

The Role Activity Diagram is part of STRIM™ (Systematic Technique for Role and Interaction Modeling), a methodology developed by Praxis Plc. of Bath (UK) for the "elicitation, modeling, and analysis of business processes" [Ould, 1995]. STRIM was developed out of research aimed primarily at *software process* modeling by Ould and Roberts [Ould & Roberts, 1987]. The focus of STRIM—and RAD— is on developing models that are "revealing and communicative" [Ould,1995]. Ould identified five concepts that he considered essential to STRIM: *roles*, *actors*, *activities*, *interactions*, and *entities*. Roles are *types*, such as *project manager role*. Within a given organization, different instances of a given business process may be ongoing, and several *instances* of the *role project manager* may be active at the same time. *Actors* are individuals or systems that *play* a particular *role* (are bound to a role instance) at a given point in time. *Activities* are what *actors do* in their *roles*. In the process of performing their activities, *actors* may *interact* to exchange information or to control each other's execution. *Interactions* may be binary or multi-party. Interactions are *synchronous* in the sense that the actors rendez-vous to interact, e.g., to exchange data. The data that actors exchange through interactions is represented by *entities*. Ould stressed that data is not his primary focus, but process is, and not surprisingly, the representation of entities is somewhat poor.

With these five ingredients, processes are represented using *role activity diagrams*. The following diagram illustrates the notation. The boxes with rounded corners represent roles. There is an implicit timeline going down from the top, and the roles are laid horizontally, side by side, much like UML's swim lane activity diagrams—see also section 7.1. White boxes represent end points of interactions. Black boxes represent activities. White boxes with diagonal lines indicate the instantiation of another role. Upward pointing empty arrows represent concurrency. Downward pointing empty arrows indicate branches of conditionals. External events are represented by horizontal arrows. In the example, the *Divisional Director* role receives the “New Project Approved” event. States are represented by circles with a handle around the time line.

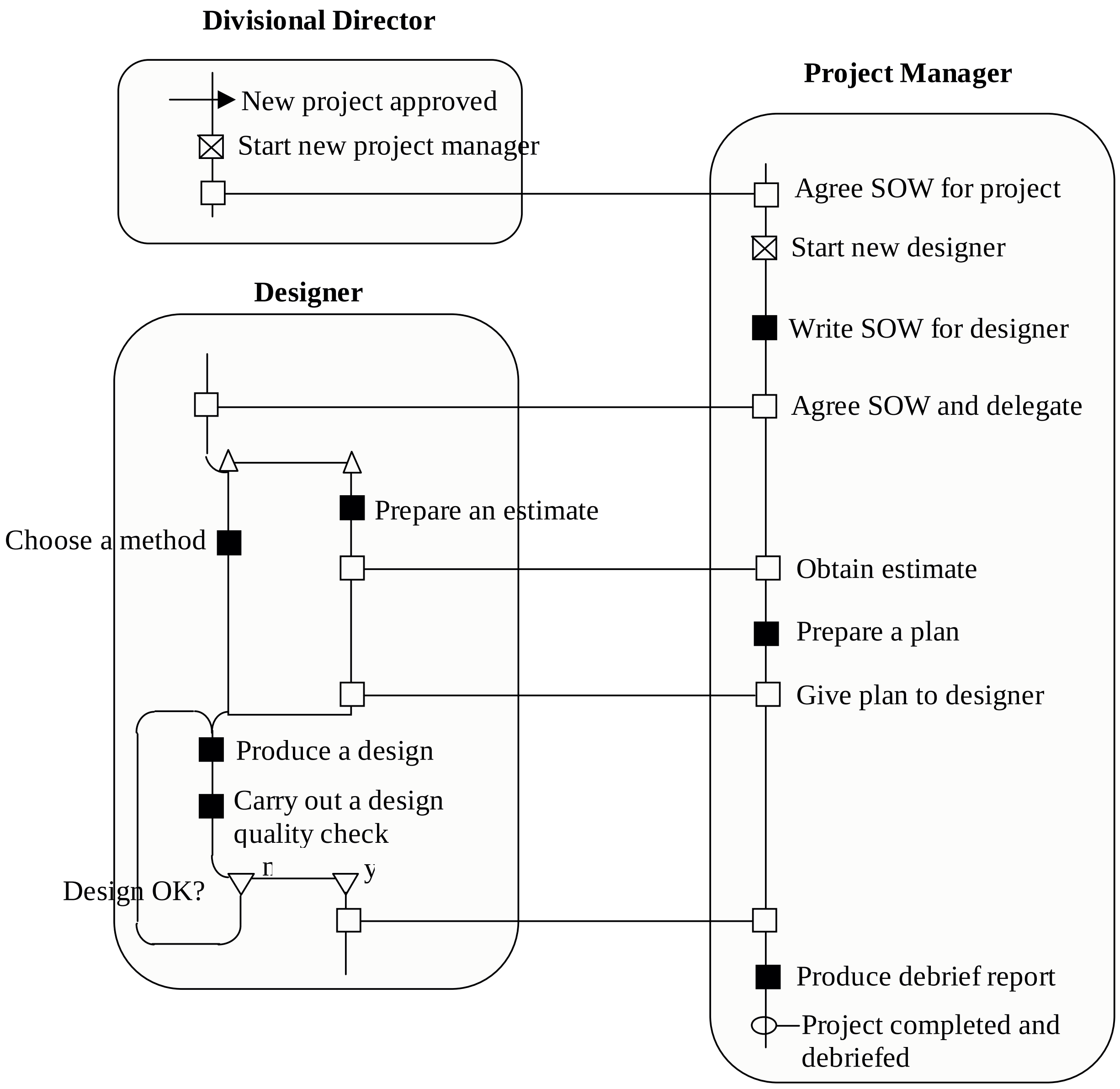


Figure 8. Role activity diagram for carrying out a project

In the process shown in Figure 8, the divisional director instantiates the project manager role, who, in turn, instantiates the designer role. Note that each role instantiates another role and then interacts with it. Also, the project manager role defines the tasks that the designer role will perform (*statement of work*, or SOW). Finally, note the final state of the process, embodied in the state “Project completed and debriefed”. In STRIM, each process has a *goal*, and these goals are often represented as desired states for actors within the organization.

The full notation extends the basic constructs shown above in many ways, to account for asynchronous communication, for concurrency within roles (illustrated in Figure 8), for timed interactions, and others [Ould,1995]. RAD supports a number abstraction mechanisms, including process composition, and process encapsulation/decomposition where an activity at a given level of detail becomes a full blown process. RAD does not prescribe its own data representation formalism—and that is a good thing— and instead assumes that the static properties of the data or resources are represented using some form of entity-relationship diagram. However, RAD prescribes the use of *entity lifetime histories* (ELH) to represent the different states that an entity or a resource undergoes during the process in the form of state *hierarchies*. State hierarchies do not represent the transitions between states explicitly; those are implicit in the process.

What can one do with a RAD process model? Ould stressed that STRIM—and its main notation RAD—are aimed at developing models that are "revealing and communicative" [Ould,1995]. He proposed an analysis method based using a combination of simulation, local optimization, and the detection of "anti-patterns". Simulation relies on the propagation of *tokens* throughout the diagram reflecting the progress of the process, much like with Petri nets. The other two methods rely on common sense heuristics whereby one submits a RAD model to a series of validations or questions aimed at identifying redundancies, conflicts, etc. Other researchers have attempted formal analysis of RADs since (e.g., see [Phalp & Shepperd,2000]).

4.4 Event Process Chains (EPC)

The Event-Driven Process Chains (EPC) method was developed in the early nineties in a joint effort between researchers at the University of Saarland and SAP [Keller et al., 1992]. Event Process Chains (EPC) are used to describe business processes at the informal business level, and are meant to support business users rather than formal manipulation.

Event process chains are composed of three basic ingredients: *events*, *functions*, and *connectors*. Figure 9 represents a book borrowing process using the EPC notation. Within a process chain, functions and events alternate, possibly separated by connectors. Functions have exactly one inbound arc and one outbound arc. Events have at most one inbound arc and at most one outbound arc. *Events* in EPC are half-way between real events, as occurrences with no "duration", and states, as steady conditions of the process or of some entities manipulated by it[7]. Connectors can have multiple inbound arcs, and a single outbound arc, or the opposite. They have somewhat similar semantics to connectors in IDEF3. An AND connector with one inbound arc but two or more outbound arcs means that the current process is forking into many parallel threads. An AND connector with two or more inbound arcs and a single outbound arc plays the role of a process join. A forking XOR connector means that only one of the outbound branches is taken. A joining XOR connector means that only one possible path can lead to the current function. A joining OR connector means that two or more possible paths

[7] Some of the litterature refers to states as *preconditions* and *postconditions* of the functions.

can lead to the next function. A forking OR connector means two alternative courses of action, which are not necessarily mutually exclusive.

The ERP and BPR tools that use EPC diagrams augment them with a description of the inputs and outputs of the various functions, and with references to other diagrams showing:

1) a data model, such as an entity relationship diagram, showing the structure of the inputs/outputs;
2) a hierarchical organization of the enterprise, and links between the functions of the EPC diagram and the actors (departments, individuals) who perform the work;
3) a hierarchical functional decomposition diagram and cross-references to the functions in the EPC diagram;
4) an entity life history-like diagram (see discussion of entity life histories in section 4.3) showing the various states that entities can take during the process.

In terms of practical use, EPC is one of the most widely used languages for process modeling, with SAP and the major BPR tool vendors supporting it.

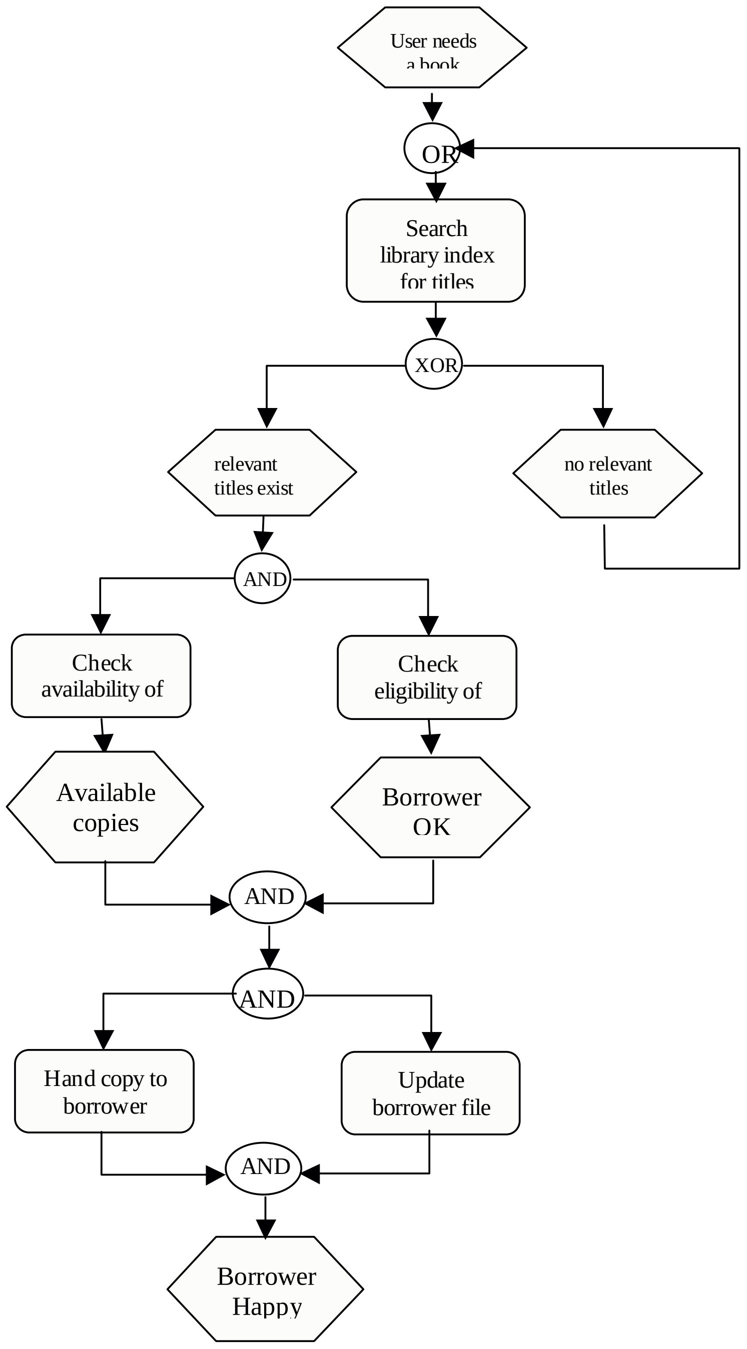


Figure 9. The book borrowing process in EPC

From a theoretical point of view, several researchers have tried to map EPCs to Petri nets to take advantage of Petri nets formal analysis arsenal. However, EPC was shown to suffer from ambiguous semantics [v.d. Aalst, 1999], [Kindler, 2003]. For example, the meaning of the joining XOR is not clearly stated in the original EPC [Keller et al., 1992][8]. Interestingly, it is not clear what that meaning *should* be, either. But also note that the same could probably be said about IDEF3's join boxes (see section 4.1.3).

Although not as rich as Petri nets or RADs, EPCs are interesting because of their use in commercial products. They are supported by major vendors of enterprise resource planning solutions and business process re-engineering tools (e.g. SAP, Aris, LiveModel/Analyst). This may be an indication about the level of complexity that real users can tolerate.

4.5 Resource Event Agent (REA)

The Resource Event Agent (REA) framework is another triad that has been used to model business processes [McCarthy, 1982], and that may gaining ground as a process modeling framework for inter-enterprise commerce [Haugen & McCarthy,2000]. In a pioneering paper, William McCarthy proposed the Resource Event Agent framework as a way of representing accounting information that would solve the problems experienced by existing accounting procedures and accounting software, including [McCarthy, 1980]:

1) limited dimensions in the accounting data, supporting a limited number of measures,
2) a rigid and too coarse classification of accounting data, leading to inappropriate classification of accounting entries or to ignoring some accounting information,
3) high aggregation level of accounting data, precluding analyses by decision makers who may need different views on the data,
4) poor (or lack of) integration with other data contained in information systems.

According to McCarthy [McCarthy, 1980] and Geerts & McCarthy (1997), accounting systems of today use bookkeeping principles (*double entry*) first laid out more than 500 years by Luca Pacioli, a Franciscan monk from Venice. The REA framework aims at a more detailed and more expressive representation of what "needs to be counted", and which is, primarily, *economic resources* and *economic events*. Economic *resources* are defined as objects that are scarce and have utility, and are under the control of an enterprise [Ijiri, 1975]. Economic *events* are defined as "a class of phenomena which reflect changes in scarce means [economic resources] resulting from production, exchange, consumption, and distribution"[Yu, 1976]. Actors come into play as participants in economic events [McCarthy, 1982].

Figure 10 shows a metamodel of the REA framework represented using the ER notation. The stock flow relationship links a resource (stock) to an event (flow). An event either increments or decrements a resource. For each event that modifies a resource on one end

[8] The problem can be stated as follows. A joining OR means that whichever incoming "event" is reached, we move on to the next function. With the joining XOR, we need to make sure that only one of the input states is reached, but we don't know how long we would have to "wait" to make sure that nothing is coming down along the other path.

of a transaction, there is a dual event that modifies the resource on the other end of the "ledger". For example, a sale is an exchange of a product for cash between a buyer and a seller. The inventory (stock) is decremented on the seller's side, but incremented on the buyer's side. Conversely, the cash is decremented on the buyer's side, but incremented on the seller's side. The actors involved in an economic event are classified into economic agents (actors outside the organization) and economic units (actors *within* the organization). Economic units represent both individuals and *units*, such as departments or teams. The responsibility relationship reflects the hierarchical relationships (subordination) within the organization.

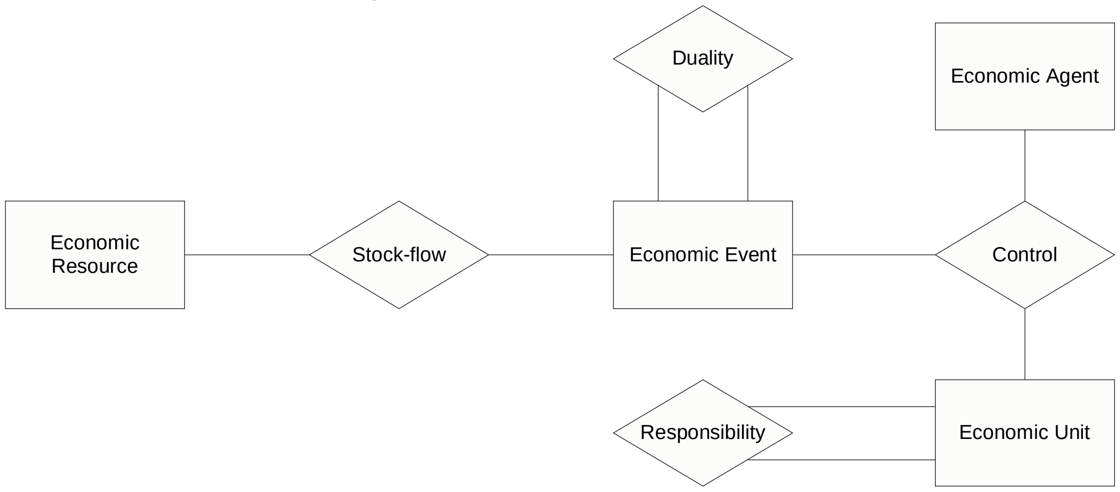


Figure 10. The REA framework metamodel [McCarthy, 1982].

Figure 11 shows an example REA diagram showing a purchase, seen from the perspective of the buyer [McCarthy,1982]. Indeed, the buyer is incrementing his inventory and decrementing his cash. A dual process is taking place at the vendor's organization, where the inventory is decremented and the cash is incremented.

As a general-purpose business process modeling language, REA is not as rich as languages such as Petri nets (section 4.2), RADs (section 4.3) or extended EPCs (section 4.4), and for a good reason: it was not meant to *describe* the dynamic behavior of processes *per se*; however, it is very good at the job for which it was developed: *recording* the transactions that take place in the course of business process execution so that they may be counted. An REA diagram is nothing but a data model to be implemented by a database that both accountants (and accounting software) and other decision makers may use. The overriding concern for data modeling and implementation—which is not found in the other approaches—has two consequences. On the negative side, the representation of processes is limited to what can be recorded. On the positive side, whatever is modeled in REA is readily implementable.

REA is gaining a following as a language for representing inter-enterprise processes [Haugen & McCarthy, 2000]. Its focus on transactions makes it an appropriate language for describing inter-enterprise transactions. The ebXML standard, described in section 6.2 of this paper, use REA diagrams to document some of the partner profiles.

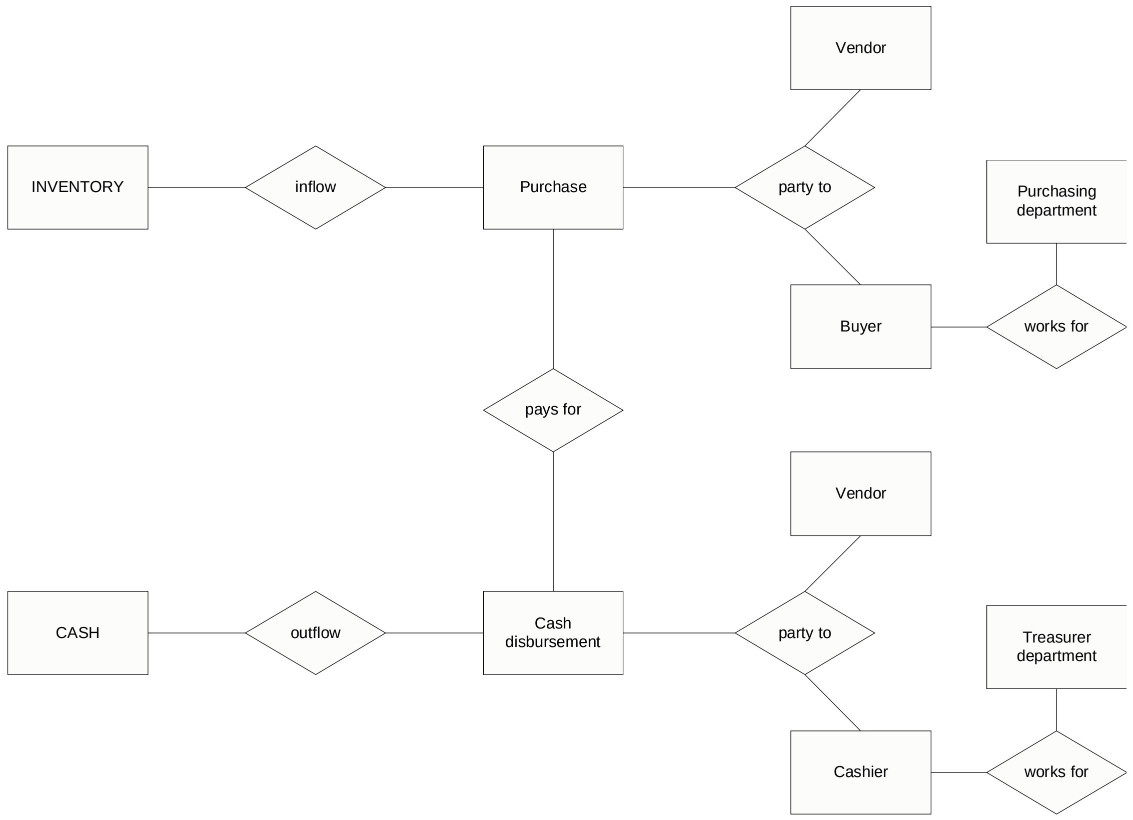


Figure 11. A purchasing process, seen from the buyer's perspective.

4.6 The Business Process Modeling Language (BPML)

The Business Process Modeling Language "defines a formal model for expressing executable processes that addresses all aspects of enterprise business processes, including activities of varying complexity, transactions and their compensation, data management, concurrency, exception handling and operational semantics" [BPMI,2003]. The BPML specification has been developed by the Business Process Management Initiative, an industry organization founded in 2000 to develop standards for business process management, both within and between organizations. The BPML makes extensive use of XML in two different places:

1. XML is used as a presentation and serialization format for business process descriptions;
2. XML Schema descriptions (XSD) are used to specify the data types manipulated by the processes.

We discuss the main constructs of the BPML below.

BPML is heavily oriented towards execution, and the reader will find strong analogies between BPML constructs and common programming language constructs. Roughly speaking, a *process* is a set of *activities* executed within a *context*. An *activity* is "some thing to be performed". BPML comes with a predefined set of fairly low-level activities, corresponding to control structures in programming languages[9]. From a transactional point of view, activities can be *atomic* or not. Most simple activities are *atomic*, and complex activities are, by default, non-atomic. The *context* of an activity is its *execution context*. It is characterized by the set of properties that are specific to that activity (local variables, exceptions, internal functions/processes, etc.). The closest thing in programming terms is the stack/call frame for a function. A *process* is, in turn, a complex activity that defines its own execution context. The closest thing, in programming terms, is a function or a procedure. Some processes are visible everywhere, others are local to other processes. Processes can be triggered in one of two ways: 1) through an explicit call from another process or activity, or 2) through an event. Processes have input/output parameters. These parameters are read from, and written into, the *properties* of their enclosing execution context. *Properties* are variables that are local to a context. We can specify a *type*, a *default value*, and an expression for assigning them a value. The type can be any valid XML Schema, including simple types, complex types, derived types, as well as anonymous types defined through element declarations.

BPML has a handful of high-level constructs that are not typically found in programming languages, such as the notions of *schedules*, *transactions*, *compensation*, and *correlation properties*. A *schedule* is a series of time events. We can associate a schedule with a process, where the first event launches the process and subsequent events launch different parts of the process. BPML supports inter-process *transactions* as long their invocation is performed within an atomic activity. Hence, if one process aborts, the results of the other(s) are retracted. With long-running "transactions" (that can take hours, days, or more), the usual transaction mechanism is not workable because it locks out shared resources for long periods of time. In that case, BPML supports the notion of a *compensation process* whose purpose is to undo the effects of processes. Finally, processes have *correlation properties* to help relate process instances to each other. Indeed, within a given organization, we typically find multiple instances of the same process (e.g., handling a customer order) running at the same time. We can *correlate* activities (or sub-processes) using instance-variable like properties of processes. For example, we can have several instances of the "handle customer order" process. To relate particular activities to the same customer order, we use the *order number*.

Clearly, BPML is a language for executing processes, not one for communicating process descriptions to people. BPMI is working on a graphical notation, but judging from the constructs discussed above (context, schedule, exceptions, etc.), either the notation will be undecipherable to business analysts, or the notation will only reflect a subset of BPML. Furthermore, the language does not handle organizational concepts such as resources, roles, actors, or organizations. Finally, data modeling (with XSDs) is limited to the types of messages exchanged, which leaves a lot unsaid about the data.

4.7 The Architectural Modeling Box for Enterprise Redesign (AMBER)

[9] There are seventeen activities including Assign, Call, Foreach, Sequence, Until, While, Synch and Switch.

The AMBER language [Eertink et al., 1999] has been designed with the dual goal that the models produced using this language be highly understandable and, yet, that they be analyzable by appropriate software tools.

Three aspects domains (views) can be expressed in AMBER:

1. The *actor* domain, to describe organizations, departments, systems, and people.
2. The *behavior* domain, where the basic concept is the notion of action, described by a number of key attributes: the actors involved, the enabling conditions, and the result (outputs).
3. The *item* domain, to describe the items handled in the business processes.

Actors are equipped with *interaction points* that indicate locations where they may interact with their environment. Such interaction points may denote relationships with more than two entities, e.g., ternary relationships. Similar to some of the other notations, complex behaviors, described in terms of possible actions and causality relations between those actions, can be specified using and/or-split/join forks, enabling vs. disabling conditions, loops, blocks, etc. Items can also be coupled to interaction point relations, in order to make explicit the data involved in the interaction; items can also be coupled to actions, to specify the data on which the behavior is performed; no special language is used, though, for specifying data items—a subset of UML is expected to be used.

Although the process description notation of AMBER does not seem to bring any new major concepts, the variety of formal analyses that can be performed on the models is certainly the most interesting feature of AMBER. Analyzability is attained through a number of formal models used for different aspects. For instance, a number of structural analyses are based strictly on the formal syntax, e.g., control or dataflow analysis. Behavioral properties can be analyzed based on an appropriate operational semantics, including checking temporal properties using model-checking [Janssen et al., 1998]. Quantitative properties—e.g., performance, completion time, or critical paths analysis—are analyzed using techniques based on queuing theory or graph models.

5 Process integration languages

5.1 Workflow modeling languages

Etymologically, the term workflow means *flow of work*, i.e., how a complex task gets done by moving work items (products or documents) through different—often specialized—work stations according to some—often predetermined—sequence. The two workflows that often come to mind are assembly line flows in manufacturing, or office workflows, whereby office documents are routed between office workers according to a predetermined sequence based on factors such as skills, data dependencies, and authority. In both cases, it pays to automate the flow and to monitor its progress [WFMC,2002]. In an assembly line, the flow of work is enforced mainly through assembly line design, i.e., the sequencing of machines or work stations, conveyor belts (positions and pace), input (material) bins, etc. In an office automation system, a software program moves electronic documents around between specific functional roles within the organization. The past fifteen years have seen increased research interest and a proliferation of software tools for managing workflows that go beyond manufacturing processes and administrative (support) processes [Dayal et al.,2001].

The Workflow Management Coalition (WfMC), a coalition of over three hundred member organizations, including tool vendors, users, service providers, and research institutions, aims at promoting the use of workflow technology through the development of standards. The WfMC defines workflow as “the computerised facilitation or automation of a business process, in whole or part” [WFMC,1995]. A workflow management system is defined as “a system that completely defines, manages and executes ‘workflows’ through the execution of software whose order of execution is driven by a computer representation of the workflow logic” [WFMC,1995]. Roughly speaking, a workflow management system automates the flow of work within a business process. Individual activities within a process may be performed by calling a software application, or by prompting human actors for input (data or decisions). Because different tool vendors use different process modeling languages (RAD, EPC, UML, etc.), an important aspect of WfMC is to develop standards that allow the interchange of process descriptions between different tools. One way of allowing the interchange is to propose a standard process description language that all tool vendors have to use, in the same way that UML is a standard modeling language. The WfMC tried to focus instead on the interchange of process descriptions. This is achieved through a combination of two elements:

1) an extensible process description metamodel, and
2) a workflow process description language (WPDL)

We briefly discuss each of them.

Figure 12 shows the main entities and relationships of WfMC’s process description metamodel. At first glance, this process metamodel is not much different from the languages seen so far—and that is a good thing. However, the executability of process descriptions in this language is apparent in many places. We see that a process activity may consist of invoking a computer application (referred to as **WorkflowApplication** in the metamodel). The WfMC has actually standardized the way such applications may be invoked [WfMC,1998]. Process nesting is embodied in the *implemented-as* association between **WorkflowProcessActivity** and **ProcessReference —** a **ProcessReference** is a kind of process invocation. A minimal organizational metamodel is provided in the specification (not shown in Figure 12) which identifies a **WorkflowParticipant** as either a **Person**, a **Resource**, a **Role**, or an **OrganizationalUnit**. For each one of the metamodel entities, the specification includes a set of required attributes and a set of optional attributes, and makes provision for users or tool vendors to add their own attributes. Such attributes will be correctly parsed across tools, but their run-time semantics can only be implemented through common agreement between users of these attributes, and vendors of the tools that will run such process descriptions. The WfMC encourages its members to submit new attributes for possible inclusion in the standard.

Some of the provided attributes support reasonably rich real-time semantics, with durations, timeouts, and synchronization. The standard also requires a minimal state metamodel for workflow process activities: `notStarted`, `running`, `suspended`, and `completed`. Users and tool vendors should, minimally, support these but may choose to support other states as well.

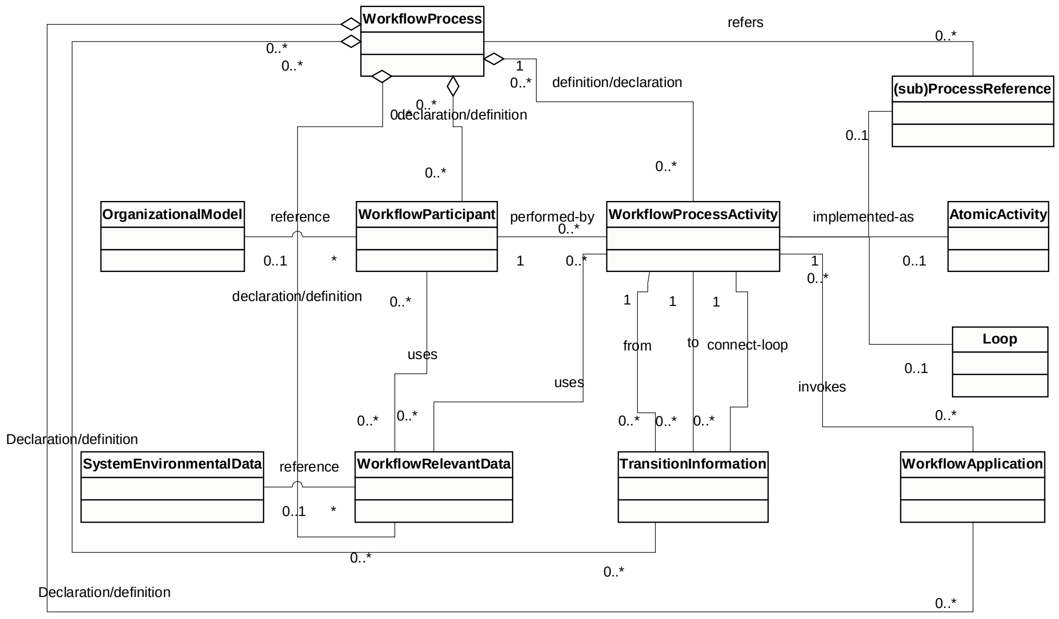


Figure 12. A high level process metamodel [WfMC,1999a].

The Workflow Process Description Language (WPDL) is a language for “serializing” process models described according to the process description metamodel shown in Figure 12. It is given as a grammar in Extended Backus-Naur Form (EBNF). Naturally, an XML-based serialization has been proposed—called XPDL [WfMC,2002b]. The WfMC standards include also an API for accessing run-time representations of process descriptions, with two bindings as of 2002: C and CORBA IDL—and thus, to any programming language that can be mapped to CORBA IDL.

The success of standardization efforts can be best measured by the extent to which tool vendors and users adhere to the standard, and by the extent to which the standard encourages or stifles innovation. We have no way of measuring these two criteria, but if we compare WPDL to UML (or XMI), WPDL comes a bit short, the main reason being that WPDL’s extensibility mechanism (attributes) is fairly limited. In our opinion, the standardization of UML is a rare feat, made possible through a unique combination of factors, including: 1) a consolidation of the OO CASE tool industry fairly early on, 2) a uniquely efficient standards development process, and 3) built-in flexibility of the language achieved at great cost in terms of complexity—the four modeling layers. Parts of the WPDL standard will probably be superceded by more recent technologies for describing process semantics (e.g., see BPEL4WS below). However, those parts of the standards that are within the exclusive areas of competence of WfMC stand a better chance of withstanding the test of time.

5.2 RosettaNet

RosettaNet is a consortium grouping more than four hundred companies from IT, electronic components, semiconductor manufacturing, and telecommunications, working "to create and implement industry-wide, open e-business process standards … [that] form a common e-business language, aligning processes between supply chain partners on a global basis" (see http://www.rosettanet.org). The RosettaNet philosophy postulates that for inter-enterprise e-business to take place, companies have to achieve compatibility at different levels of the computational infrastructure. Accordingly, the consortium has sought to standardize those levels. Figure 13 illustrates the different layers and the relevant standards. The left hand side of the figure illustrates the different layers of the standards, starting from the alphabet for communication between peer information systems, to the specific vocabulary used in transactions, to the implementation framework, dialog, process, and then applications. We will describe each layer in some detail below.

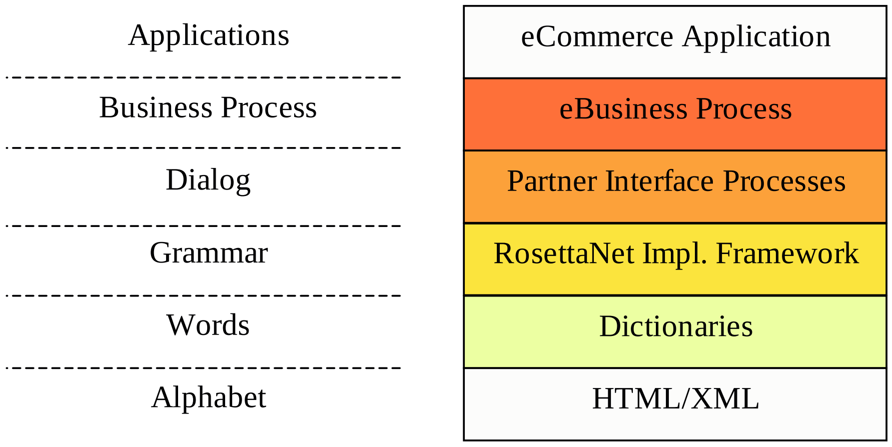


Figure 13. Structure of the RosettaNet standard

The *dictionaries* define a common vocabulary that can be used in electronic transactions. The standard includes two kinds of dictionaries, a *RosettaNet Business Dictionary*, and the *RosettaNet Technical Dictionary*. The business dictionary includes the definition of industry-independent generic business terms (**AccountInformation**, **BillOfMaterial**, **FeeInformation**, **DeliveryException**, etc)**.** The technical dictionary provides a common vocabulary for describing *products* and *services* in target industries—mostly electronic components and information technology—with terms such as **AMPLIFIER IC – RF**, **TRANSISTOR – RF**, **ANALOG MODEM**, **PRINTER INKJET**, and **KEYBOARD**.

The Partner Interface Processes (PIPs) describe generic processes involving two—typically—or more partners. The PIPs are grouped under *segments*, themselves grouped under seven *clusters*, covering different process areas, including *Product Information*, *Order Management*, *Inventory Management*, and *Manufacturing*.

Partner Interface Processes, as the name indicates, are processes that happen at the interface between partners. A typical process starts with one or more tasks within one partner, followed by one message send, followed by more tasks in the second partners, possibly followed by sending a reply to the first partner or a message to another partner, and so forth. Figure 14 shows an example process. The process 3A8[10] is for requesting a purchase order change. It is represented using an activity diagram like notation with two "swim lanes", one for each partner. A PIP may refer to other PIPs either in its preconditions—in this case assuming that another PIP 3A4 has already taken place—or in the steps of the process—in this case the launching of a follow-up PIP, namely 3A7.

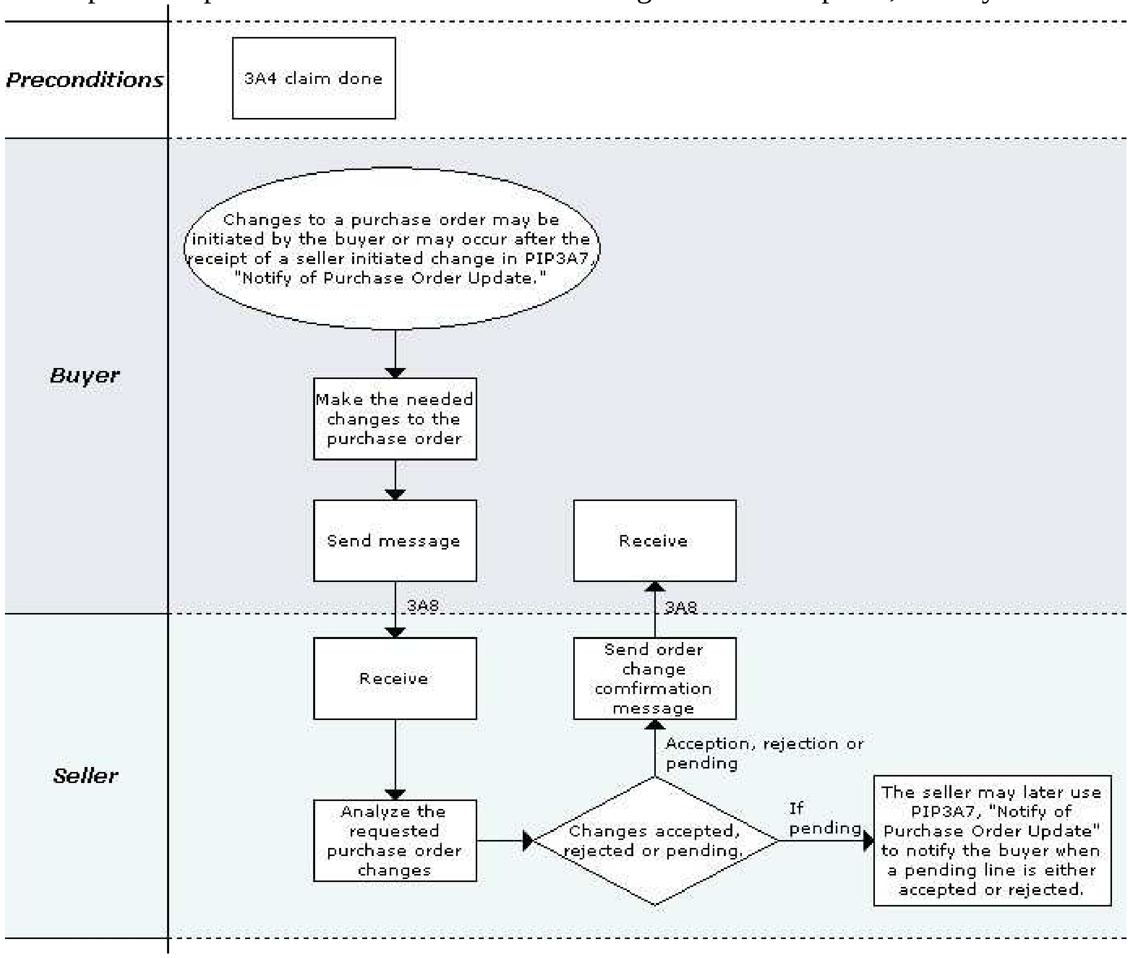

Preconditions
3A4 claim done
Buyer
Changes to a purchase order may be initiated by the buyer or may occur after the receipt of a seller initiated change in PIP3A7, "Notify of Purchase Order Update."
Make the needed changes to the purchase order
Send message
Receive
3A8
3A8
Seller
Receive
Send order change comfirmation message
Acception, rejection or pending
Analyze the requested purchase order changes
Changes accepted, rejected or pending.
If pending
The seller may later use PIP3A7, "Notify of Purchase Order Update" to notify the buyer when a pending line is either accepted or rejected.

Figure 14. PIP for requesting that a change be made to an existing order.

PIP specifications include three *views* on the process: 1) the *business operational* view, representing the business semantics as illustrated by Figure 14; 2) the *functional service view*, describing the required network components (embodying partners) and their protocols, and the mapping between PIP actions and documents; 3) the *implementation framework view*, describing the message formats and specific message formats. Message formats are provided as XML document type definitions (DTDs).

[10] The number 3 is the number of cluster (Order Management) and A the number of the segment (Quote and Order Entry).

The RosettaNet Implementation Framework (RNIF) specification specifies exchange protocols for PIPs that, when followed, should enable participating supply chain members to inter-operate. The specification covers business message formats, specifying XML usage guidelines and message component formats, security provisions, procedures and rules for assembling (packing) and disassembling of messages (unpacking), message transfer protocols, and message flow semantics (one-action, request/response, handling failure, etc.).

The process that the RosettNet organization uses to derive these specifications starts with the modeling of existing business processes by process specialists from the relevant industries or process areas. These "as-is models" are later analyzed to find commonalities, leading to generic business processes which are then used to identify PIPs to be specified. Those generic processes correspond to the upper layer of the RosettaNet standard (see Figure 13) and would typically involve more than one interaction (request/response) between partners, but also includes private tasks. We understand that, in practice, they are more of an intermediary deliverable of the process of specifying PIPs, rather than a standardized output in and of itself. To the best of our knowledge, those generic processes have not been published.

In conclusion, the RosettaNet standard does not propose new concepts for representing business processes. However, it offers a novel focus on processes or process fragments that occur at the interface between two partners, and standardizes message formats using XML. The reader will find similarities in the next two technologies, ebXML, and BPEL4WS.

5.3 ebXML

ebXML (electronic business using XML) is a family of standards aiming at enabling companies to conduct electronic business transactions with other companies by exchanging XML documents over the internet. ebXML is sponsored by the United Nations' Center for Trade Facilitation and Electronic Business (UN/CEFACT, www.unece.org/cefact) and the Organization for the Advancement of Structured Information Standards[11] (OASIS, www.oasis-open.org), a non-profit organization grouping over six hundred members (corporate and individuals) worldwide. At first glance, the goals of the ebXML standard seem to overlap considerably with those of RosettaNet—and Web services, for that matter. However, the focus of the two technologies is different. Whereas RosettaNet focuses on the *business semantics* of inter-enterprise process integration, with its dictionaries and PIPs, ebXML focuses on the integration *process* and on the support infrastructure. The differences will become clearer as we present the key elements of ebXML.

Roughly speaking, the ebXML family of standards addresses the following areas:

[11] OASIS, which was founded in 1993 under the name SGML open, was initially concerned with SGML standards. In 1998, it broadened its scope and changed its name.

1) A standard mechanism (and language) for describing business processes and their associated information (the *Business Process Specification Schema* [ebXML,2001b]);
2) A standard mechanism through which enterprises describe the business processes they support, and *how* they support them (API, message format, communication protocol, etc.), i.e., the *Collaboration Protocol Profile* (CPP, [ebXML,2001c]);

3) A standard mechanism for registering and querying CPPs through a publicly accessible registry (the ebXML *Registry Services* [ebXML,2001d]);
4) A standard mechanism for matching two CPPs to produce *Collaboration Protocol Agreements* (CPA, [ebXML,2001d]);
5) A library containing a set of *core business processes* and supporting *documents* (business messages) that cover the most common business scenarios.

Figure 15 illustrates a business scenario involving two companies A and B that enter into an electronic business transaction under the ebXML framework.

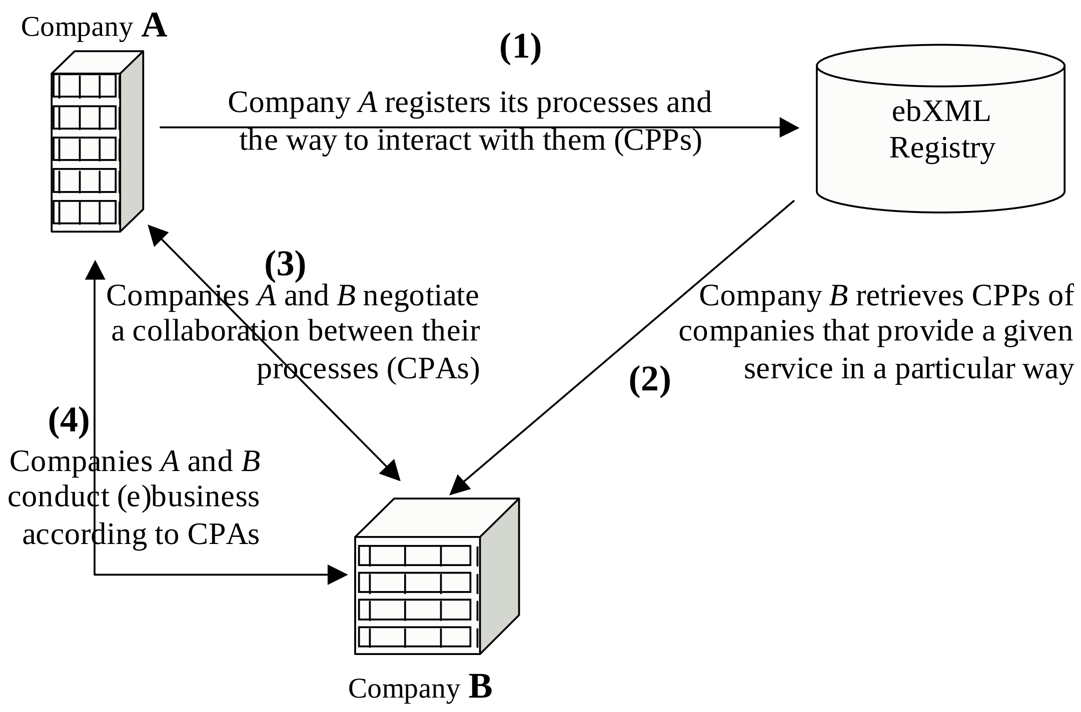

Company A
(1)
Company A registers its processes and the way to interact with them (CPPs)
ebXML Registry
(3)
Companies A and B negotiate a collaboration between their processes (CPAs)
Company B retrieves CPPs of companies that provide a given service in a particular way
(2)
(4)
Companies A and B conduct (e)business according to CPAs
Company B

Figure 15. A collaboration scenario under the ebXML framework

Companies register their *collaboration protocol profiles* (CPP), which describe their business processes and the way they are supported (communication protocol, message format, API, etc.). The ebXML registry enables companies, say company A, to register CPPs and to search for them using different query formats [ebXML,2001d]. A company *B* can query the registry for CPPs that satisfy certain criteria (step 2). Such criteria include: i) offering a given service, ii) offering it competitively, and iii) providing a workable way of invoking it. Once company *B* finds a CPP that matches those criteria, it enters into negotiations with the provider (company A) regarding the business (price, availability), operational (which specific process variant to use), and technical parameters of the transaction (step 3). A *collaboration profile agreement* (CPA) represents that agreement. Once the agreement is in place, it is implemented on both sides, and business transactions can start (step 4).

To ensure that the business processes of the potential business partners are inter-operable, it is not sufficient to specify them using the same notation (BPSS, see [ebXML,2001b]). We have to strive towards standardizing the processes themselves, or failing that, to build processes using a set of reusable business process fragments and business documents. Those business process fragments and business documents constitute the *core library*, which is made accessible through the ebXML registry so that companies may retrieve them to build and document their own processes. Referring back to Figure 15, in practice, company *A* would have built its CPPs by first consulting the ebXML registry for reusable processes and documents.

In practice, the core library may be populated … by RosettaNet's *partner interface processes* or PIPs. This enables the ebXML community to leverage the efforts of the RosettaNet community. Compatibility with other emerging standards has been an explicit requirement in ebXML [ebXML, 2001a], although enforcing it is non trivial considering that a number of initiatives with overlapping goals started and delivered their specifications within a few months of each other (Web services, RosettaNet, ebXML). While we don't expect all those efforts to merge, we do expect the three communities to sharpen their focus on the areas where they have the most added value: RosettaNet focuses on business content, ebXML focuses on the architecture, and Web services (which are also used in BPEL4WS, see section 5.4) focuses on the support technology. In fact, some prototype implementations have used this mix of technologies [Dogac et al.,2002]

The *business process specification schema* (BPSS, [ebXML,2001b]) is ebXML's process modeling language. In ebXML, an inter-enterprise process is called a *business collaboration*. Each partner plays a *role*. A business collaboration consists of a set of business transactions sequenced according to some *choreography*. Each business transaction involves the exchange of *business documents* and, possibly, the triggering of *business signals;* each transaction also has a *requesting role* and a *responding role*. Business transactions and collaborations are typically defined between two partners. However, *multiparty* transactions and collaborations can be synthesized from binary ones [ebXML,2001b:12]. Business collaborations can be nested. The choreography is described using UML's activity diagrams. The ebXML community recognizes the role of patterns in specifying processes using BPSS. Figure 16 shows a simple binary collaboration involving two transactions, and one of two terminal states (success or failure), depending on the outcome of the second transaction. Note that the type **BusinessTransactionActivity** is used to refer to activities that are atomic (the ones that are *business transactions*) and to other embedded collaborations (processes, not illustrated here).

```
<BinaryCollaboration name="Product Fulfillment" timeToPerform="P5D">
      <InitiatingRole name="buyer"/>
      <RespondingRole name="seller"/>
      <BusinessTransactionActivity name="Create Order"
            businessTransaction="Create Order"
            fromAuthorizedRole="buyer"
            toAuthorizedRole="seller"/>
      <BusinessTransactionActivity name="Notify shipment"
            businessTransaction="Notify of advance shipment"
            fromAuthorizedRole="buyer"
            toAuthorizedRole="seller"/>
      <Start toBusinessState="Create Order"/>
      <Transition fromBusinessState="Create Order"
            toBusinessState="Notify shipment"/>
      <Success fromBusinessState="Notify shipment"
            conditionGuard="Success"/>
      <Failure fromBusinessState="Notify shipment"
            conditionGuard="BusinessFailure"/>
</BinaryCollaboration>
```

Figure 16. A sample *binary collaboration* using ebXML's BPSS.

We feel that the full choreography language is not as rich as that of BPML, or better yet, that of the more recent BPEL4WS, discussed next. We argue that the main contribution of the ebXML standards resides in the integrated eBusiness framework, more than on the strength of individual specifications.

5.4 BPEL4WS

The Business Process Executable Language For Web Services (BPEL4WS, [Andrews et al., 2003]) is a language for modeling executable business processes written for the Web services context. The basic idea is that a process may be thought of as a collaboration between services or tasks described in the Web services format, and more specifically, in the WSDL language. Whereas WSDL defines a syntax for expressing services, but says little about the interaction model (or rather assumes a simple one-way or round-trip message passing protocol), BPEL4WS describes the entire interaction sequence. BPEL4WS has two target uses, which are clearly stated and explained:

1) executable processes: these describe actual business processes that are internal to an organization and are completely specified (i.e., executable),
2) abstract processes: these are the parts of a business process of an enteeprise that is *exposed* to outside processes in the context of an inter-enterprise interaction.

This distinction is very helpful and not made in the case of BPML (see section 4.6). This, plus the fact that BPEL4WS makes provision for *roles* and *partners*, makes it more appropriate for describing inter-organizational processes.

Roughly speaking, a BPEL4WS process description consists of a declaration part that introduces various elements needed to describe the process, followed by the actual description of the process. The declaration part includes the following elements:

1) *A description of the messages exchanged between services*. The message structure is similar to that used in Web services: a message consists of *parts*, each with a

name and a type. The type component is described using XSD types, as with BPML. The use of XSD is not exclusive, and BPEL4WS can accommodate other type systems.

```
…
<message name="POMessage">
      <part name="customerInfo" type="sns:customerInfo"/>
      <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
<message name="InvMessage">
      <part name="IVC" type="sns:Invoice"/>
</message>
…
```

2) *A description of the services invoked.* The description follows the WSDL standard: each service, defined as a *portType*, consists of a bunch of operations. An operation has a name and a set of parameters. The parameters are nothing but messages playing the role of inputs and outputs. A service is defined using WSDL port type concept (like an interface concept or an API).

```
…
<portType name="purchaseOrderPT">
      <operation name="sendPurchaseOrder">
            <input message="pos:POMessage"/>
            <output message="pos:InvMessage"/>
            <fault name="cannotCompleteOrder"
                              message="pos:orderFaultType"/>
      </operation>
</portType>
…
```

3) *A description of the contracts between process participants.* Each contract—called *service link type*—defines roles and associates them with interfaces/portTypes. For example, in a supply chain example, we have customers, businesses, and their suppliers. The interaction between a customer and the business to fulfill an order, and between the business and its suppliers to restock, are managed by two separate contracts/service link types, each of which identifies the *roles* played by each service interface (port type)

```
…
<slnk:serviceLinkType name="invoiceLT">
      <slnk:role name="invoiceService">
            <slnk:portType name="pos:computePricePT"/>
      </slnk:role>
      <slnk:role name="invoiceRequester">
            <portType name="pos:invoiceCallbackPT"/>
      </slnk:role>
</slnk:serviceLinkType>
…
```

4) *A description of partners*. While service link types define the various contracts, it is not clear which entity will play which side in the contract. A BPEL4WS process thus contains an identification of the various partners in the different contracts, the roles they play in the contract (service link type), and the role ("myRole") the enterprise doing the modeling plays in that contract. Those partners will be referred to later in the description of the steps of the process: each step is performed by a partners (or self). Here, we show a description of two partners, where the first is the customer and the second is the “invoiceProvider”

playing the role “invoiceService” in the contract (service link type) “invoiceLT”, where I, myself, play the role of “invoiceRequester”.

```
...
<partners>
      <partner name="customer"
            serviceLinkType="lns:purchaseLT"
            myRole="purchaseService"/>
      <partner name="invoiceProvider"
            serviceLinkType="lns:invoiceLT"
            myRole="invoiceRequester"
            partnerRole="invoiceService"/>
      …
</partners>
...
```

Other elements in the declaration include *local variables* defined within the scope of the process and exchanged as inputs/outputs between the process steps, and a description of *fault handlers*, which specify the desired response in case of a fault.

The process is defined using a flow, which is a partially ordered set of activities that correspond to invocation of operations, defined in the various services, that will be performed by the partners identified above. Process flow supports sequential activities (using the "sequence" activity), concurrent activities (using the "flow" activity), and control dependencies between process steps using the link mechanism, which ensure that a particular process step can only be executed after another step has completed. Like with BPML processes, BPEL4WS processes include descriptions of *compensation handlers* and support the notion of scope (called *context* in BPML) and *correlation sets*, which are data values that uniquely identify process instances (see also BPML). The following shows excerpts from the body of the process. The (purchase order handling) process starts by receiving a purchase order (PO) from a customer. It then makes a copy of the customer info from PO into the customer info of a shipping request. Then it invokes an operation of the partner “shippingProvider” to ship the order to the customer. This example illustrates the use of control dependencies: the “requestShipping” operation is the source of a control dependency (a *link*, called “ship-to-invoice”) linking it to the operation (not shown here) “sendShippingPrice” of the partner called “invoiceProvider”.

```
…
<sequence>
      <receive    partner="customer"
                  portType="lns:purchaseOrderPT"
                  operation="sendPurchaseOrder"
                  variable="PO">
      </receive>
      <flow>
            <links>
                  <link name="ship-to-invoice"/>
                  <link name="ship-to-scheduling"/>
            </links>
            <sequence>
               <assign>
                  <copy>
                        <from variable="PO" part="customerInfo"/>
                        <to variable="shippingRequest"
                              part="customerInfo"/>
                  </copy>
               </assign>
               <invoke  partner="shippingProvider"
```

```
                              portType="lns:shippingPT"
                              operation="requestShipping"
                              inputVariable="shippingRequest"
                              outputVariable="shippingInfo">
                        <source linkName="ship-to-invoice"/>
                     </invoke>
                   ...
                  </sequence>
                  …
            </flow>
            …
      </sequence>
      …
```

BPEL4WS has richer and clearer semantics than BPML. The distinction that BPEL4WS makes between the two usage patterns, i.e., the description of inter-enterprise processes using abstract processes (choreography), on the one hand, and the description of executable internal processes (orchestration), on the other hand, is very useful. The concepts presented above belong to the common core of these usages. In terms of semantic coverage, there are important areas where BPML and BPEL4WS overlap, viz., the representation of data using XSDs, the definition of processes flow, local variables, scope/contexts, correlation sets/variables, fault handlers, etc. We feel that BPEL4WS's concepts are more sharply defined, and that the terminology is more natural. However, BPEL4WS goes beyond BPML by explicitly representing contracts, roles, and partners. This adds clarity to process descriptions. The different levels of maturity between the two standards could be explained by their corporate histories. BPEL4WS is the result of the convergence of two technologies, XLANG and WSFL, and is in its second version. The first version had some shortcomings which were overcome by borrowing concepts from BPML. We, the users of these competing technologies, can only hope that they would converge.

6 Object-oriented languages

Object-oriented models have long been touted as information models that both business people and technologists can understand [Coad & Yourdon, 1988; Henderson-Sellers & Edwards, 1990; Isoda, 2001]. However, the focus has been mostly on *information system* modeling, as opposed to *business process modeling*. Do object-oriented modeling languages have what it takes to model business processes? Clearly, in comparison to the business process modeling languages seen so far, and using the ontology shown in section 2.3, object-oriented notations support a good many concepts needed for business process modeling. On the other hand, they are also missing important ones, such as the notion of a *role*, and organizational aspects, with one notable exception: OORAM [Reenskaug, 1996]. EDOC used UML's built-in extension mechanism to add a *business process profile* [EDOC,2001]. Some of the important concepts that were proposed by EDOC were integrated into the core of UML 2 [OMG,2003]. We first start by discussing OORAM. We then talk about EDOC by first discussing UML 1.x constructs for process modeling, and then talking about the extensions proposed by EDOC. We conclude this section by talking about the new features of UML 2.

6.1 OORAM

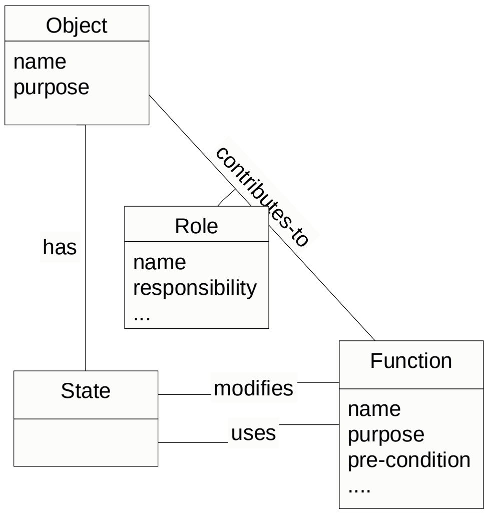


Figure 20-a. A (partial) behavioral ontology of objects in OORAM

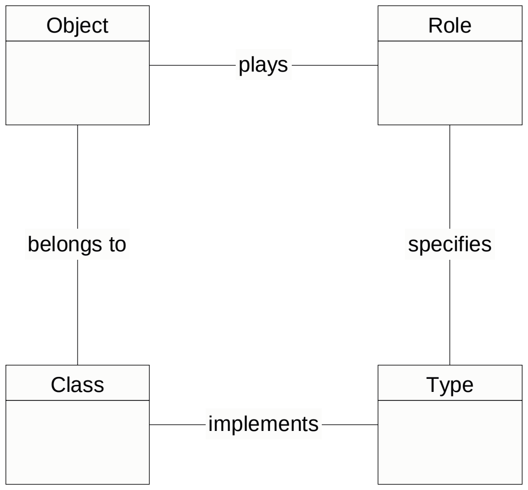
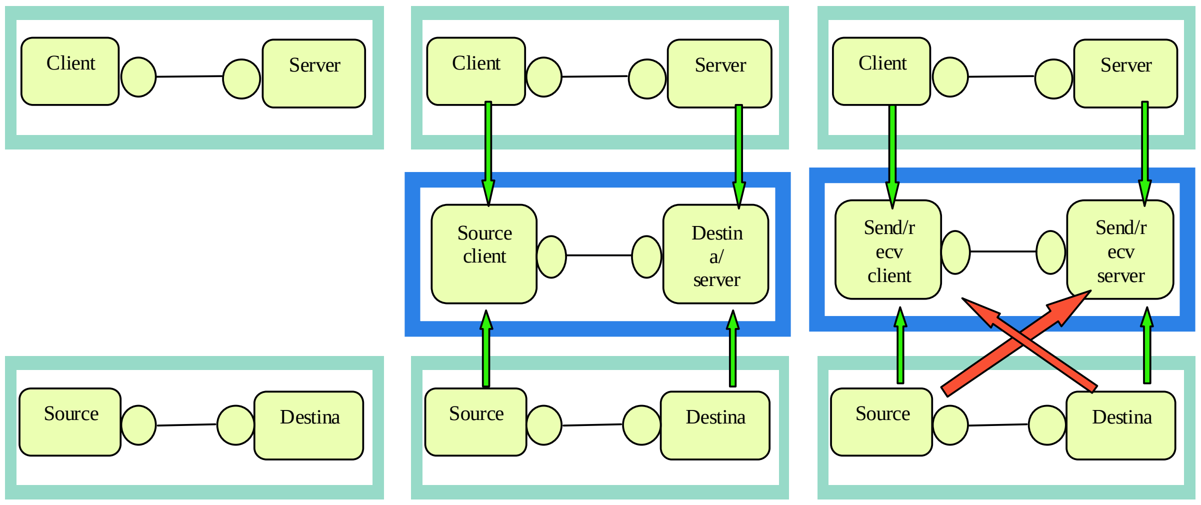


Figure 20-b. A (partial) structural ontology of objects in OORAM



Two role models for FTP client/server & data send / recv

FTP client sends data to FTP server

FTP client sends and receives data from FTP server

Figure 21. Role synthesis in OORAM [Reenskaug,1996]

The Object-Oriented Role Analysis Methodology (OORAM) is a joint effort of the Center for Industrial Research in Oslo, TASKON AS, and the university of Oslo

[Reenskaug, 1996]. OORAM recognizes a *class* as the appropriate *implementation* paradigm for objects, but advocate the concept of *roles* and *role models* as the appropriate *modeling abstractions*. The general idea is as follows. An object plays different *roles* in an information system. Each role corresponds to a coherent set of collaborations in which the object participates, fulfilling its share of a common function. *Role models* describe the structure and behavior of collaborating objects in terms of *roles* as opposed to *classes* [Reenskaug, 1996]. The mapping of *roles* to *classes* is a *design decision*, and the mapping can be many to many. A key step in this mapping is *role synthesis* whereby it is determined that certain roles are to be synthesized so that they could be played by the same objects. Reenskaug compares role synthesis to inheritance in the following terms:

1) inheritance (specialization) is used in programming both for concept building and for code sharing, which are two different intellectual activities;
2) role synthesis is used exclusively for model building, and thus separating the two activities.

Figure 20 shows a partial ontology of objects in OORAM, from a behavioral point of view and from a definitional point of view.

Figure 21 shows an example of role synthesis. We have two collaborations, a *client-server* collaboration, and a *sender-receiver* collaboration. We show two possible syntheses, the first making the client the source for the data, and the server the destination. In the second case, both client and server can send and receive data.

In addition to roles, which are interesting from a process modeling point of view, OORAM has a number of interesting ideas from a pure object-modeling point of view, related to separation of concerns and reuse. Some of the distinctive concepts of OORAM found their way into UML 2.

6.2 EDOC

EDOC is an OMG standard modeling framework aimed at simplifying the development of *component-based* enterprise distributed object computing (EDOC) systems. The modeling framework was based on UML 1.4 and conformed to OMG's MDA development [OMG,2002]. The standard built on the experience of several companies (the submitters) in the area of modeling of enterprise distributed object computing systems, including Data Access Technologies, EDS, Fujitsu, Iona, IBM, Sun, and others. Two factors make this modeling framework particularly relevant to our needs:

1) an extension of UML to provide explicit representation for business processes—the *business process profile;*
2) a built-in traceability mechanism that enables us to trace software components to the actors and activities of the business processes that they support.

We first discuss the principles underlying EDOC. In section 6.2.2, we discuss the *Component Collaboration Architecture* (CCA), which is the technical core of EDOC—and the basis for the business process modeling framework. Finally, we discuss the business processing framework.

6.2.1 Principles

The EDOC standard consists of two major parts : 1) a set of extensions to UML (profiles) to model enterprise distributed object computing (EDOC) systems, and 2) a framework for the application of these extensions to model EDOC systems in a way that is consistent with the MDA development model. The UML extensions include:

1) the Enterprise Collaboration Architecture (ECA), which in turn consists of five profiles—discussed below—each embodying a different view of an EDOC system,
2) a patterns profile, enabling us to model recurring ECA patterns and applying them to specific EDOC systems, and
3) a set of technology-specific models allowing the definition of platform-dependent models, *à la* MDA.

The five profiles included in the ECA are:

1) the *Component Collaboration Architecture* (CCA), which views a system as consisting of a set of components participating in collaborations in which they play specific roles, to fulfill an overall function,
2) the *Entities* profile, which provides UML extensions to represent *problem domain concepts*—as opposed to software artifacts, something that UML *classes* are supposed to handle,
3) the *Events* profile, which provides UML extensions to model event-driven systems,
4) the *Business Process* profile, which specializes the CCA and focuses on the case when the *components* are business activities, tasks or processes,
5) the *Relationships* profile, which extends UML's support for the definition of associations with richer semantics.

These profiles are to be used to represent different *viewpoints* of an EDOC system. In particular, the EDOC standard considers the five viewpoints of the reference model for open distributed processing (RM-ODP), namely, the *enterprise viewpoint*, the *computational viewpoint*, the *informational viewpoint*, the *engineering viewpoint*, and the *technology viewpoint*. We discuss them briefly, in turn.

The *enterprise viewpoint* is embodied in an enterprise specification, which "models the structure and behavior of the system in the context of the business organization of which it forms a part" [OMG,2002]. Referring back to our discussion of section 3.2, the enterprise specification corresponds to the real-world modeling of the enterprise *with* the automated EDOC system in place. This modeling is done in terms of: i) the business processes supported (in part) by the system, ii) the steps in those processes and their inter-relationships, iii) the business rules that apply to those steps, iv) the artifacts acted upon by those steps (resources, entities), v) enterprise objects representing the business entities involved (individuals, departments, other automated systems), vi) the roles that they fulfill in supporting the business processes, and vii) the relationships between those roles. A key concept in enterprise specification is the notion of *community* which is a set of entities interacting to achieve some purpose. Communities are represented by composed components, with associated composition and choreography definitions. We will discuss components and compositions in more detail in the next section (Component Collaboration Architecture).

The *computation viewpoint* is embodied in a *computational specification*, which describes the implementation of the EDOC system (or its components) in order to carry out the functionality identified in the enterprise specification. This implementation is described in terms of interactions between *computational* components that interact through well-defined interfaces (used and provided) to achieve some function.
The EDOC computational specification uses the CCA profile, which is discussed in the next section. A *component* in the computational specification is related to one or more activities in one or more processes in the enterprise specification. Furthermore, an entity in the computational specification corresponds to an entity referenced in an activity or process of the enterprise specification.

The *informational viewpoint* is embodied in the *information specification* which describes the semantics of the information and of the information processing performed by the system, regardless of software packaging—embodied in the computational viewpoint. The information specification includes a configuration of information objects as well as a *functional* description of those objects in terms of states, transitions, and constraints on the entities of the system. The information objects correspond to the enterprise objects in the enterprise specification for which information is held and processed by the system. Information objects are modeled as entities and relationships.

The *engineering specification* defines the architecture of the EDOC system in terms of infrastructure services such as distribution, persistence, and transactions. It is constructed by mapping the computational specification to *technology abstraction models*. The *technology specification* is concerned with the mapping of the engineering specification to a specific hardware and software platform such as EJB, CORBA, COM+, etc. Note that the engineering specification is not concerned with the platform, but does embody general design decisions to be realized or implemented by the platform.

Figure 22 shows the relationship between the different viewpoints, and the profiles used by each viewpoint. In terms of MDA levels, the enterprise specification corresponds to *computation independent models*, the computational specification, the informational specification, *and* the engineering specifications correspond to *platform-independent models*, and the technology specification corresponds to *platform-specific models*.

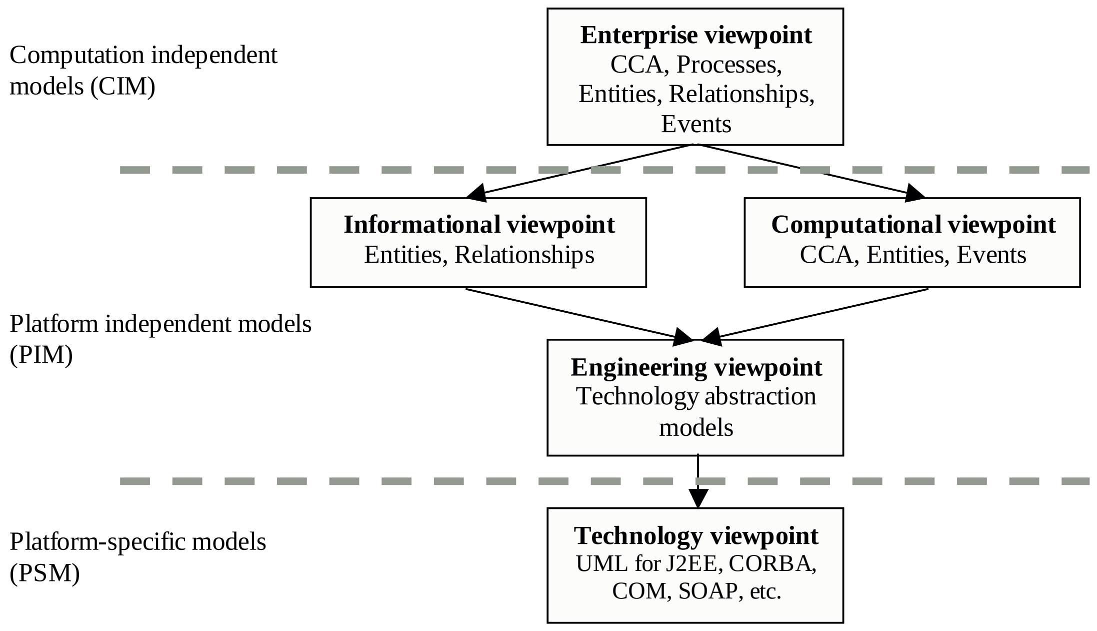

Computation independent models (CIM)
Enterprise viewpoint
CCA, Processes, Entities, Relationships, Events
Informational viewpoint
Entities, Relationships
Computational viewpoint
CCA, Entities, Events
Platform independent models (PIM)
Engineering viewpoint
Technology abstraction models
Platform-specific models (PSM)
Technology viewpoint
UML for J2EE, CORBA, COM, SOAP, etc.

Figure 22. Architecture of the EDOC standard

6.2.2 The Component Collaboration Architecture

The component collaboration architecture views a system as a set of components that interact through a set of ports according to a set of protocols. Components—called *ProcessComponents*—may themselves be decomposed into a set of collaborating sub-components, and so on, recursively. The structure of a *composite* component is specified using two constructs: 1) *composition*, using UML's collaboration diagrams and showing which components are to be assembled and how they are put together; 2) *choreography*, using UML statecharts, to show the coordination of activities required to achieve the function of the composite component.

Figure 23 shows parts of the metamodel of CCA. A process component has ports, enabling it to interact with other components, and properties, enabling us to configure it. There are several kinds of ports; here we illustrate just two, *flow ports* and *protocol ports*. Flow ports are used for simple inputs/outputs, whereas *protocol ports* are used for complex two-way exchanges between two process components; protocol ports are particularly useful for representing business to business transactions. The behavior of a process component is specified using *protocols*, which are sequence of message "sends" and "receives" between two (or more) process components, one of which is the initiator of the protocol, and the other is the responder. A protocol is a subtype of *Choreography* which is, roughly, anything that can be represented by an activity diagram.

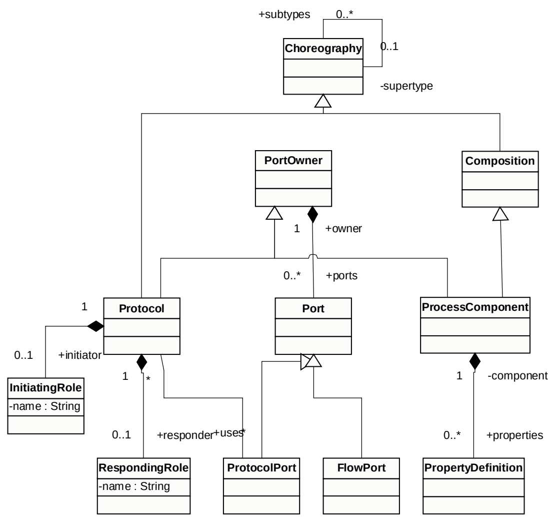


Fig. 23-a. Excerpts of the structural metamodel

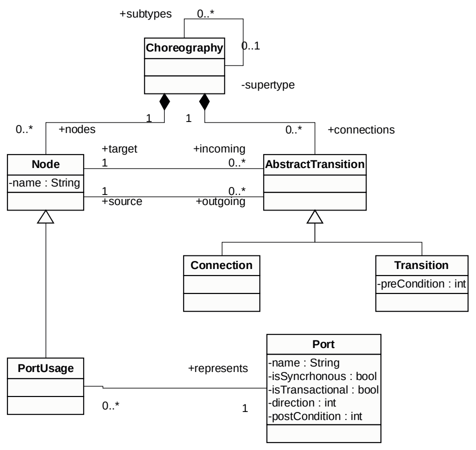


Fig. 23-b. Excerpts from the choreography metamodel

Figure 24-a shows the notation used to represent process components. A ProcessComponent is represented using the UML subsystem icon, to which was added notation for ports and properties. In this case, “Receives” is an input flow port, “Sends” an output flow port. “Responder” is a protocol port in which the component plays the role of a responder. Hence, the protocol starts by receiving a message (A) from the initiator. Conversely, “Initiator” represents a protocol port where the component plays the role of an initiator. The protocol starts by sending message X. Subsequent steps involve sending and receiving additional messages.

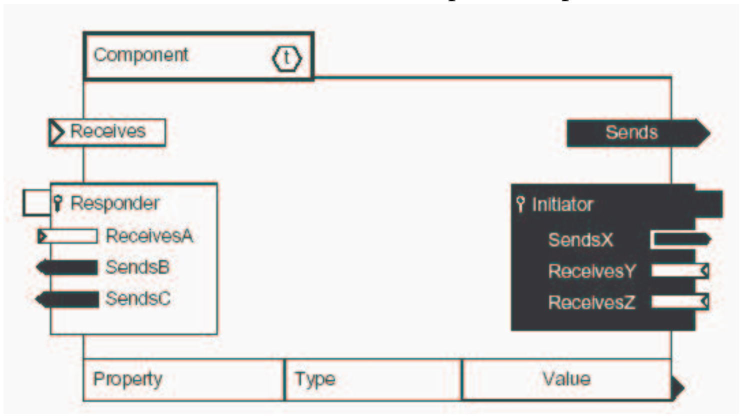


Fig. 24-a. Notation for a component

ProcessComponents are connected by linking their ports (*port connectors*), e.g., linking an output flow port of component A to an input flow port of component B, or protocol ports corresponding to the two roles of the same protocol[12]. A composite component will show the composition of its components inside of it, with the outer input and output ports connected to input and output ports of the subcomponents.

[12] The authors of the standard asserted that the CCA standard was heavily influenced by OORAM, discussed in section 7.1, and by ROOM, a real-time object-oriented modeling methodology.

Figure 24-b shows a special kind of composite component called *community process*. A community process is a composite ProcessComponent that does not interact with its outside environment, and thus has no external ports of its own.

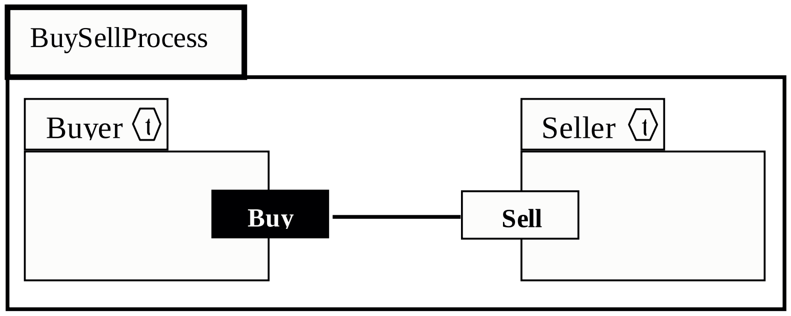


Fig. 24-b. A community process

6.2.3 The Business Process Profile

The business process profile is a specialization of the component collaboration architecture aimed at representing business processes in terms of a composition of business activities (specializations of *process components*). It enables the expression of:

- Complex dependencies between business tasks (data, control, timing)
- Various time expressions on tasks such as durations and deadlines
- Exception handling
- Explicit association between business tasks and the business *roles* that perform those tasks
- Selection criteria for selecting entities (or actors) to fulfill roles at process instantiation time.

Figure 25 shows excerpts from the business process metamodel. A business process is a special kind of ProcessComponent (see previous section), and is thus defined by a composition. We know from the CCA that a composition is an aggregation of ComponentUsages, each describing a role for one of the subcomponents—not shown in figure 23 to avoid cluttering the model. For the case of a business process, the component usages (in essence, the subcomponents of the business process) are *activities*. An **Activity** may be elementary, in which case it is executed by an object bound to the **Performer** role. Activities may also be compound, in which case their structure is described using a corresponding **CompoundTask** (via the "*uses*" association between **ComponentUsage** and **ProcessComponent)**. Note that many activities may be usages of the same **ProcessComponent**—in this case, a **CompoundTask**—and many activities in the same **CompoundTask** may be performed by the same **ProcessRole**. At run-time, the execution of an atomic activity requires us to create or select objects to play the various roles (performer, artifact, and responsible party). The "creationRule" and "selectionRule" attributes of **ProcessRole** are used for this purpose.

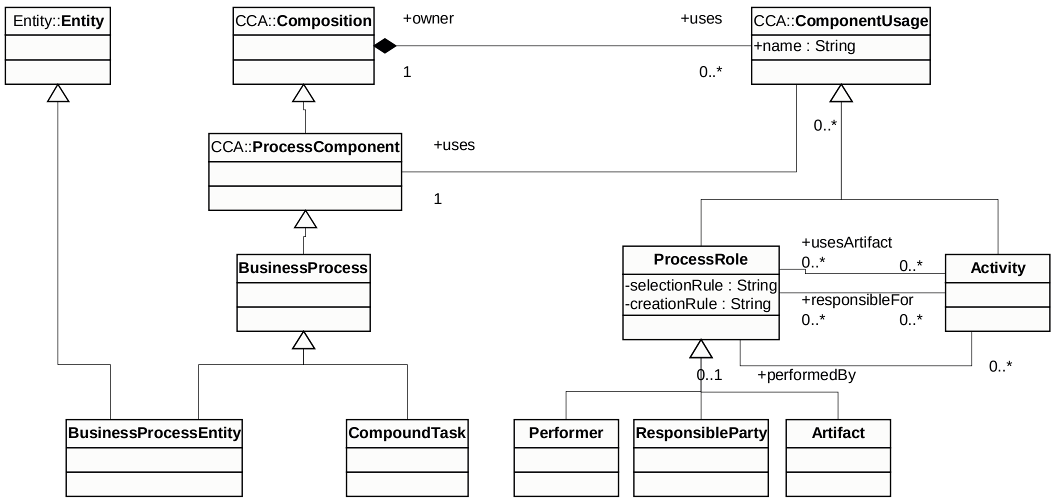


Figure 25. A partial metamodel of the business process profile.

The distinction between a **CompoundTask** (a **ProcessComponent**) and an **Activity** is that the former defines a set of capabilities (the protocols supported by the various ports) whereas a business process may use only one such capability—an **Activity**—in the context of achieving its goal. The difference between a **BusinessProcess** and a **CompoundTask** is that the former is used as the outermost layer of composition for business processes, but **CompoundTask** is used to recursively decompose a process. The **BusinessProcessEntity** represent business process *instances*, whereas **BusinessProcess** represents business process definitions. We may need to represent instances of a process explicitly in case the process is long-running, and in case we have different instances (e.g., different customer orders; see also correlation sets in BPML and BPEL4WS).

In addition to the basic modeling constructs provided by the profile, the EDOC specification provides several other constructs in the form of *process patterns* including the use of activity pre/post-conditions, timeouts, multi-tasking, programming like control structures (loops), and others.

6.3 UML2

UML2 is the latest (and yet to be approved) and largest member of the UML family. The request for proposal, going back to late 2000, requested enhancements in four major areas [UML-RFP,2000a], [UML-RFP,2000b]:

1) *UML fundamentals*: these are best understood in the context of OMG's four layer modeling architecture[13]. The UML 2 RFP called for more precise semantics for

[13] The four layers correspond to, a) the data layer ("John Smith", "IBM"), b) the model layer (**Employee**, **Employer**), c) the metamodel layer (a **Class**, an **Association**, a **UseCase**), and c) the meta-metamodel layer, which corresponds to the language used to represent the concepts of **Class**, **Association** and **UseCase**. The metamodel layer consists of definitions of modeling languages (UML, E/R, OMT, OOSE, etc.). The meta-metamodel layer consists of the language for defining languages. What is confusing in UML is that that language is itself a subset of UML (UML core). If that weren't enough, the Meta Object

UML's meta-metamodel constructs, and alignment with the Meta Object Facility standard.

2) *User-level constructs*: users who build models with UML will use instances of the UML metamodel (**Class**es, **UseCase**s, **Association**s, etc.). The UML2 RFP requested i) a consolidation of existing constructs to avoid overlaps and ambiguity, and ii) new constructs to handle component-based systems.
3) *Object Constraint Language*: the RFP requested that the OCL metamodel be fully aligned with UML. Indeed, OCL expressions refer to UML model elements (classes, associations, attributes), and its metamodel should match that of UML's—which it didn't.
4) *Diagram interchange standard*: while the XMI standard handles the semantic aspects of model interchange, the graphical layout of models was lost during serialization/deserialization of models. UML 2's RFP requested that graphical information be recorded.

The current submission addressed these requirements and introduced a number of minor enhancements to existing constructs.

For our purposes, two major development are of interest. Philosophically, UML2 continues the move away from a modeling language for *object-oriented software* to a modeling language for "systems" in general, that need neither be software nor object-oriented[14]. Practically, UML2 incorporates a slew of new constructs for modeling *hierarchically-structured component-based* systems. The two keywords are *hierarchy* and *components*. UML2 supports the notion of structured classes, which are classes that have interconnected parts. UML2 finally recognizes that structure dictates behavior—something that philosophers of science have known for long (e.g., see [Darden & Rada, 1986]), but that software researchers have been rediscovering in various forms for two decades (e.g., see [Borning et al, 1986], [Mili et al., 1989], [Helm et al., 1990], [Holland et al., 1992]). Thus, collaborations are now specified within the context of structural aggregations. UML2 also has the concepts of components (a special kind of structured class[15]), ports, interfaces, and protocols. UML2 components have richer semantics (ports, protocols, connectors) and are applicable throughout the development lifecycle, as opposed to UML 1.x components, which were deployment time artifacts.

Figure 26 shows the structure of a nested component in UML 2. The top-level component is called "**Store**" and has two ports, an input port (left hand-side), providing the interface "OrderEntry" (full circle), and an output port (right hand side), requiring the interface "Account" (half circle). The internal structure of **Store** shows three subcomponents whose ports are connected as shown: **Order** provides the interface "OrderEntry" (which is exposed/exported by the top-level component **Store**) and requires the interfaces

Facility (MOF) iself defines a language for defining modeling languages, and is very close but not identical to UML's meta-metamodel ☺.

[14] Granted, no object-oriented metalevel constructs have been retired, but the meta-metalevel layer is losing some of its object-oriented flavor.

[15] UML2's metamodel has several classes called **Class**, which belong to different packages. The class **Class** from package *StructuredClasses*, or StructuredClasses.**Class**, inherits from Ports.**EncapsulatedClassifier**, which inherits from InternalStructures.**StructuredClassifier**, which in turn inherits from Kernel.**Classifier**

"Person" and "OrderableItem". These interface are provided by the subcomponents **Customer** and **Product**.

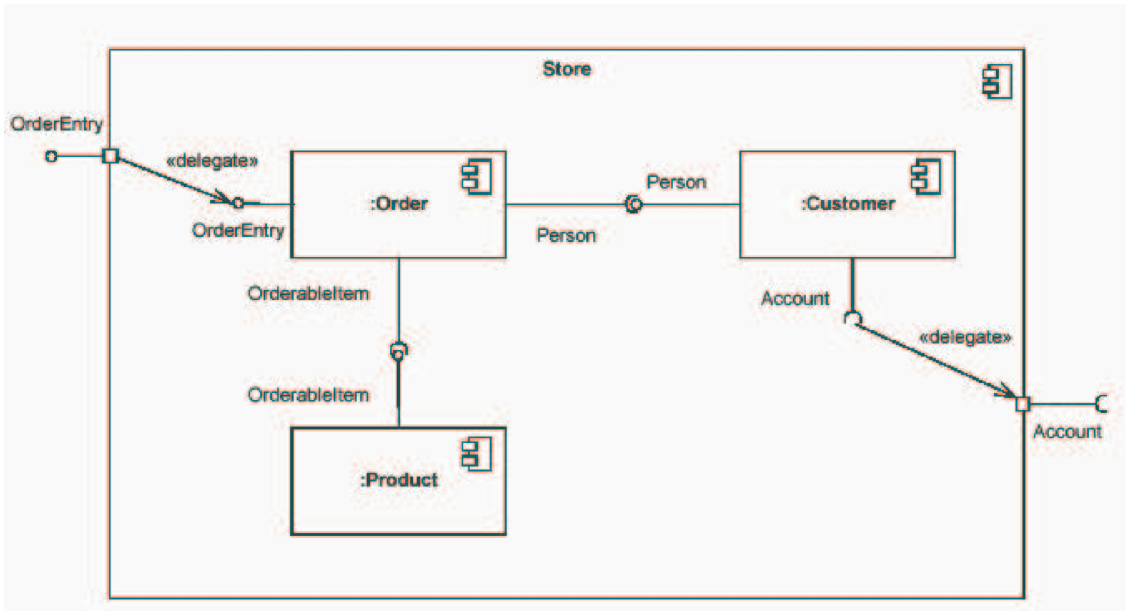


Figure 26. The structure of an aggregate component in UML 2 [UML2-Sup,2003]

In conclusion, UML2 does not yet provide *explicit* support for business process modeling. However, it is doing better than UML 1.x in many respects. First, it continues the shift away from *object-oriented software* to simply *dynamic systems*. Second, it introduces new low-level constructs that handle a number of process-related features that were missing from UML 1.x (timing diagrams, Petri net concepts in activity diagrams, token queuing concepts, and much richer action semantics [UML2-Sup,2003]). Third, the new emphasis on components and on structural and behavioral aggregation provides the foundation for defining business processes. In comparison to EDOC (see 6.2), one could say that UML2 incorporates, as part of the basic UML metamodel, the contents of EDOC's Component Collaboration Architecture (CCA) profile, which was the basis for the business process profile (see sections 6.2.2 and 6.2.3). This is a natural evolution: UML remains the rallying point for interesting ideas in modeling, and newer versions of UML will probably keep integrating extensions developed for earlier ones.

7 Conclusion

In this paper, we have studied a great variety of languages, coming from different scientific traditions, and with widely varying characteristics. A feature by feature comparison of the fifteen or so languages would be useless as the different languages cater to different but complementary needs. However, as the previous sections showed, there is significant overlap between some languages such as BPML versus BPEL4WS, EPC versus REA, ebXML versus RosettaNet, or EDOC versus UML2. At the same time, the families of languages used to organize this paper belie significant differences between some members of the same family. Accordingly, we will use the categorization presented in section 3.1 to classify and compare the different languages. Such a classification is presented in section 7.1.

While the purpose of this study was not to pick a winner language, it is useful to provide a framework for deciding which language to use in a particular setting. This is attempted in section 7.2.

7.1 Comparing process modeling languages

We will compare the various languages presented in the preceding sections along two general dimensions: 1) the contents or *coverage* of the language (which aspects of a business process are covered), and 2) the intended use of the resulting process descriptions. To this end, we will use the categorizations proposed by Curtis [Curtis et al., 1992] and by Ould [Ould, 1995], respectively, and discussed in section 3.1.

Recall from section 3.1 the four *views* of a business process, namely:

1) the *informational view*, describing the objects manipulated by the process. Those objects can be real-world objects (e.g., a machine tool, a raw material, a quantifiable commodity), as well as abstract objects (orders, transactions, audits, etc.);
2) the *functional view*, describing the transformations that the objects that are manipulated by the process undergo. These transformations are expressed in terms of data/functional dependencies;
3) the *dynamic view*, describing the control and timing aspects of the process (when things happen, for how long, etc.);
4) the *organizational view*, representing the roles and actors within an organization who are responsible for executing process activities.

Table 1 shows the views supported by each language. We used four levels of support for each view. “Definitely” means that the language includes a coherent set constructs to represent the view in question, but without judgment as to the expressiveness of that set. For example, most of the languages presented support the dynamic view, but there is obviously a difference in the expressiveness of REA, say, and UML 2.“References” means that the language core does not support the view at hand, but language designers have made a conscious effort to link their models to models in a separate language that handles the view. For example, EPC does not include an informational metamodel, but its designers assume that process descriptions are linked to a data model expressed in some form of entity-relationship notation. The same applies to BPML and BPEL4WS where the description of data refers to an outside type system—for example, XML Schemas.We used “Somewhat” when the language provides an incomplete set of constructs to represent the view. In most cases, it is assumed that a more complete metamodel is available for other purposes. This is often the case for the organizational view. Our process metamodel of Figure 1 shows three entities (and the corresponding relationships) relevant to the organizational view: roles, actors (individuals), and organizations. A number of languages support the notion of *role* but not necessarily *actor* or *organization*.

And then there is the case of UML. The widespread use of UML has made it into a vehicle for modeling ideas, and it has incorporated a number of modeling concepts from a variety of traditions, including the ones discussed here. Thus, by and large, along the information, functional, and dynamic views, UML is overall richer than most of the alternatives. With respect to the organizational view, UML has the ingredients but not yet the *focus* or *tone*. With its four-layer architecture, UML can probably handle *any* view; we just need to define the corresponding metamodel using UML core (or MOF). However, in terms of *user-level* constructs, UML 2 still does not have an extensive business process vocabulary. EDOC does have that vocabulary, but as some of its extensions got incorporated in UML 2—namely, the Component Collaboration Architecture (CCA)—its fate appears uncertain at this point. Will EDOC's business process profile be incorporated into UML 2.x, the way CCA got incorporated, or will it be redesigned on top of UML 2, instead of on top of the CCA profile? This, for now, remains an open question.

Language	Informational view	Functional view	Dynamic view	Organizational view
IDEF0		Definitely		
IDEF1	Definitely			
IDEF3			Definitely	
Petri nets			Definitely	
RAD	Somewhat		Definitely	Definitely
EPC	References		Definitely	References
REA		Definitely	Definitely	Definitely
BPML	References		Definitely	Somewhat
WPDL	References			
RosettaNet	Definitely		Definitely	
ebXML	Definitely		Definitely	Somewhat
BPEL4WS	References		Definitely	Definitely
OORAM	Definitely	Somewhat	Definitely	Somewhat
EDOC	Definitely	Definitely	Definitely	Definitely
UML2	Definitely	Definitely	Definitely	Has the ingredients

Table 1. A comparison of process modeling languages in terms of coverage.

A feature by feature comparison of the various languages along any given view is beyond the scope of this paper. Furthermore, to be useful, such a comparison would need to take into account the intended use of the corresponding model. We briefly compare the presented languages along the use dimension, using Ould's classification [Ould, 1995]. Recall from section 3.1 the potential uses of a process model:

1) description, to either a human being or to a machine,
2) analysis, both qualitative (for re-engineering) and qualitative (for detecting deadlocks, races, etc.), and
3) enactment, for both simulation and execution.

Language	Description		Analysis		Enactment	
	Human	Machine	Qualitative	Quantitative	Simulation	Execution
IDEF0	Yes		Yes			
IDEF1	Yes		Yes			
IDEF3	Yes		Yes			

Petri nets	Somewhat	Yes	Somewhat	Yes	Yes	Somewhat
RAD	Yes		Yes	Somewhat	Somewhat	
EPC	Yes	Yes	Yes	Possibly		
REA	Yes		Somewhat	Yes		
BPML	Yes				Yes	
WPDL		Yes			Yes	Yes
RosettaNet	Yes		Yes		Somewhat	
ebXML		Yes	Yes		Somewhat	
BPEL4WS		Yes		Yes	Yes	Yes
OORAM	Yes	Yes		Yes		
EDOC	Yes	Yes	Yes	Possibly	Yes	
UML2	Yes	Yes	Yes	Possibly	Yes	

Table 2. Potential uses of the various languages.

This matrix belies a few trends. First, as would be expected, languages for communicating processes to humans do not support those tasks that require formality and precision, namely, quantitative analyses and enactment. Second, the machine readability aspect refers more to the existence of a serialization format, rather than to the level of formality. A number of languages have textual serialization formats that support process description interchange. EDOC and UML2 (and OORAM) use XMI, BPEL4WS, BPML, ebXML and XPDL use XML, and so forth. Further, while a few languages have been noted to support quantitative analyses, the type of analyses that are supported can vary widely. The value “possibly” refers to the fact that a given model may be transformed into a simplified model that supports some kinds of analyses. For example, researchers have often tried to extract Petri nets or finite state machines from dynamic models to perform analyses. We can not say that UML’s dynamic model supports deadlock detection, but if we extract and merge state machines from statecharts of various classes, we might get a state machine model that supports things such as deadlock detection, reachability analysis, and the like.

7.2 A framework for language selection

Rather than picking a winner, we propose guidelines that, when followed, should maximize the benefits that one can draw from process models.

What for. The first question that one needs to answer is the intended use of the resulting process models. This will be the most determinant factor in picking a language. Table 2 above should be of some help, although the evaluation should be more refined

Go with the language(s) that most people use. Clearly, there is a great value in picking a commonly used language. This guarantees several things:

1) maximizes the chances for process description interchange;
2) most practical research tends to be focused on such languages;
3) the language will keep evolving, incorporating the state of the art and practice of the field—with some inevitable lag.

Identifying such languages is not easy in new domains with several competing standards.

Go with the language with the most extensibility. We see two kinds of extensibility:

1) meta-model extensibility: the language enables the addition of new metamodel constructs;
2) hooks: the meta-model constructs provide hooks that enable users and tool vendors to incorporate semantic information.

The first kind is clearly more powerful. However, it is risky, and not all languages support metamodel extension. The second kind of extensibility only requires that the language makes it possible to attach textual descriptions to entities of different granularity. This enables us to attach structured semantic information to model constructs without complicating the metamodel. Such information will be ignored by tools that do not support the capability. For example, we could add Petri net models to describe individual operations within a functional model, or attach finite state machines to resources in REA diagrams.

Merge and view. It is desirable to use the same language to support the tasks described in Table 2. When no single language fits the bill, we should mix and match languages using a "host" language as the modeling core, and attaching different semantic hooks from other languages that can be extracted and collated into their own models to support a given task.

The premise of this paper was that business process modeling was important for eliciting software requirements. If software is the end product, it may be a good idea to use UML as a host language, and add views from other languages that may be extracted and processed on demand.

8 Bibliography

[v.d. Aalst, 1999] W.M.P. van der Aalst, "Formalization and Verification of Event-driven Process Chains," *Information and Software technology*, vol. 41 no. 10, 1999, pp. 639-650.

[Andrews et al., 2003] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Aivana Trickovic, Sanjiva Weerawarana, Business Process Execution Language for Web Services (BPEL4WS) version 1.1 , 5 May 2003.

[Bolognesi & Brinksam, 1987] T. Bolognesi and E. Brinksma. "Introduction to the ISO specification language LOTOS". *Computer Networks and ISDN Systems*, vol. 14, pp. 25-59, 1987.

[BPMI,2003] *Business Process Modeling Language*, Business Process Management Institute, January 24, 2003, 96 pages

[Brooks, 1987] Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer* 20, 4 (April 1987): 10-19

[Carlson, 1979] W.M. Carlson, "Business Information Analysis and Integration Technique (BIAIT) - the new horizon." Data Base, 10 (4), p. 3-9, Spring 1979.

[Coad & Yourdon, 1988] P. Coad & E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, First edition, 1989.

[Curtis et al., 1992] Curtis, B., Kellner, M.I. and Over, J.: "Process modeling", *Communications ACM*, 35, (9), pp. 75- 90, 1992

[Dayal et al., 2001] U. Dayal, M. Hsu, and R. Ladin, "Business Process Coordination: State of the Art, Trends and Open Issues," in *Proceedings of the 27th Very Large Databases Conference*, VLDB 2001, Roma, Italy, pp. 3-13.

[Dogac et al., 2002] Asuman Dogac, Yusuf Tambag, Pinar Pembecioglu, Sait Pektas, Gocke Laleci, Gokhan Kurt, Serkan Toprak, and Yildiray Kabak, "An ebXML Infrastructure Implementation through UDDI Registries and RosettaNet PIPs, in *proceedings of ACM SIGMOD 2002*, June 4-6, Madison-Wisconsin, pp. 512-523.

[Dussart et al., 2002] Aymeric Dussart, Benoit Aubert & Michel Patry, "An Evaluation of Inter-Organizational Workflow Modelling Formalisms," technical report 2002s-64, CIRANO (http://www.cirano.qc.ca) , July 2002.

[ebXML,2003] see http://www.ebxml.org

[Eertink et al., 1999] H. Eertink, W. Janssen, P.A. Luttighuis, W. Teeuw, and C. Vissers. "A business process design language". In *FM'99 --- Formal Methods: World Congress on Formal Methods in the Development of Computing Systems*, pp. 76--95. Springer-Verlag, LNCS-1119, 1999.

[Fernandez et al., 1996] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. "CADP: A protocol validation and verification toolbox". In *Computer Aided Verification*, pp. 437--440. Springer-Verlag, LNCS-1102, 1996.

[Gale & Eldred, 1996] Thornton Gale and James Eldred, *Getting Results with the Object-Oriented Enterprise Model*, SIGS Boooks, ISBN 1-884842-16-X, January 1996.

[Geerts & McCarthy, 1997] Geerts, Guido L. & William E. McCarthy, "Modeling Business Enterprises as Value-Added Process Hierarchies with Resource-Event- Agent Object Templates," in *Business Object Design and Implementation* J. Sutherland and D. Patel (eds.), 1997, Springer-Verlag, pp. 94-113

[Glikas & Valiris, 1999] Michalis Glykas and George Valiris, "Formal methods in object-oriented business modelling," *The Journal of Systems and Software*, vol. 48 (1999), pp. 27-41.

[Gruhn & Wellen, 2001] Volker Gruhn and Ursula Wellen, "Analyzing a process landscape by simulation," *The Journal of Systems and Software*, vol. 59 (2001), pp. 333-342.

[Hammer, 1990] M. Hammer, "Re-engineering work: don't automate, obliterate", *Harvard Business Review*, July-August 1990, pp. 104-112.

[Hammer & Champy, 1993] Hammer M., and Champy, J., *Reengineering the Corporation*, Harper Business, 1993, New York.

[Haughen & McCarthy,2000] Haugen, R & McCarthy W.E., "REA, a semantic model for Internet supply chain collaboration", OOPSLA Workshop on *Business Object Components: Enterprise Application Integration*, OOPSLA 2000, Minneapolis, Minnesota.

[Henderson-Sellers & Edwards, 1990] Brian Henderson-Sellers and Julian M. Edwards, "The Object-Oriented Systems Life Cycle," *Communications of the ACM*, vol. 33, no. 9, September 1990, pp. *143*-159

[Ijiri, 1975] Ijiri, Y., *The Theory of Accounting Measurement*, American Accounting Association, Sarasota, Florida, 1975.

[Isoda, 2001] Sadahiro Isoda, "Object-Oriented real-world modeling revisited," *Journal of Systems and Software*, vol. 59 (2001), pp. 153-162.

[Jackson & Twaddle, 1997] Michael Jackson and Graham Twadlle, *Business Process Implementation: Building Workflow Systems*, Addison Wesley, 1997, ISBN 0-201-177684.

[Janssen et al., 1998] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld. "Verifying business processes using SPIN". In G. Holzman, E. Najm, and A. Serhrouchni, editors, *Proceedings of the 4th International SPIN Workshop*, pp. 21--36, Paris, France, 1998.

[Keller et al., 1992] Keller, G; Nüttgens, M, Scheer, A.-W, "Semantische Prozebmodellierung auf der Grundlage, publication of the Institut für Wirtschaftsinformatik, paper 89, Saarbrucken, 1992.

[Kindler, 2003] Kindler, E., "On the semantics of EPCs: A framework for resolving the vicious circle" technical report, Reihe Informatik, University of Paderborn, Paderborn, Germany, August 2003.

[Lee et al., 1996] J.Lee, M.Gruninger,Y.Jin, T.Malone, A.Tate, G. Yost, and other members of the PIF Working Group, *The PIF Process Interchange Format and Framework (May 24,1996)*,1996, http://ccs.mit.edu/pif8.html.

[McCarthy, 1980] William E. McCarthy, "Construction and Use of Integrated Accounting Systems with Entity-Relationship Modeling", in P. Chen eds., *Entity-Relationship Approach to Systems Analysis and Design*, North Holland Publishing, 1980, pp. 625-637.

[McCarthy, 1982] William E. McCarthy. "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* (July 1982) pp. 554-78

[Malone et al.,1999] T.W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C.S. Osborn, A. Bernstein, G. Herman, M. Klein, and E.O'Donnell, "Tools for inventing organizations: Toward a handbook of organizational processes," *Management Science* 45(3) pp 425-443, March, 1999.

[Mayer et al., 1995] Mayer, R.J., C. P. Menzel, M.K. Painter, P.S. deWitte, T. Blinn, and B. Parakath, *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*, Knowledge Based Systems Inc., September 1995.

[NIST,2002] *The Process Specification Language* (PSL 2.0), September 2002, NIST, http://ats.nist.gov/psl/

[OASIS, 2001] Business Process and Business Information Analysis Overview v1.0 (ebXML), May 11, 2001, http://www.ebxml.org/specs/bpOVER.pdf

[OMG,2001] OMG, *EDOC: UML Profile for Enterprise Distributed Object Computin*g, Document ptc/2001-12-04, December 2001

[Ould & Roberts, 1987] M.A. Ould & C. Roberts, "Defining Formal Models of the Software Development Process," in *Software Engineering Environments*, ed P. Brereton, Ellis Horwood, 1987.

[Ould, 1995] M.A. Ould, *Business Processes: Modelling and Analysis for Re-engineering and Improvement*, 1995, John Wiley & Sons.

[Peterson, 1981] J. L. Peterson, *Petri Nets Theory and the Modeling of Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[Phalp, 1998] Keith Phalp, "The CAP framework for business process modeling," *Information and Software Technology*, vol. 40 (1998) , pp. 731-744.

[Phalp & Shepperd,2000] Keith Phalp and Martin Shepperd, "Quantitative analysis of static models of processes," *The Journal of Systems and Software*, vol. 52, pp. 105-112, 2000.

[Peltz, 2003] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, no. 10, pp. 46-52, October 2003.

[Reenskaug, 1996] Trygve Reenskaug, *Working with Objects: The OORAM Software Engineering Methodology*, Manning, 1996.

[RosettaNet,2003] http://www.rosettanet.org

[Ross,1977] Ross, D., and K. Schoman, "Structured Analysis for Requirements Definition," *IEEE Trans. Software Engineering*, vol. 3, no. 1, January 1977.

[Turban et al., 1999] E. Turban, J. Lee, D. King, et H.M. Chung, *Electronic Commerce: A Managerial Perspective*, Prentice-Hall, 1999.

[Yourdon,1989] Yourdon, E. N., *Modern Structured Analysis*, Prentice-Hall, 1990.

[Yu, 1976] Yu, S. C., *The Structure of Accounting Theory*, The University Presses of Florida, 1976.

[WFMC,1995] Workflow Management Coalition, *The Workflow Reference Model*, document no TC00-1003, January 1995, see http://www.wfmc.org

[WfMC,1998] Workflow Management Coalition, *The Workflow Management Application Programming Interface Specification*, WFMC-TC-1009 - V 2.0, July 1998.

[WFMC,1999a] Workflow Management Coalition, *Interface 1: Process Definition Interchange Process Model*. WfMC-TC-1016-P, Version 1.1, October 1999.

[WFMC,1999b] Workflow Management Coalition, *Terminology and Glossary*,document number WFMC-TC-1011, Feb. 1999.

[WfMC,2002a] "An introduction to workflow," in *The Workflow Handbook 2002*, ed. Layna Fischer, the Workflow Management Coalition (2002), ISBN 0-9703509-2-9,

[WfMC,2002b] The Workflow Management Coalition, *Workflow Process Definition Interface -- XML Process Definition Language*, October 2002, document number TC-1025.