Data4BPM, Part 1:

Introducing Business Entities and the Business Entity Definition Language (BEDL)

A first class representation of data for BPM applications

Prabir Nandi
Research Staff Member
IBM Research, Hawthorne, NY

Dieter König
Senior Technical Staff Member
IBM Research & Development, Boeblingen, Germany

Simon Moser
WebSphere BPM Architect
IBM Research & Development, Boeblingen, Germany

Rick Hull
Research Manager
IBM Research, Hawthorne, NY

Vlad Klicnik
Senior Technical Staff Member, BPM Tools
IBM Software Labs, Markham, Ontario

Shane Claussen
Senior Technical Staff Member, BPM Architecture
IBM Software Labs, Foster City, CA

Matthias Kloppmann
Distinguished Engineer, BPM Technology
IBM Research & Development, Boeblingen, Germany

John Vergo
Senior Manager and Strategist, Business Architecture
IBM Research, Hawthorne, NY

April 21, 2010

In this article series, experts from IBM lay out a technical vision and approach for how data can be represented and specified in a first-class way for BPM applications. In this series, to be published over the next several months, the authors will present various facets of this vision, including:

- Proposing new specification standards and positioning them with existing standards like WS-BPEL and BPMN.
- Presenting process modeling use cases and scenarios enabled by this new architecture.
- Ideas for implementing the architecture with the IBM BPM stack.

Part 1 of this series introduces the concept of *Business Entities* as a means of representing the business view of data. It proposes two new standards, the Business Entity Definition Language (BEDL) and BPEL4Data, an extension to WS-BPEL for the holistic design and execution of process with Business Entities. Part 1 covers BEDL in detail

Part 2 will cover the BPEL4Data language elements in depth, and discuss the architecture that brings together the BPEL family of languages (WS-BPEL, WS-HumanTask) with BEDL in execution scenarios.

Part 3 will cover process modeling scenarios and patterns with Business Entities using Websphere Business Modeler notations and semantics, and will use a complex scenario to demonstrate the modeling patterns.

In Part 4 we'll turn to the execution side and look at how the holistic model of processes and Business Entities extends design-to-deploy with additional semantics for a richer interactive process design experience.

Introduction

The specification and deployment of business processes and operations is crucial to the successful management of medium or large-scale enterprises. In most business process management tool suites, data is treated mostly as an afterthought. Activities and their flows are the main abstractions and the data manipulated by the processes is essentially hidden in process variables. The presentation and aggregation of data is handled outside of the process definition, and implemented through generic service calls. This process-only approach ignores the important *data* perspective during business operation analysis, often obscures key aspects of the operations, and can lead to costly re-factoring throughout the solution lifecycle [25]. Over the past decade, a new approach to business process and operations modeling has emerged [2, 3, 10, 11, 16] , that is based on *Business Entities*[1]. Business Entities provide a new basis for specifying business operations that combines data and process at a fundamental level. While the approach has proven successful in several contexts [6, 7, 15, 17] its application has taken the approach of creating a Business Entity layer on top of existing SOA and middleware tools. In contrast, this article introduces a way to take advantage of the Business Entity approach while still using and building upon standards such as WS-BPEL [19] and BPMN [20]. This enables the use of Business Entities in conjunction with the large industrial investment in, and vast embedded base of, tooling for these process-centric approaches. This article introduces a new proposed standard, called *Business Entity Definition Language (BEDL)*, and describes how you can use it alongside process-centric technologies such as WS-BPEL and BPMN.

Business Entities (*BEs*) are key business-relevant dynamic conceptual objects that are created, evolved, and (typically) archived as they pass through the operations of an enterprise. A Business Entity includes both an *information model* for data about the business objects during their lifetime, and a *lifecycle model*, which describes the possible ways and timings that tasks can be invoked and performed on these objects. An example of a Business Entity type is Courier Shipment, whose information model would include attributes for package ID, sender, recipient, shipping method, arrival times, delivery time, and billing information. The lifecycle model would include the multiple ways that the package could be delivered and paid for, and would be used in tracking each instance of the Courier Shipment Business Entity type. Other examples of BEs are Claim in an Insurance Claims process, going through the states of Filed, Approved, Fulfilled, and so on; Trouble Ticket for a Services Delivery process, going through the lifecycle states of Opened, Assigned, Rejected; financial Deal in a loan-giving organization, going through the lifecycle states of Draft, Offered, Signed, Active, and so on.

In much of the literature to date on Business Entities, the lifecycle models are specified using finite state machines, where each state of the machine corresponds intuitively to a business-relevant *milestone*, or operational objective, that might be achieved by a Business Entity instance. Business Entities define a useful way to understand and track business operations, such as the locations that the package has passed through and its arrival times, and the distribution of timings (for example, how many two-day air shipments took longer than two days in the last week) and ways of handling (for example, what percentage of cash-on-delivery shipments required more than one delivery attempt), which are useful for

[1] The basic notion of *business entities* has also been termed *business artifacts* [2], *adaptive business objects* [16], and *adaptive documents* [3]. The basic notion is also described in [30].

monitoring, dashboards, and more broadly, business intelligence. More generally, Business Entity types can provide a unifying basis for understanding many aspects around the operations of an enterprise, including requirements gathering, business rules, compliance, and process user interactions [15, 14, 32]. Because business operations models based on Business Entities can be implemented in a reliable manner (for example, [11, 33]) this approach opens the door to the development of a variety of BPM tools that are focused on the common basis of Business Entities.

In order to bring the advantages of the Business Entities perspective into the existing world of WS-BPEL, BPMN, and similar process-centered technologies, this article introduces the *BEDL variant* of the Business Entity notion. This variant focuses on four essential aspects of Business Entities: information model, (macro-level) lifecycle model, access policies based on role and lifecycle state, and notifications of state and data change events. Unlike much of the existing literature and practical applications of Business Entities to date, the BEDL variant does not include mechanisms to specify the detailed processing steps involved in a Business Entity lifecycle model; these can be specified in a separate but integrated fashion using, for example, WS-BPEL or BPMN. More specifically, we will describe an approach for specifying a complete *Business Operations Model* (*BOM*) using a BEDL specification that specifies the relevant Business Entity types, in combination with a WS-BPEL (or BPMN) specification that specifies the various processing steps used in conjunction with those Business Entity types. A runtime implementation of such BOMs can be supported using software components dedicated to implementing the BEDL specification and a conventional WS-BPEL (or BPMN) engine. The components supporting the BEDL specification will:

- Provide a repository for storing and maintaining Business Entity instances as they evolve during the operations of an enterprise.
- Enforce constraints concerning how these instances can move from state to state.
- Enforce constraints concerning what kinds of (human or automated) actors can access these instances.
- Support a pub-sub capability, whereby processes can subscribe to be alerted when state transitions and data changes occurring in the BEDL component.

The focus of the BEDL variant of Business Entities on the macro-lifecycle and constraints is similar to the use of Business Entities to facilitate service interoperation as described in [34].

The rest of this article focuses on using WS-BPEL in connection with BEDL, but this approach can also be used with BPMN and other process specification languages.

The Business Entities approach is closely related to the *Case Management* [35, 36] and *Document Engineering* [37] approaches. In both of these approaches, the data being managed to support enterprise operations is at the core of the underlying conceptual models, and key conceptual objects (*cases* in the first approach and *documents* in the second) evolve as they move through the enterprise operations. Further, those approaches are typically applied in contexts where there is very strong human influence on which tasks are performed and how they are sequenced. Business Entities embody the spirit of both of these approaches, but have been applied in a variety of application areas (such as banking, finance, supply chain) that have traditionally used process-centric approaches, in which the IT infrastructure typically controls which tasks are performed and in what sequence. Speaking in broad terms, the Business Entity approach can help to support the spectrum of contexts ranging from the largely human-driven to the largely process-driven.

This article is organized as follows.

- A meta-model for a standard based on Business Entities presents the conceptual meta-model for the BEDL variant of Business Entities, and describes how BEDL specifications can be coordinated with WS-BPEL and other process-centric specifications.
- Business Entity Definition Language (BEDL) presents the BEDL meta-model and syntax; briefly discusses possible runtime architecture to support BEDL specifications; and overviews BPEL4Data, an approach for extending the WS-BPEL standard to support explicit interoperation with BEDL components. (Part 2 of this series will provide a detailed specification of BPEL4Data, an extension of WS-BPEL to work with BEDL.)
- Advantages of using Business Entities describes the business value of the Business Entity approach.
- Design methods using Business Entities discusses design methods and guidelines that can be used in connection with BEDL.

A meta-model for a standard based on Business Entities

This section provides a detailed introduction to the key elements of BEDL and describes how you can use it to specify and deploy business processes and operations. We use a simple scenario based on managing shipments by an express courier, and describe how BEDL can be combined with a process-centric standard such as WS-BPEL or BPMN to provide a fully specified Business Operations Model (BOM). For a description of the basic design methods for creating entity-centric BOMs, see Design methods using Business Entities.

The Courier Shipment scenario used here focuses on the Business Entity type used to manage shipments end-to-end, from the initial receipt or pick-up by the courier, through shipping and billing, and delivery to the recipient. We'll also touch on how you might design Business Entity types for truck trips, plane trips, customer contacts and a customer loyalty program.

Business Entity types in BEDL introduces BEDL Business Entity types, and Specifying full Business Operations Models using BEDL describes how they can be used in conjunction with processes and other BE types to create a full BOM.

Business Entity types in BEDL

Business Entity types in BEDL have four main components: information model, lifecycle model, access policies and notifications. Figure 1 shows the four components of the Courier Shipment BE type.

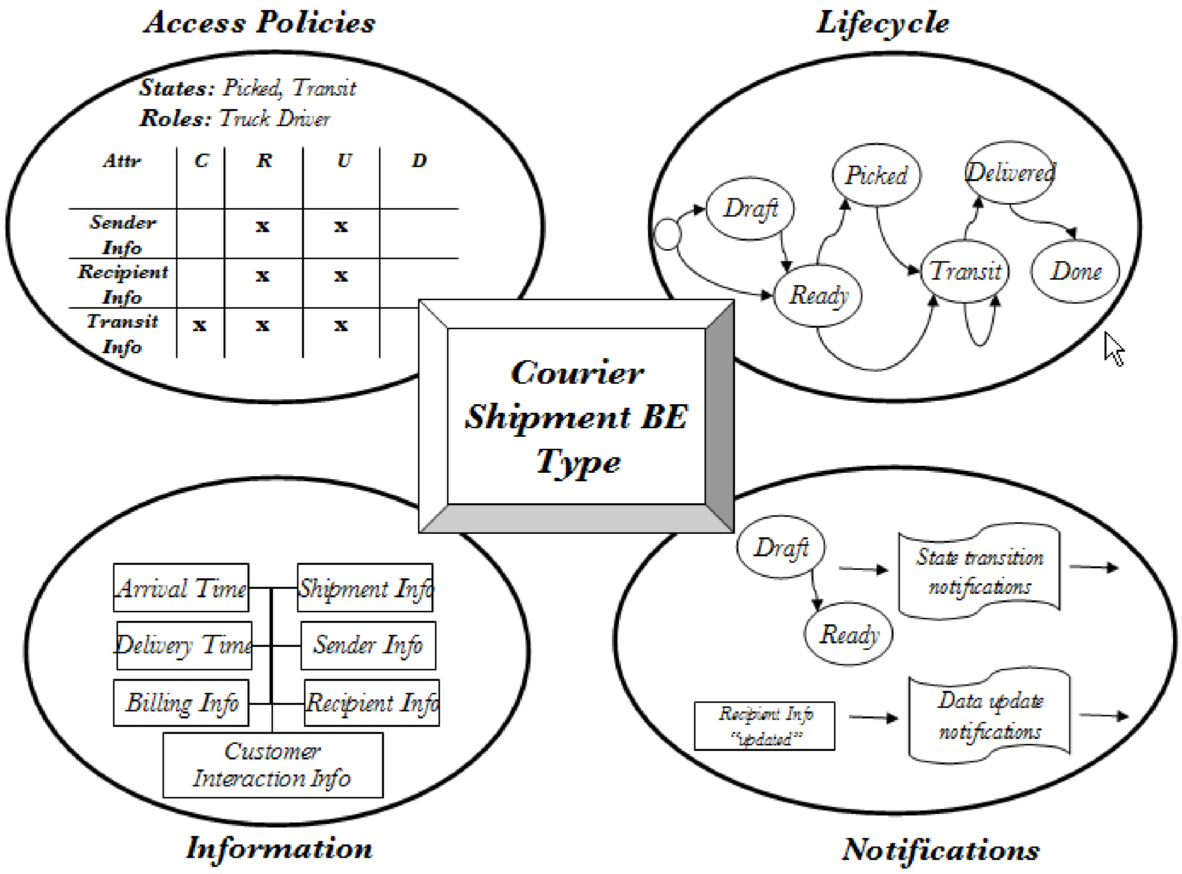

Figure 1. The Courier Shipment Business Entity type

In BEDL, the information model of BE types is specified as a family of *attribute/value pairs*, where an XML schema is associated with each attribute, that governs the structure of the corresponding values. Figure 1 shows some of the attributes of the information model of Courier Shipment. Some of these have a simple structure (such as Arrival Time and Delivery Time), and others have a richer structure (such as Billing Info, Transit Info, Sender Info, Recipient Info, and Customer Interaction Info). Additional attributes might be included.

In many cases, the data populating a given attribute is stored entirely by the software component implementing a BEDL specification. In some cases, however, an attribute might explicitly hold some of its data, and hold references to other data. For example, in the Sender Info, if the sender is a regular customer with an account, perhaps only the sender ID will be held as part of a Courier Shipment BE instance. However, references should be used with care. For example, if some properties of the Sender may be dynamic, such as billing address, and the history of those properties will not be retained by the Customer database, it might be important to store an explicit copy of such values in the BE instance.

In general, when a BE instance is first created, only a fraction of its attributes have values. As the BE instance moves through the enterprise, some attribute values are updated and others are populated. In some cases an attribute might be populated for some but not all of the BE instances of the type. For example, the attribute for Customer Interaction Info might remain un-initialized for shipments by senders who do not contact the shipper after dropping off the package.

In BEDL, the lifecycle model for Business Entities is specified as a finite state machine. In the lifecycle model for the Courier Shipment BE type, there are six states along with a unique `initial state`, which is present in all lifecycle models. As shown in Figure 1, a Courier Shipment BE instance can move from the initial state to `Draft` or to the state Ready. Intuitively, the BE instance can be in the `Ready` state if the Sender Info and Recipient Info, along with a plan for payment are recorded. The `Draft` state corresponds to the case where a Courier Shipment is initiated, but there is some delay in getting all of the information needed before moving into the `Ready` state. The other states and transitions in the lifecycle for the Courier Shipment type are largely self-explanatory. The package might have been brought into a shipping office, in which case the corresponding BE instance will move directly into the `Transit` state. Otherwise, it will move into the `Picked` state when it has been picked up from the sender, and then into the `Transit` state.

Although not illustrated here, directed loops of transitions involving multiple states are permitted. Also, in this state machine model, self-loops are implicit. These arise when a process leaves a BE instance in the same state. This might arise with a Courier Shipment instance, for example, when it is in the `Transit` state, and information is included about which truck it is on for what time interval and which holding location it is in for what time interval, or which plane it is on, and so on. A self-loop might also arise when new billing information arrives (for example, if the payment processing was COD or performed on a monthly basis).

In BEDL, the lifecycle specification itself does not provide detail on the activities that might be performed while a Business Entity instance is in a given state, nor the activities that might be performed as part of transitioning a Business Entity instance from one state to another. Rather, a BEDL specification is typically accompanied by a WS-BPEL or other specification that focuses on the processing aspect of an overall BOM. The typical approach for this is described in Specifying full Business Operations Models using BEDL.

A software component supporting BEDL will provide an interface that supports two key capabilities. The first of these enables external processes to access and manipulate the Business Entity instances. In particular, an external process (for example, a WS-BPEL process) can request access to one or more Business Entity instances, request updates to one or more attribute values, and request that the instances change state. The second key capability is support for subscriptions for notifications about changes to Business Entity instances, including both data modifications and state transitions. For both processes and subscriptions, it is assumed that there is an associated *role*, that is, a category of agents, human or automated, that requests the actions. We'll discuss the specification of processes and how they interact with BE instances more in Specifying full Business Operations Models using BEDL.

The third component of a BE type is the specification of *access policies*. There are two focus areas for these policies: *CRUD* in connection with `Creates, Reads, Updates,` and `Deletes` of attribute values in the information model, and `Executions` of state transitions in the lifecycle model. We sometimes refer to the collection of these two types of access policies as *CRUDE* policies.

The CRUD restrictions focus on what roles have authority to modify attribute values. Importantly, these restrictions are keyed not only by attribute and role, but also by the state that a BE instance is in. You can see this in Figure 1, where the CRUD matrix for the role `TruckDriver` is shown for the states `Picked` and `Transit`. Although not illustrated in Figure 1, a CRUD restriction may also include a `guard`, or condition, expressed in XPath, based on the attribute values of the BE instance. For example, a `TruckDriver` might be permitted to update the `Sender Info` only if the `Transit Info` indicates that the truck is schedule to go, or has gone, to the sender's address. The CRUD restrictions are enforced in connection with processes that attempt to access BE instance attributes, and also in connection with notifications in response to subscriptions.

Similar to the CRUD policies, the Execution policies involve two components. First, an Execution policy can specify which transitions can be invoked by a given role. Second, similar to a CRUD policy, an Execution policy may include a *`guard`*, or condition, that must be satisfied in order to transition from one state to another. For example, it may be specified that a Courier Shipment instance cannot move to the `Ready` state unless the Sender Info and Recipient Info are populated.

In the current BEDL meta-model, the lifecycle models do not support parallelism at the level of the states and state transitions. In our experience, much of the parallelism that arises in practice can be accommodated by the processes executing while a BE instance is within a single state. For example, as mentioned above, even though different processes that make up the Billing activity for a Courier Shipment instance might be happening in parallel with the lifecycle, the individual processes (such as perform cash-on-delivery payment collection, or post a monthly payment against a given Courier Shipment) can be done while the instance is in a single state.

The last part of specifying a BE are the *notifications.* Basically, these are the noteworthy CRUDE activities happening on the BE that external parties, such as other processes, might want to know about. External parties will subscribe to these events through the BEDL interface. In our example, the transition to the `Ready` state could be a noteworthy event. A process that deals with dispatching trucks for pick-up could be subscribing to this event, so that the pick-up can be scheduled efficiently.

Specifying full Business Operations Models using BEDL

Now let's look at the recommended way to use BEDL in conjunction with a process specification language in order to specify and deploy a full Business Operations Model (BOM). Although any process specification language can be used in conjunction with BEDL, we focus here on the use of WS-BPEL. For more information on the method for discovering or identifying BE types, see Design methods using Business Entities.

Figure 2 shows a high-level illustration of how BEDL and WS-BPEL specifications can work together.

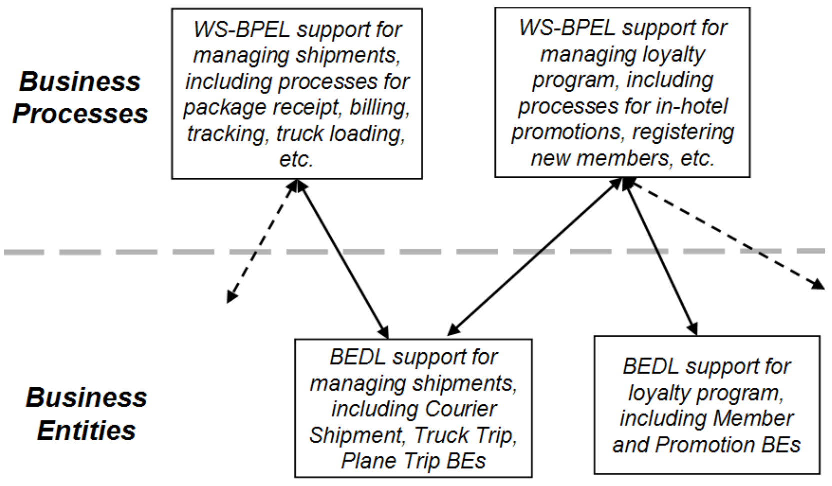
Business Processes
WS-BPEL support for managing shipments, including processes for package receipt, billing, tracking, truck loading, etc.
WS-BPEL support for managing loyalty program, including processes for in-hotel promotions, registering new members, etc.
Business Entities
BEDL support for managing shipments, including Courier Shipment, Truck Trip, Plane Trip BEs
BEDL support for loyalty program, including Member and Promotion BEs

Figure 2. High-level depiction of the interaction between Business Entity specifications in BEDL

The box in the lower left corresponds to the BEDL specification for a closely related family of BE types, including Courier Shipment, Truck Trip, and Plane Trip. Here the Truck Trip BE type is used to manage individual trips made by a truck, including the packages it is loaded with, the itinerary, the drop-offs and pick-ups made, and so on; and similarly for Plane Trip. The box in the upper left corresponds to the family of WS-BPEL specifications that perform the processes associated with managing the shipments, including the truck trips and plane trips. The boxes in the lower and upper right correspond respectively to additional Business Entity types and processes associated with managing the customer loyalty program, which is not described here. The two BEs (bottom two boxes) might be supported by the same executing software components or by distinct executing software components.

The two business processes (top two boxes) hold families of WS-BPEL specifications. Both boxes will hold a number of individual WS-BPEL specifications, each one corresponding to a single *process*. Each process may include one or more *atomic activities*. (In some cases, multiple atomic activities might be combined to form a composite *flow activity*, but that is not germane to the current discussion.) Some of the atomic activities will involve the BEDL run-time components. The syntax for these atomic activities is based on an extension of WS-BPEL, and will be described in detail Part 2.

A software component supporting BEDL provides an interface that enables processes in external components to invoke the following capabilities:

1. **Request metadata about BE types:** This enables a process to query the BEDL component about the schemas of the BE types that it is supporting; for example, any runtime metadata like the current set of readable/writeable attributes.
2. **Request to access data from a Business Entity instance**: This enables a process to perform attribute creates (and appends), reads, updates, and deletes (and removes). If the requested access violates the access policies, an `authorizationFailure` fault is thrown.
3. **Request to query on BE instances:** This enables a process get the references to all BE instances of a given type that satisfy a query.
4. **Request to execute a transition of a Business Entity instance from one state to another**: This enables a process to move a Business Entity from one state to another. This includes the creation of a BE instance, that is, moving it from the initial state to another state. If the state change request violates the given BE lifecycle, an `invalidState` fault is thrown. If an access policy concerning guards is violated, an `inconsistentData` fault is thrown.
5. **Request to lock (parts of) a Business Entity instance:** This is used in connection with concurrency control for BE instances. If the BE instance is not available (for example, because "hard" locks are being used and another process already has a conflicting lock), then a `lockFailure` fault is thrown.
6. **Release lock on (parts of) a Business Entity instance:** Also used for concurrency control.
7. **Subscribe to data or state changes of a class of Business Entity instances:** This enables a process to subscribe to notifications about data and state changes in BE instances. Since the access policies can restrict read access to attributes based on role and current state, the subscription request may be partially or fully restricted. In the former case, a warning may be returned, and in the latter case an `authorizationFailure` fault is thrown. Receipt of such notifications may trigger new instances of a process to be initiated, "fast forward" or terminate running process instances, and launch compensation activities. The process may use the WS-BPEL `pick` construct to wait for the notifications associated with a subscription.

In a typical interaction between a WS-BPEL process and a BEDL component, a process will request to lock parts of one or more BE instances, access and possibly modify some of the attribute values held by the instance, possibly request a state transition, and then release the locks on the BE instances. In particular, if a process changes the state of a BE instance, this is the last thing to be done before releasing the lock. Thus, a new state is reached when one of the processes executes and achieves some business-relevant milestone. The BE instance may remain in that state as subsequent processes are executed. Eventually a process will execute and again advance the BE instance to a new state. Note that the discipline of having processes make state transitions only as they conclude is a guideline, but not a requirement. BEDL does not prohibit a process from making one or more state transitions, and following each one with additional updates to the information model.

You can achieve synchronization between BE instances by using processes that manipulate two or more BE instances at the same time. For example, a process associated with the customer loyalty program in the Courier Shipment scenario might request locks on both the BE instance for a Courier Shipment and the (loyalty program) Member BE instance associated with the sender of the shipment. The process might then record a discount on the Courier Shipment instance and record a credit on the Member instance, and then release both instances.

The BEDL framework can support four approaches to concurrency control. These approaches are based on the notions of read and write locks, as typically found in two-phase locking regimes used in database concurrency control regimes. The read and write locks are also made in connection with the current state of a BE instance. This is achieved by acting as if there is a designated *state attribute* in the BE type information model. The approaches described below are based on two dimensions of variation. These dimensions are:

- **Pessimistic vs. optimistic:** In the pessimistic (or "hard locks") approach, a process is given a lock on a Business Entity if another process already has a conflicting lock on it. In the optimistic (or "soft locks") approach, multiple processes might hold conflicting locks, and some forms of warning or compensation may need to be performed when conflicts arise.
- **Explicit vs. implicit:** In the explicit case, processes explicitly specify which attributes of a BE instance are to be read-locked or write-locked. In the implicit case, the read and write locks are inferred from the CRUD portion of the access policies, along with the current state of the BE instance and the role that invoked the process.

The current proposal for BEDL uses the optimistic, implicit approach to concurrency control. Thus, each "request to lock" (item 5) will succeed, but may receive a warning that the requested lock is in conflict with some already established lock. Further, the process that holds the conflicting lock may also receive a warning. In such situations, it may be desirable for processes to include logic for handling the exceptional situations where modifications to BE instances by processes that hold conflicting locks might lead to unwanted outcomes.

Business Entity Definition Language (BEDL)

In this section, we'll introduce *Business Entity Definition Language (BEDL),* the XML language to specify BE types. A BEDL specification is typically used to define a small number of closely related BE types. For example, a BEDL specification might be used to define the BE types in the bottom two boxes in Figure 2. There may be business implications and semantics around the *clusters* of BE types in a BEDL specification and the process specifications that work with it. For example, such a cluster can correspond to functional areas or business capabilities, such as Procurement, Shipping, Purchasing, and so on, or these can be carried forward from strategic business decomposition techniques like Component Business Modeling [18].

The BEDL specification of the Courier Shipment BE type describes the BEDL specification of the Courier Shipment BE type. The BEDL meta-model describes the type specification for BEDL specifications. Use of BEDL components revisits the topic of how external software components can interact with BEDL components. Finally BPEL4Data: Extending WS-BPEL for BEDL briefly describes how WS-BPEL is extended to enable graceful interaction between WS-BPEL components and BEDL components.

Note: The BEDL schema and specifications in the following sections is provided for download with this article.

The BEDL specification of the Courier Shipment BE type

Listing 1 shows the BEDL for the `Courier Shipment` BE type that was introduced in A meta-model for a standard based on Business Entities.

Listing 1. BEDL for Courier Shipment BE type

```
<?xml version="1.0" encoding="UTF-8"?>
<businessEntity name="CourierShipmentBill"
           targetNamespace="http://example.com/businessEntities"
           xmlns:be="http://example.com/businessEntities"
           xmlns:inf="http://example.com/information"
                        xmlns="http://www.ibm.com/bedl"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.ibm.com/bedl bedl.xsd">

       <!— pointer to the information model -->
       <informationModel>inf:courierShipmentBill</informationModel>

       <!— set of attributes contributing as primary keys -->
       <primaryKey>
           <keyAttribute>/@inf:shipmentID</keyAttribute>
       </primaryKey >

       <!— lifecycle: valid states and transitions -->
       <lifecycle>
           <states initialState="Start">Start  Draft  Ready  Picked  Transit  Delivered
                      Done</states>
           <transitions>
               <transition name="StartToDraft" fromState="Start" toState="Draft"/>
               <transition name="StartToReady" fromState="Start" toState="Ready"/>
               <transition name="DraftToReady" fromState="Created" toState="Ready"/>
               <transition name="ReadyToPicked" fromState="Ready" toState="Picked"/>
               <transition name="ReadyToTransit" fromState="Ready" toState="Transit"/>
               <transition name="PickedToTransit" fromState="Picked" toState="Transit"/>
```

```
            <transition name="TransitToDelivered" fromState="Transit"
        toState="Delivered"/>
            <transition name="DeliveredToDone" fromState="Delivered" toState="Done"/>
        </transitions>
    </lifecycle>

    <!— the CRUDE access policies -->
    <accessPolicies>
        <!-- mandatory definition of roles -->
        <roles>OriginatingCustomer PickupDriver TransitDriver DistributionCenter
        ReceivingCustomer</roles>
        <!-- one or more access policy elements, containing one or more roles and zero
                    or more states -->
        <accessPolicy name="CustomerBillCreation" roles="OriginatingCustomer"
        states="Created">
            <accessEntry dataOperations="update">
                <!-- zero or more attributes -->
                <attributeOrTransition>/inf:recipientInformation</attributeOrTransition>
            </accessEntry>
        </accessPolicy>
        <accessPolicy name="BillReadyTransition" roles="OriginatingCustomer">
            <accessEntry dataOperations="execute">
                <!-- if the data operation is "execute" then expect a transition -->
                <attributeOrTransition>CreatedToReady</attributeOrTransition>
                <!-- The condition is effectively a transition guard  -->
                <condition expression=”fn:exists(/@inf:recipientInformation)” />
            </accessEntry>
        </accessPolicy>
        <accessPolicy name="InTransitPlatinum" roles="OriginatingCustomer"
                    states="Transit">
            <accessEntry dataOperations="read update">
               <!-- zero or more attributes -->
               <attributeOrTransition>/inf:recipientInformation</attributeOrTransition>
               <attributeOrTransition>/inf:billingInformation</attributeOrTransition>
               <!-- zero or one condition -->
               <condition expression=”/@inf:customerType="Platinum Customer"”/>
            </accessEntry>
        </accessPolicy>
        <accessPolicy name="InTransitBasic" roles="OriginatingCustomer"
        states="Transit">
            <accessEntry dataOperations="read">
               <attributeOrTransition>/inf:recipientInformation</attributeOrTransition>
               <condition expression=”/@inf:customerType="Basic Customer"”/>
            </accessEntry>
        </accessPolicy>
    </accessPolicies>

    <!— state and data chance notifications -->
    <notifications>
        <stateTransition transition="CreatedToReady"/>
        <dataChange dataOperation="update" attribute="/inf:recipientInformation"/>
    </notifications>
</businessEntity>
```

The BEDL meta-model

Listing 2 shows the basic structure of the BEDL schema. The cardinality of the various elements can be interpreted as follows:

? - zero or one occurrence
***** - zero or more occurrences
+ - one or more occurrences
no specification - exactly one

Listing 2. Basic structure of the BEDL

```
<businessEntity >
     name="NCName"
    <information="QName" ? />
    <primaryKey>
          keyAttribute="xsd:string" *
    </primaryKey> ?

    <lifecycle>
     <states initialState="NCName">NCNameList</states>
      <transitions>
         <transition
           name="NCName"
           fromState="NCName" ?
           toState="NCName"
         <transition /> *
       </transition>
   </lifecycle>

    <accessPolicies>
      <roles>NCNameList</roles>
      <accessPolicy
          roles="NCNameList"
          states="NCNameList"  ?
          <accessEntry>
               dataOperations=" DataOperationList"
               attribute | transition="xsd:NCName" +
               condition="xsd:string" ?
          </accessEntry> *
      </accessPolicy> *
    </accessPolicies>

   <notifications>
       stateTransition | dataChange="xsd:NCName"  +
   </notifcations> ?

</businessEntity>
```

- `businessEntity`: A Business Entity contains a `lifecycle`, an optional reference to an `information model`, and an optional set of `access policies` and `notifications`.
- `Information`: A reference to an XML schema. The schema will have attributes and other child elements with the `Business Entity` type at the root. Listing 3 shows the information model used for Courier Shipment Bill.

Listing 3. Courier Shipment Bill information model

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:inf="http://example.com/information"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/information" elementFormDefault="qualified"
           attributeFormDefault="unqualified">
           <xsd:element name="courierShipmentBill" type="inf:CourierShipmentBillType"/>
           <xsd:complexType name="CourierShipmentBillType">
               <xsd:sequence>
                       <xsd:element name="arrivalTime" type="xsd:dateTime"/>
                       <xsd:element name="deliveryTime" type="xsd:dateTime"/>
                       <xsd:element name="billingInformation" type="xsd:string"/>
                       <xsd:element name="shipmentInformation" type="xsd:string"/>
                       <xsd:element name="senderInformation" type="inf:tPerson"/>
                       <xsd:element name="recipientInformation" type="inf:tPerson"/>
                       <xsd:element name="customerInteractionInformation"
           type="inf:tInteractionsList"/>
               </xsd:sequence>
               <xsd:attribute name="shipmentID" type="xsd:string"/>
               <xsd:attribute name="customerType" type="inf:tCustomerType"/>
           </xsd:complexType>
```

```
            <xsd:simpleType name="tCustomerType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Platinum Customer"/>
                        <xsd:enumeration value="Basic Customer"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:complexType name="tPerson">
                <xsd:sequence>
                        <xsd:element name="name" type="xsd:string"/>
                        <xsd:element name="address" type="inf:tAddress"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="tAddress">
                <xsd:sequence>
                        <xsd:element name="street" type="xsd:string"/>
                        <xsd:element name="city" type="xsd:string"/>
                        <xsd:element name="zip" type="xsd:integer"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="tInteractionsList">
                <xsd:list>
                        <xsd:element name="interactionItem" type="inf:tInteractionItem"/>
                </xsd:list>
            </xsd:complexType>
            <xsd:complexType name="tInteractionItem">
                <xsd:sequence>
                        <xsd:element name="interactionTime" type="xsd:dateTime"/>
                        <xsd:element name="description" type="xsd:string"/>
                        <xsd:element name="resolution" type="xsd:string"/>
                        <xsd:element name="perceivedCustSatisfaction" type="xsd:integer"/>
                </xsd:sequence>
            </xsd:complexType>
</xsd:schema>
```

- `primaryKey:` One or more attributes in the information model can be assigned to the `primaryKey` list for the BE. Together this attribute or set of attributes forms a unique key for the BE type.
- `lifecycle:` Lifecycle is represented by a set of `states` and `transition` elements:
  - `States:` **A** flat list of state names with one designated as the `initial state`. When a BE instance is created, it transitions from the designated initial state into one of the states reachable in one step from the initial state.
  - `Transition:` A `transition` has a `name`, an optional `fromState` and a `toState`. If the `fromState` is unspecified, this indicates that a transition to the `toState` is allowable from any state. Note that this is purely a convenience feature. However, if there exists a transition for `fromState` to `toState`, in addition to a transition element with just the `toState`, the former will override the latter for all processing purposes.
- `accessPolicies:` An access policy is specified for a set of roles and states. It contains a set of access entries:
  - `accessEntries:` An access entry is a grouping of `dataOperations`, `attributes` or `transitions`, and `conditions`. The `dataOperations` can be one of five types: `create`, `read`, `update`, `delete` and `execute`. In the case of `execute`, the target is a transition that means that this set of transitions is allowed by these roles if the set of conditions are met. Following is an example of an `execute` access policy. In this example, the transition `DraftToReady` is allowed only if executed by the `OriginatingCustomer` role and the condition `Recipient Information` has a

value `fn:exists`. The default expression language syntax to specify conditions is XPath/XQuery.

```
<accessPolicy name="BillReadyTransition" roles="OriginatingCustomer">
     <accessEntry dataOperations="execute">
         <!-- if the data operation is "execute" then expect a transition -->
         <attributeOrTransition>DraftToReady</attributeOrTransition>
          <!-- The condition is effectively a transition guard  -->
          <condition expression= "fn:exists(/@inf:recipientInformation) "/>
     </accessEntry>
</accessPolicy>
```

For all other `dataOperations`, the *access entry target* is a set of *data attributes.* One additional nuance is that only `read` and `update` are applicable for *simple attributes*, whereas for *complex attributes,* all `dataOperations` are applicable.

- `Notifications:` Notifications are used to indicate the intent of Business Entities to broadcast to interested parties, a state or data change. Each notification entry specifies the `transition` or `data operation`. In case of a `data operation`*,* the corresponding `data attribute` can also be optionally specified. In the example below, a notification will be sent whenever the `recipientInformation` is updated or when the Courier Shipment Bill transitions from the `Created` to the `Ready` state.

```
<notifications>
    …..
    <stateTransition transition="CreatedToReady"/>
    <dataChange dataOperation="update" attribute="/inf:recipientInformation"/>
</notifications>
```

Use of BEDL components

The BEDL meta-model is used to specify Business Entity definitions. Each BE definition specifies a single BE type; these can be grouped to specify multiple BE types, as shown in Figure 2.

We expect that BEDL can be used in a variety of contexts, some of which are essentially "green field" (in other words, with little or no restriction to depend on legacy applications for data storage) and others that are "brown field" (with some requirements that depend on legacy applications for data storage). In the green field cases, a family of BE definitions will be managed by the Business Entity runtime. As suggested in Figure 3, a BE runtime can manage the instances associated with multiple BE types. The actual BE instance data (including both attribute values and instance state) is stored in a persistent data store, called BE Content. The BE runtime enforces the access policies, and also sends notifications to processes that have subscribed for events. The database is accessed only by the BE runtime, so proper behavior and evolution of the BE instances is ensured.

In the brown field case, some or most of the data relevant to one or more BE types is already being stored by legacy applications. In these situations, the cost of replicating the data or porting to a new BEDL-controlled data store is prohibitive because of the diverse uses of the data and the numerous applications that are based on the legacy application's interface for accessing the data. In such cases, the information model used by a BE type will not match the data model used by the legacy application, and further, the data model of the legacy application may not hold all of the information appropriate for the BE information model. For example, it would be unusual for a legacy application to maintain information corresponding to the state of a BE type. In some cases, a BE type may span across the data of multiple legacy applications. In such cases, the BE runtime can be augmented with a targeted persistent store that holds linkage data, or data that can be used to link or connect the BE information view of data with the legacy application view of the data. While there are techniques for creating and working with such linkage data, it is still an open research challenge to develop a systematic, effective approach for the design and management of such data.

A group of BE definitions can be applied in most business contexts and scenarios, from individual applications (with a single associated process) to a department cluster (with a few processes*)* to supporting multiple department clusters (with many processes*)* in an enterprise-wide scope. These processes can be of varied types.

Figure 3 illustrates how a BE runtime can interact with multiple types of processes. These processes might be executing in one or more p*rocess execution containers*. Figure 3 shows a single process execution container that includes a BPEL/BPMN [20]-based process, a two-tiered Web application (*process as a page flow*), an off-the-shelf packaged application like SAP [26], a Master Data Management (MDM) maintenance workflow or a XPDL-based [27] application. Essentially, any process type can take advantage of BE definitions in two ways: declaratively via formalized extensions or through the prescribed APIs of the BE runtime (as described in Specifying full Business Operations Models using BEDL). The two-tiered Web application will likely use the APIs, whereas declarative process languages like BPEL, BPMN, XPDL, and so on, will likely "bind" through declarative language extensions.

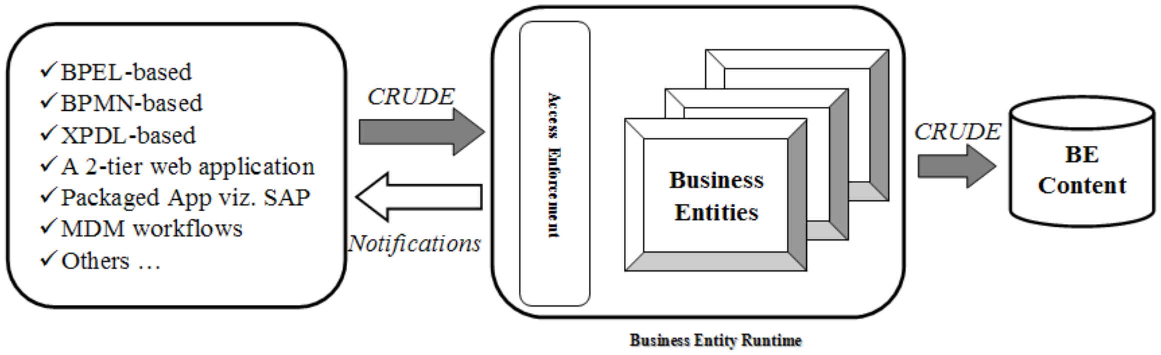

Figure 3. Business Entity definitions and multiple process types

BPEL4Data: Extending WS-BPEL for BEDL

This section briefly introduces, at a conceptual level, BPEL4Data, a *declarative* extension to WS-BPEL that has been developed so that WS-BPEL processes can work easily with BE definitions expressed in BEDL. The complete semantics and specification of BEDL and BPEL4Data will be covered in Part 2 of this series.

BPEL4Data will contain the extensions to WS-BPEL to formally consume BEDL elements. Specifically, it will provide a declarative syntax for annotating a BPEL activity to either specify a BE state change or to indicate BE content manipulation.

As shown in Figure 4, BPEL4Data is the “glue” language between WS-BPEL and BEDL.

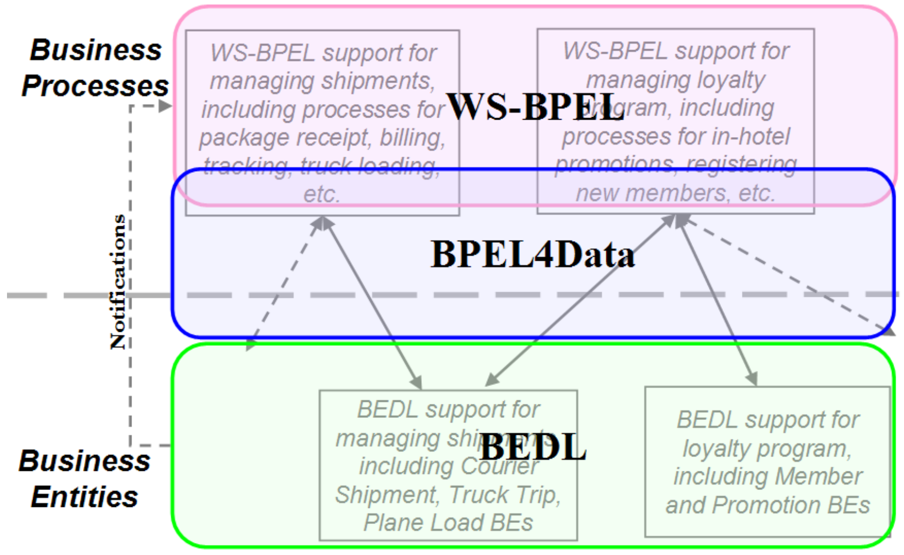
Business Processes
WS-BPEL support for managing shipments, including processes for package receipt, billing, tracking, truck loading, etc.
WS-BPEL
WS-BPEL support for managing loyalty program, including processes for in-hotel promotions, registering new members, etc.
BPEL4Data
Notifications
BEDL support for managing shipments, including Courier Shipment, Truck Trip, Plane Load BEs
BEDL
BEDL support for loyalty program, including Member and Promotion BEs
Business Entities

Figure 4. BPEL4Data in relation to WS-BPEL and BEDL

Part 2 will also discuss the following:

- Execution semantics between the BPEL (with BPEL4Data extensions) container and the Business Entity runtime)
- Execution semantics between the BPEL (with BPEL4Data extensions) container and a Human Task Manager container
- Execution semantics between the Human Task Manager container and the Business Entity runtime.

Advantages of using Business Entities

As we’ll discuss in Design methods using Business Entities, the use of BEDL in conjunction with a process-based specification language enables a holistic approach to modeling and deploying BOMs, an approach that combines both data and process in a fundamental way throughout the model building and deployment activities. Using pre-cursors of BEDL (see [2, 11]) several case studies and applications of this holistic approach [6, 7, 13, 15, 17] have demonstrated some of the benefits of this holistic approach. In this section, we’ll describe some of these benefits.

Customer satisfaction and buy-in

- The BE approach complements the traditional top-down, process-centric way of decomposing business operation domains. Indeed, for business processes with a large end-to-end scope, starting the design activity with a few key Business Entities provides an insight and clarity very early that is rarely achieved with the complexity of a multi-level process decomposition [15]. The entity *lifecycle view* can serve to derive the reference process models and for further detailed decomposition.

Indeed, in applications using the holistic approach, business managers have obtained substantially better insight into the business operations of their enterprise, because of the high-level, end-to-end view that Business Entities facilitate.

- The BE approach enables substantially improved communication across different silos of an enterprise, because in many cases a BE type spans multiple silos, and provides a common vocabulary for the stakeholders of those different silos.

- The BE approach can be used as a framework to manage requirements. The approach brings data and process together in a coherent and consistent way that can serve as a framework to plug all other requirements, such as rules, user interface design, process variations, key performance indicators, low-level business logic, and so on. [11, 15] .

- To minimize operational risk and boost adoption of BPM initiatives. Most BPM projects carry a certain level of risk because process and data concerns are often tackled as separate design activities, leading later on to costly re-factoring [1]. Following the holistic approach, all relevant concerns are synchronized right up front and flow from a common design base, thus mitigating these risks.

Simplicity, flexibility and time to value

- Using the BE approach, one can capture the key entities, their states and tasks that change states rather quickly even for a large complicated system. This provides enough semantic information to deliver an application almost from the first design iteration (and with proper tooling this rapid prototype generation can be automated). This then serves as a seed application that can grow as more details are added [11, 15, 14]. Business managers find these rapid prototypes invaluable for quickly gaining an understanding of the overall BOM design, expressing necessary changes early in the design process, and gaining confidence that the business transformation process will succeed.

- The approach forces a *simple (but complete) start* and then a *spiral evolution* of the design as all requirements are modeled incrementally but always stay consistent and coherent.

- The *business entity* - *access policies* - *process* is akin to a server-side rendition of the proven model-view-controller (MVC) [21] pattern. As with MVC, this enables an unprecedented flexibility in how the three layers can be attached, re-attached and independently developed and maintained.

- The BE approach allows for modeling and execution of data-related concerns that hitherto were tackled outside the BPM scope. For example, automatic generation of data access calls for owned data, data governance access rules and policy enforcement, dynamic “smart” forms and user interface generation based on access policies, ad-hoc data updates in human tasks, and so on. These are examples of concerns that can be modeled and executed in a unified, integrated fashion from within the BPM domain. [14].

- Business Entity lifecycle *states* represent process milestones and a reflection of business objectives. These milestones provide a stable base to anchor top-level processes that are unlikely to change over time. Furthermore, as a business strives to gain operational efficiencies most changes are detailed low-level functional processes and desk procedures. This separation ingrains a level of flexibility and dynamicity in which most changes can be contained in low-level process fragments with minimal impact to the rest of the stable base.

Cost reduction

- As more requirements are modeled up-front, there is less re-factorization and ripple downstream. This results in fewer change requests late in the solution development lifecycle, and in the deployment and maintenance lifecycle [14].

- Processes are factored around the entity life cycles. This increases the scope for reuse of the process definitions. It also leads to agility in design, as changes can be localized in process fragments without affecting other parts of design.

Design methods using Business Entities

Similar to other modeling paradigms in computer science, such as entity-relationship modeling in database management or object-orientation in programming, the Business Entity paradigm is based on a novel way of combining previously known constructs. As such, new ways of thinking are needed to apply the paradigm effectively. Speaking in broad terms, finding or "discovering" BE types that capture appropriate pieces of the business operations of an enterprise is at this time partly a science and partly an art. While a full discussion of methods and guidelines for finding appropriate BE types is beyond the scope of this article (see [11, 2] and the case studies [6, 7, 13, 15, 17, 29]), this section briefly introduces some of the main concepts used when designing with BEs.

We envision three related approaches for incorporating the Business Entity approach when designing BOMs:

- **Process-first approach:** This is the traditional approach to business operations modeling, in which processes are the primary focus. This can be augmented, once some of the processes are sketched or more completely designed, by thinking about the key business objects that are being manipulated by the processes. The business objects, whose instances evolve as they are manipulated by the processes and pass through the business--that is, the business objects with a clear lifecycle--are good candidates for the BE types. From here, the design focuses on understanding the linkages between the processes and the BE types, and then refining the design of both, including a refined design for how they interact.
- **Data-first approach:** In this approach, one begins by looking for the key BE types in the focus area of the business transformation. By talking with subject matter

experts (SMEs), one searches for key conceptual business objects that evolve as they move through a business (see Discovering Business Entities). Once some key BE types are identified, it is possible to provide more detail for their lifecycles by specifying the processes that modify the BE instance data and move the instances along their lifecycle. From here, as with the process-first approach, there is continued refinement in the design of both the BE types and the processes, and of how they interact.

- **Holistic approach:** This approach starts with a perspective that unifies, from the beginning, the intrinsic interconnection of the BE instances and the processes that act upon them. One begins by thinking about potential BE types, their high-level lifecycles, and key processes that can move the BE instances along their lifecycle. A unified meta-model can be used [similar to that in 11, 2]. A spiral model approach for extending and refining the design is then applied. The resulting BOM, which incorporates both BE types and processes into a unified model, can then be mapped to the BEDL and WS-BPEL components for deployment.

The holistic approach will perhaps prove to be most effective in the future, as the community of BOM designers becomes more familiar and practiced with the BE perspective. We sometimes refer to the process-first approach as *using the process-centric on-ramp* and the data-first approach as *using the data-centric on-ramp*. Either of these may be more useful to a given design practitioner. Importantly, both of these on-ramps should be used to work towards a BOM design that integrates both the BE types and processes into a robust, unified whole. The loose coupling between BEDL and process-centric standards such as WS-BPEL helps to enable the process-centric and data-centric on-ramps, because BEDL and WS-BPEL are both largely self-contained. Also, the separation makes it relatively easy to view partial or completed BOM designs primarily from the perspective of BEs or primarily from the perspective of processes.

Discovering Business Entities provides some high-level guidelines for discovering useful BE types and Detailed design methodology describes a method for creating BOMs that take advantage of the BE perspective. Design patterns involving Business Entities and processes introduce some new patterns of BOM design. Part 3 in this series will describe the patterns and their usage scenarios in greater detail.

Discovering Business Entities

Finding a family of useful BE types to form the skeleton of a BOM for a given scope of an enterprise typically requires a taking a fresh look at the enterprise. It is important to clear one’s mind and ask questions about the fundamental objectives of the enterprise and the high-level activities used to achieve them. It is essential to avoid getting distracted by the current modus operandi, which in most cases evolved from a largely process-centric perspective, and may also be deeply siloed.

When searching for the key BE type or types within a scope, it is useful to focus on the following kinds of questions:

1. **What is your business producing?** Enterprises are typically set up to perform certain activities, or produce certain goods, in a repeatable manner. The repetitions typically correspond to BE instances and the classes of similar activities or goods typically correspond to BE types. Each time the activity is performed or the good produced, this corresponds to one instance of a conceptual entity that evolves as it passes through the business. In many cases, a BE type will have a lifecycle that has a definitive end (for example, servicing an insurance claim ends with a settlement payment or a denial), while others may have a very long life (for example, customer history in a loyalty program). In some cases, one BE type might have multiple children of another BE type (for example, an academic conference might have numerous submissions, o there might be BE types for both *conference* and *submission*; a customer loyalty program typically has reward redemptions, and so there might be BE types for both *customer history* and *redemption*).

   There can be other patterns of the relationship between BE types. In an e-commerce scenario, for example, one might focus on *order* and *shipment* as two main BE types, where a single order might be broken into multiple shipments. If parts of two orders might be combined into a single shipment, then it may be useful to consider *line item* as a BE type along with both *order* and *shipment*. In most cases, there are between one and five BE types in a typical department or division of a larger enterprise.

2. **What are the key goals, and how do you measure them?** Often, the goals, including key performance indicators (KPIs), are closely correlated with the over-arching BE types of an enterprise. These goals and KPIs are often conceived by a team of business leaders, who are not thinking about, or even aware of, the details of how processes are being implemented. The kinds of data collected for the KPIs often suggest important attributes of the BE type information models.

3. **How do you measure progress towards the goals? Are there key sub-goals?** Often the steps or sub-goals that are monitored during the progression towards a goal correspond the business-relevant milestones, for example, states in the BE type lifecycle state machine. If there are different ways to achieve a major goal, these may correspond to different possible paths in the state machine.

4. **Are there different stages of your activity that work on the same underlying entity, even if different documents are used to describe it, or different parts of your enterprise are performing the work?** One of the more subtle challenges in identifying BE types is deciding what should be grouped into a single BE type vs. separated into two or more BE types. As a general rule of thumb, it is helpful to make each BE type span as far as possible across an enterprise, even if it cuts across several divisions or silos. In this way, you enable a focus on conceptual entities that retain their identity as they progress through the enterprise. For example, consider the process of making a loan and tracking the repayments over time. In some cases ([15]) it will be useful to combine these two aspects into a

single BE type. This facilitates communication between the loan creators and the loan trackers, reducing hand-off problems between these two sides of the overall business. In other cases it might be useful to separate these into two BE types, for example, if the loans might be bundled together after creation. Often, it is the possibility of one-to-many or many-to-many relationships that leads to breaking related notions into two or more BE types.

Philosophically there are two approaches to finding Business Entities – the *concrete* and the *conceptual*, as shown in Figure 5. In the concrete approach, as shown on the right-hand side of Figure 5, the existing data domain is analyzed to come up with the *logical view* of the BE types. This can be equated to traditional data modeling, in which disparate information sources (databases, applications, content, and so on) are analyzed and the logical view created to capture the key structures (data and metadata) and their relationships. Unlike traditional data modeling, however, here one is most interested in finding BE types whose instances follow a somewhat predictable lifecycle as they flow through the business. In doing so, one could identify the BE (the key data structures of importance) and its child data structures (*Business Items* or BIs in Figure 5) used by the BE instance during its lifetime. In Figure 5, BE2 is a *concretely* identified BE. After identification, BE2 is "upgraded" for business analysis, where its lifecycle, access policies and information model are created or further rationalized. Thus in the concrete view, the BE actually has an existing physical representation. Even with the concrete identification, it is not necessary to flesh out the entire data model up front. The identification usually results from inspecting the business goals and objectives. Even a fledgling design of a family of BE types can be very helpful in the early phases of designing a BOM.

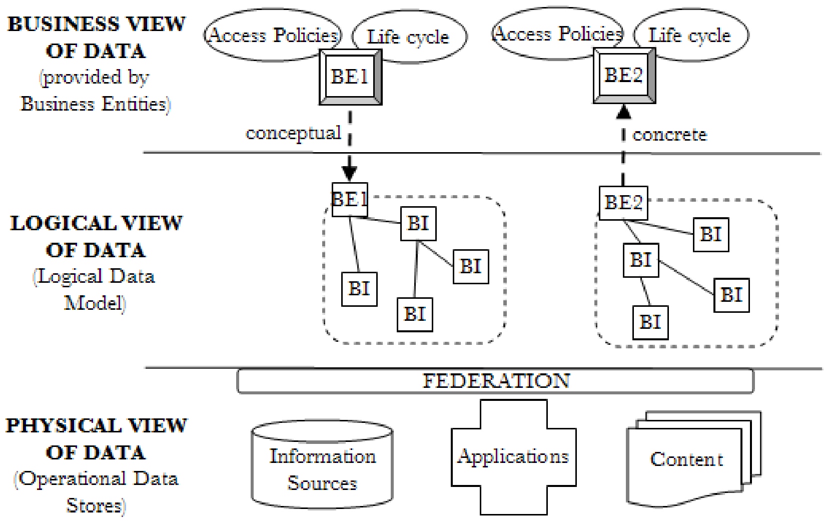

Figure 5. BE discovery: conceptual vs. concrete

The *conceptual* approach shown on the left-hand side of the figure begins with talking to the business SMEs to understand in a top-down way what the key dynamic conceptual business-relevant entities that underlie their operations are. These are typically already manifested in a fragmented manner in the existing IT systems. For example, in a typical restaurant, the concept of Guest Check can serve as a key BE type. This is because a Guest Check typically records all the necessary information about restaurant operations, from customer ordering, to the kitchen preparing meals, to payment, to accounting. However, its IT representation is likely fragmented into finer data structures like an Order, Kitchen Order, Payment, and so on. In this case, the Guest Check BE type is a discovered *conceptual* entity that aggregates all necessary information in its lifecycle. By following the Guest Check the restaurant has end-to-end visibility of its operations. So Guest Check is identified as a BE type and gets formally added to the logical view of the data model as a container for other data structures (the children), as represented by BE1 in Figure 5.

Design patterns involving Business Entities and processes

The combination of Business Entities and processes permits the development and use of several new patterns for BOM design that are not generally available in a purely process-centric context. We briefly discuss here two fundamental reasons for this.

First, is that BE instances persist across long swaths of a process, and progress through the states of their lifecycle model. In the simplest case, suppose that there is a long process that is focused primarily on a single BE type. A useful way to modularize this process is to fragment or break the process into chunks, where each chunk corresponds to the work needed to move a BE instance from one of its lifecycle states to the next lifecycle state. (It may also be useful that some fragments focus on sequences of activities that are done while the BE instance remains in a single state.)

Second, it is sometimes useful to view synchronization between processes in terms of how the underlying BE instances are progressing along their lifecycle. For example, in a conference scenario, if there are *conference* and *submission* BE types, then it can be useful to let submission evaluation be handled by one process, and let submission ranking (where all the papers are considered simultaneously) be handled by a distinct process. As another example, consider a hotel scenario, where there are BE types for *charge* (corresponding to the fee for a single service, such as dry cleaning or restaurant) and *folio* (to hold all charges made against a room). The routine processing for each charge (for example, computing discounts and taxes) might be done by one process, where as the linking of charges to folios might be done by another. The former processing might be done one charge at a time, whereas the latter might be done in bulk. In this way, the fragmenting of processes permits the natural use of different styles, yet all of the processes are appropriately stitched together by the underlying BE type lifecycles.

Detailed design methodology

In this section we'll describe a methodology for creating a BOM that uses Business Entities. The method starts with identification of high-level BE types and processes, and leads to a detailed design that incorporates both. The design task can be structured and conducted in two distinct phases:

Business operations modeling phase: This is the high-level design phase, typically corresponding to Levels 1 and 2 in the process decomposition hierarchy. Following are the high-level steps in this phase:

1. Determine the scope that the BOM is to cover. The scope could range from the entire enterprise to a functional area.
2. Decide whether data-centric on-ramp or process-centric on-ramp makes sense. For large scopes it is preferable to start with the data-centric on-ramp to gain early insight. The reason is that with a handful of BE types it is possible to conceptualize the entire scope with a small number of end-to-end concepts. The high-level, end-to-end perspective often helps to gain early buy-in from the business owners.
   - In the case of the data-centric on-ramp, identify the key BE types, including their intent--possibly an informal specification of their information models, and a high-level specification of their lifecycle models. Then define, at a coarse-grained level, the key processes required to create and evolve the BE instances through their lifecycles.
   - In the case of the process-centric on-ramp, list at a high level the key processes done in the project scope. Then identify the key BEs (and their states), annotate the activities with state changes, and document the attributes that will be manipulated by activity.
3. Identify the *business roles* associated with each process. Add the access policies covering the list of attributes identified for the BE types at this stage
4. Optionally, generate a UI-driven functional prototype application [15], based on these initial, but self-contained, specifications of processes (with state change annotations), Business Entities and access policies. Use the prototype to validate the business design with all key stakeholders.
5. Repeat steps 1 through 4 until consensus is reached across the board on the BOM design.

In the case of both data-centric and process-centric on-ramps, what you will have at the end of this phase are:

- The key Business Entities and their lifecycle
- The Level 1 and 2 process steps that take the BE instances through their lifecycles.
- The initial set of BE attributes (and associated child structures) that make sense in the given scope.
- The initial set of access policies.
- Optionally, the functional prototype.

Functional Design phase: This is the low-level functional design phase where the high-level design from the first phase is fleshed out, equivalent to designing Layers 3 and below in the process decomposition hierarchy. In this phase the BE types play a lesser role and design work is done with the low-level *business items* (the children of the BEs, as described in Discovering Business Entities). Following are the possible steps in this phase:

1. Identify natural groupings of BE types and of processes, as described in Figure 2 and its description.

2. Determine whether we are modeling the as-is or to-be functional design. In either case, the next set of steps is identical, but the stated purpose is as decided a priori.
3. Model the processes and identify business items passed from activity to activity.
4. Identify the business item details being CRUD-ed and add the business items to the BE type information model.
5. Identify *current roles* (for as-is) or *future roles* (for to-be) that will have responsibility for invoking and executing processes.
6. Identify the *notifications* needed from the BE types in order to trigger or inform the processes in the system.
7. Simulate and adjust the processes to establish a baseline.
8. Refine the access policies.

The work products at the end of this phase are:

- The *clusters* of BE types and processes
- Level 3 and below processes
- The complete BE type information and lifecycle models.
- The *roles* and *access policies*.
- The *notification* types to be supported and *subscriptions* from the processes.

We are heartened to observe that there is a growing community of researchers and practitioners who recognize the importance of bridging process and data as the Holy Grail of business operations design and implementation. The Business Entity Lifecycle Analysis (BELA) technique [11] introduced with IBM's SOMA [28] is a commercially available methodology that follows the above design approach, and is being aggressively adopted in the field.

Related work

Work on the Business Entity approach began with two seminal papers, on "Business Artifacts" [2] and "Adaptive Documents or ADocs" [3]. A survey of much of the related work is available in [10]. While IBM Research has been involved with much of this work, constructs very similar to Business Entities are being studied by other researchers (for example, [30, 38]). This section provides an overview of some of the entity-centric research that is most relevant to the BEDL approach to Business Entities introduced in this paper.

While there are some similarities between the notions of Business Entity and Master Data Management (MDM), there are also significant differences. Both attempt to find integrated views of data that may be spread across an enterprise. The difference is in the application context, and in particular in the emphasis in Business Entities on the dynamic lifecycle. For example, most would consider a Customer or a Product as master data, because in the context of everyday business transactions across functional units, the Customer and Product are non-transactional (that is, they maybe referred to, but not changed). But there are certainly customer on-boarding and product development processes, and in those processes the Customer and Product act as Business Entities. Once they achieve steady-state they are pushed into the realm of MDM. Thereafter, maintenance processes take over to manage their continued life.

BPM vendors focus on processes first, while data is at best an after-thought. MDM vendors, on the other hand, keep data in the forefront with processes as a necessary evil. The holistic view that we propose here allows for both on-ramps without losing focus on either. Perhaps this could be the common converging lingua franca for BPM and MDM and as such could help alleviate serious issues [1] faced today by keeping and evolving the two separately.

Conceptually, Business Entities can be made accessible as REST [23] resources. In the BPM context, the states of Business Entities reflect the progression of business data as it is "touched" by business processes. Thus, the lifecycle of the Business Entity provides a layer of business-relevant governance around the CRUD access of the generic resources managed through the Business Entity. Note we are drawing an analogy to the enablement of the REST "style" with Business Entities. This is independent and unrelated to the usage of REST as a data access protocol. How Business Entities enable REST-style BPM is discussed in more detail in [22].

There has been considerable work in the theoretical aspects of the process and business object lifecycle duality. In particular, the use of business object life cycles to assess the "goodness" of process models [4, 5, 8, 9]. While the duality is an important perspective from an analysis point of view, the holistic combination and coordinated evolution of process and key business objects is the focus of using BEDL for BOM design and deployment. In addition, we focus here on the architecture, data-first on-ramps, and relationship to data modeling, master data management, REST and most importantly the execution of this combination.

The notion of the *artifact-centric interoperation hub* introduced in [34] has significant similarities to the notion of BEDL component developed in this paper. The focus of [34] is to use an "interoperation hub" to facilitate the collaboration of multiple SOA-style services that are working together to achieve a common goal (such as to form a supply chain or to support hiring in a largely outsourced context). Speaking intuitively, the hub acts as a "structured whiteboard" that the different services can write on and refer to as they contribute to the overall activity of the collaboration. Although using different terminology, the hub is organized around BE types, and the actions of the participating services correspond to performing processes against the BE instances being managed by the hub. It appears that these interoperation hubs could be realized using a BEDL specification and supporting engine (perhaps with some extensions).

There is also ongoing work at IBM Research to investigate alternative approaches to represent Business Entity lifecycles [10,12] that are more declarative than finite state machines. This work might lead to an extension of the BEDL perspective in coming years.

Summary

This article has introduced a new proposed standard, called *Business Entity Definition Language (BEDL)* that can bring the advantages of the Business Entity approach into the mainstream of business process modeling. BEDL can be used alongside widely accepted process-centric standards such as WS-BPEL [19] and BPMN [20], without disrupting the industry investment in those process-centric standards and their supporting engines and tools. This article describes how BEDL and WS-BPEL (or BPMN) can be used together and in complimentary ways to design and deploy Business Operation Models (BOMs) that embrace both process and data as first-class citizens that blend in a holistic manner. It also provides the formal syntax of the proposed BEDL standard, and outlines a method for designing BOMs using BEDL and WS-BPEL. Part 2 of this series will present the formal extension of WS-BPEL that enables the specification of interaction between WS-BPEL and BEDL components in a declarative manner.

Acknowledgements

The authors would like to acknowledge the expert advice, constructive criticism and continuous encouragement from the following:

- Geoff Hambrick, Greg Adams, Pablo Irassar, Joachim Frank, Vishwanath Narayan, Santhosh Kumaran, Eric Herness, Jerry Cuomo, Naveen Balani, Rob High, Asit Dan, Claus T Jensen, Claudia Zentner, and Bob Madey from IBM's Software Group Division
- David Cohn, Anil Nigam, Fred Wu, Adrian Flatgard, Mark Linehan, Noi Sukaviriya, Florian Pinel, Terry Heath, Jana Koehler, Jochen Kuester, Sridhar Iyengar, Kevin McAuliffe, and Jorge Sanz from IBM Research.
- Lance Walker from the IBM CIO's office.
- Ali Arsanjani, Kerrie Holley, Ray Harishankar, Sham Vaidya, Jerry Denman, Jim Redman, Julien Petruic, Brian Paulsen, Shuvanker Ghosh, Abdul Allam, Mike Starkey, and Al Hamid of IBM Global Business Services to push forward with the release of the BELA capability with SOMA.
- Jack Parsio, Blair Binney, Gautam Majumdar, and Jay Strosnider from IBM's Internal Supply Chain organization for adopting the some of the early generation of the technique and tools.
- Marc Hoffman and Mike Kennedy from Emericon Inc., a valued IBM BPM business partner, who were instrumental in formulating the design methodology laid out in the Detailed design methodology section.
- Richard Hull is grateful for partial support from NSF grant IIS-0812578 for his contributions to this article.

About the authors

Prabir Nandi is a Research Staff Member in the Business Informatics Department at IBM's T.J. Watson Research Center. He is an inventor of the Business Entity concepts and has led its continued research and development for the last several years, including the

now commercially available Business Entity Lifecycle Analysis (BELA) capability as part IBM Global Business Services method and tools.

Dieter König is an architect for IBM WebSphere BPM products. He is a member of several OASIS technical committees for the standardization of Web Services Business Process Execution Language (WS-BPEL) and Service Component Architecture (SCA) specifications. Dieter has published many articles and has given talks at conferences about Web services and workflow technology, and is co-author of two books about Web services.

Simon Moser is a Software Engineer and Architect with the Business Process Solutions Group at IBM's Software Laboratory in Boeblingen, Germany. He holds a M.Eng in Computer Science and Engineering from the Technical University of Ilmenau, Germany. He has published many papers and given talks at international conferences, mainly in the area of Web service systems and business processes.

Richard Hull is a Research Manager in the Business Informatics Research Department at IBM's T.J. Watson Research Center. He is widely recognized for his research contributions in the areas of Web services, business process management, and database theory. He became an ACM Fellow in 2007. His current research foci include bringing a declarative style to the Business Entity lifecycle approach for the modeling of business operations, and applying the Business Entity lifecycle approach to the challenges of service composition and interoperation.

Vlad Klicnik is the lead architect for the WebSphere Business Modeler product. Prior to joining the Modeler team, Vlad was an architect for WebSphere Integration Developer, and a foundation member of the Eclipse development team.

Shane Claussen is the Chief Architect of WebSphere Process Server and oversees other aspects of architecture and development in the IBM BPM portfolio.

Matthias Kloppmann oversees the architecture of IBM's WebSphere BPM runtime, and is involved in BPM standardization activities at OASIS (BPEL4People) and OMG (BPMN 2.0). Over the last fifteen years, he has been involved in the development of three generations of BPM middleware products.

John Vergo is the Senior Manager of the Business Architecture Department and Strategist in the Services area of Research in the Business Informatics Research group at the IBM T.J. Watson Research Center in Hawthorne, New York. He has extensive business modeling and architecture experience, working with large enterprise IBM clients. His current research interests include Business Entities, business architecture, component business modeling (CBM), business design, and business transformation. His past research areas include human-computer interaction, user-centered design methods, multimodal user interfaces, e-commerce user experiences, speech recognition, natural language understanding, scientific visualization, 3D graphics and software development methods.

He has a BS in Mathematics and Psychology from the University at Albany, an MS in Computer Science from Polytechnic University and an IBM Research MBA.

References

1. Clay Richardson & Rob Karel, Forrester Research, *Process Data Management: Like Your Brain And Your Heart, BPM and MDM Can't Survive Independently*
2. A. Nigam and N.S. Caswell. "*Business artifacts: An approach to operational specification*." IBM Systems Journal, 42(3):428-445, 2003.
3. P. Nandi, S. Kumaran, J. Chung, T. Heath, R. Das, K. Bhaskaran, "*ADoc-Oriented Programming*", The 2003 International Symposium on Applications and the Internet (SAINT'2003, Jan 27-31, 2003, Orlando, Florida, USA)
4. K. Ryndina, J. M. Küster, H.Gall. *Consistency of Business Process Models and Object Life Cycles*. In Workshops and Symposia at MoDELS 2006, volume 4364 of LNCS, pages 80-90, Springer, 2006.
5. K. Wahler, J. M. Küster. *Predicting Coupling of Object-Centric Business Process Implementations*. In proceedings of the 6th International Conference on Business Process Management (BPM). 2008
6. K. Bhattacharya, N.S. Caswell, S. Kumaran, A. Nigam, F.Y. Wu. "*Artifact-centered operational modeling: Lessons learned from engagements*." In IBM Systems Journal, Volume 46, Number 4.
7. S. Kumaran, P. Bishop, T. Chao, P. Dhoolia, P. Jain, R. Jaluka, H. Ludwig, A. Moyer, and A. Nigam. *Using a model-driven transformational approach and service-oriented architecture for service delivery management*. IBM Systems Journal, Volume 46, Number 3, pp. 513—529
8. J. M. Küster, K. Ryndina, H. Gall. *Generation of Business Process Models for Object Life Cycle Compliance*. In proceedings of the 5th International Conference on Business Process Management (BPM), volume 4714 of LNCS, pages 165 -181. Springer, 2007.
9. S. Kumaran, R. Liu, and F.Y. Wu. "*On the Duality of Information-Centric and Activity-Centric Models of Business Processes*", 20th Intl. Conf. on Advanced Information Systems Engineering (CAiSE), June, 2008, Montpellier, France.
10. R. Hull. *Artifact-centric Business Process Models: Brief Survey of Research Results and Challenges*. To appear in Proc. of On The Move Federated Conferences, November, 2008. (published in Springer-Verlag LNCS).
11. P. Nandi, J. Strosnider, S. Kumaran, S. Ghosh & A. Arsanjani, "*Model-driven synthesis of SOA Solutions*", IBM System Journal's special issue on SOA: Modeling to Implementation, Vol 47, No 3, pg 415, Aug 2008
12. R. Hull. "*Artifacts in Business Processes: Helping Workflow become Declarative, or, New Model -> New Questions*", Keynote presentation at 20th Intl. Conf. on Advanced Information Systems Engineering (CAISE), June, 2008, Montpellier, France.
13. Rong Liu, Frederick Wu, Yasodhar Patnaik, Santhosh Kumaran. "*Business Entities: An SOA Approach to Progressive Core Banking Renovation*" IEEE International Conference on Services Computing, September, 2009, Bangalore, India.
14. N. Sukaviriya, S. Mani, V. Sinha. "*Reflection of a Year Long Model-Driven Business and UI Modeling Development Project*" 12th IFIP Conference on Human-Computer Interaction (INTERACT 2009), September 2009, Uppsala, Sweden
15. Tian Chao, David Cohn, Adrian Flatgard, Sandy Hahn, Mark Linehan, Prabir Nandi, Anil Nigam, Florian Pinel, John Vergo, and Frederick y Wu. "*Artifact-based transformation of IBM Global Financing*". To appear, Intl. Conf. on Business Process Management (BPM), Sept. 2009.
16. P. Nandi and S. Kumaran. "*Adaptive Business Objects - A New Component Model for Business Applications.*" Proc. Intl. Conf. on Enterprise Information Systems, pp. 179-188, 2005
17. K. Bhattacharya, R. Guttman, K. Bowles, F.F. Heath, S. Kumaran, P. Nandi, F.Y. Wu, P. Athma, C. Freiberg, L. Johannsen, A. Staudt. "*A model-driven approach to industrializing discovery processes in pharmaceutical research.*" In IBM Systems Journal, Volume 44, Number 1.
18. IBM Consulting Services, *A Component-based approach to strategic change*
19. *Web Services – Business Process Execution Language (WS-BPEL) Specifications*
20. Business Process Modeling Notation (BPMN)
21. Model-view-controller

22. *A RESTful Architecture for Service-Oriented Business Process Execution*, Proceedings of the 2008 IEEE International Conference on e-Business Engineering table of contents (ICEBE'08), pgs 197-204
23. REST, http://en.wikipedia.org/wiki/Representational_State_Transfer
24. *BPEL4Data: binding WS-BPEL to Business Entities,* Prabir Nandi, Dieter Koenig, Simon Moser, Richard Hull, Vlad Klicnik, Shane Claussen, Matthias Kloppmann, John Vergo, to be published soon on IBM Developerworks as part of the *Data4BPM* series.
25. O. Zimmermann, V. Doubrovski, J. Grundler, and K. Hogg, ''*Service-oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned*,'' Proceedings of the 20th SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM Press, New York (2005), pp. 301–312.
26. SAP Solutions
27. XML Process Definition Language
28. A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah,S. Ganapathy, and K. Holley, ''*SOMA: A Method for Developing Service-oriented Solutions,*'' IBM Systems Journal 47, No. 3, 377–396 , 2008.
29. Rong Liu, Frederick Wu, Yasodhar Patnaik, Santhosh Kumaran. "*Business Entities: An SOA Approach to Progressive Core Banking Renovation*" IEEE International Conference on Services Computing, September, 2009, Bangalore, India.
30. Matthias Born, Florian Dorr, and Ingo Weber. "*User-Friendly Semantic Annotation in Business Process Modeling*." Proc. Intl. Conf. on Web Information Systems Engineering (WISE) Workshops, Lecture Notes in Computer Science Volume 4832/2007. Springer Berlin/Heidelberg. November, 2007.
31. Guy Redding, Marlon Dumas, Arthur H.M. ter Hofstede, and Adrian Iordachescu. "*Modelling Flexible Processes with Business Objects*". Proc. IEEE Conference on Commerce and Enterprise Computing (CEC), Vienna, Austria, July 20-23, 2009.
32. Liangzhao Zeng, Hui Lei, and Henry Chang. "*Monitoring the QoS for Web Services*." Proc. Intl. Conf. on Service Oriented Computing (ICSOC), pp. 132-144, 2007.
33. D. Cohn, P. Dhoolia, F. Heath III, F. Pinel, and J. Vergo. "Siena: From PowerPoint to Web App in 5 Minutes" (demo paper), Intl. Conf. on Service Oriented Computing (ICSOC), December, 2008.
34. *Richard Hull, Nanjangud C. Narendra, and Anil Nigam. "Facilitating Workflow Interoperation using Artifact-centric Hubs." Proc. Intl. Conf. on Service-Oriented Computing (ICSOC), November 2009.*
35. Henk de Man. "*Case Management: Cordys Approach*." BPTrends, February, 2009.
36. Wil M. P. van der Aalst, Mathias Weske, Dolf Grünbauer: *Case handling: a new paradigm for business process support. Data Knowl. Eng*. 53(2): 129-162 (2005)
37. R.J. Glushko and T. McGrath. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services*. MIT Press, Cambridge, MA, 2005.
38. Serge Abiteboul, Pierre Bourhis, Alban Galland, Bogdan Marinoiu: *The AXML Artifact Model*. 16th International Symposium on Temporal Representation and Reasoning (TIME), Bressanone-Brixen, Italy, 23-25 July 2009, pp. 11-17.