Business Entity Lifecycle Analysis (BELA) Capability Pattern: *an information centric, but complementary approach to SOA*

Ali Arsanjani, Prabir Nandi, Shuvanker Ghosh, Senthil Mani

Abstract

Although information is important everywhere, traditional process modeling (the prime driver for most SOA applications), tend to consider activities and their sequencing as the main abstraction. Information is relegated as 'passive' data structures expressed as input and output to an activity. In this paper we make the case that information should be a first class construct in process modeling and in some cases is the only viable approach to lead with. In that context, we discuss the value of the Business Entity Lifecycle Analysis (BELA) capability pattern being introduced starting SOMA 3.2. We illustrated how BELA provides a rigorous and consistent approach to align Process, Information & Business Rules towards identifying services with the right levels of granularity. We also propose a complimentary Service Component Design pattern that provides a modular componentization of the design space engineered for performance and graceful evolution.

Introduction

We revisit *Business Entity Lifecycle Analysis (BELA)* as a SOMA [3] capability pattern [2]. In particular we examine the key value propositions towards process model validation, service and service component identification. We examine usage scenarios where an information-centric approach (aka BELA) is more viable. We propose a consistent service design pattern and show it usage in different user interaction patterns. Finally we take a first look at conceptual design of Websphere & GBS tooling to support the paradigms.

Process Modeling techniques and Service Identification

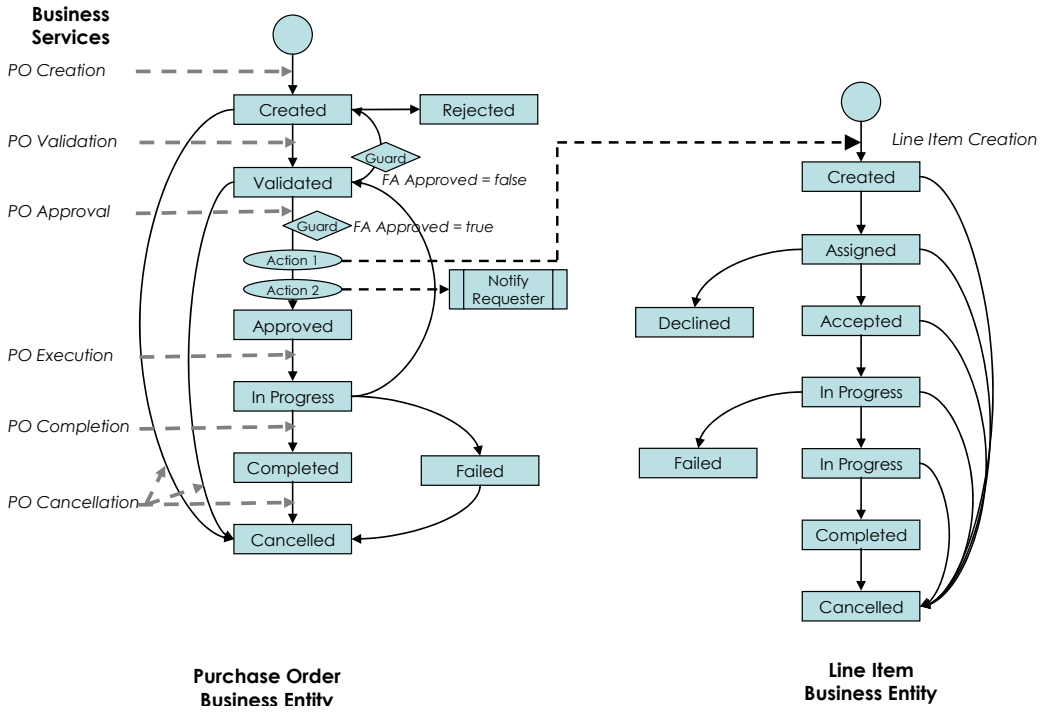
Business Entity Lifecycle driven Service Identification

The key innovation behind Model-driven Business Transformation (MDBT) [4] from IBM Research is to approach business process modeling from the perspective of business entities (or business artifacts [1]) and their lifecycles. In this approach, business entities are the information structures that in essence are the key drivers to the functional area in which business process modeling is being performed. In most cases, these business entities are quite evident eg. the "Claim" in an Insurance Claim Process, the "Trouble Ticket" in a Telecom Service Assurance Process, the "Service Order" in a Service Delivery Management process etc.

The approach begins with identifying the primary business entities in the functional area and laying out their "life story" i.e *who (aka the business role)* does *what* to these entities throughout their lifecycle, from the point they are created to their ultimate archival or deletion. A convenient representation to capture entity lifecycle is a *state chart.* Although state charts maybe deemed as an IT level construct, experience has shown that business subject matter experts are comfortable to talk about the *states* of the entities and the *activities* that causes the entity to *transition* from one state to the next. In the business entity driven business process modeling, entity states can be equated to *process milestones* and the overall "business state" of an executing process is the combined set of states each business entity is in at that point.

A *Business Process* in this context is a set of *communicating business entities* and a *Business Service* is a *state transition* in a business entities lifecycle model. The diagram below shows a Procurement Process modeled as a lifecycle of two communicating business entities, the *Purchase Order (PO)* and *Line Items.* The *Line Items* are business entities (as well) because during initial analysis it was noted that processing of the Line Items need to be tracked individually as they get fulfilled by different Supplier organizations and at different times.

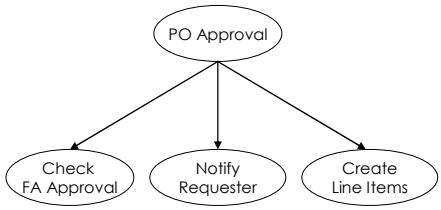
The Business Services identified from the PO lifecycle model are ones that cause the PO to change state.



Let's dig a bit deeper and focus on the transition from the *Validated* to the *Approved* state of the PO. For the PO to reach the state of being *Approved,* three things need to happen[1],

1. The Financial Analyst (FA) must approve. This is modeled as a *guard* condition on the transition. If the FA disapproves, the PO is sent back to the Requester to be updated.
2. If the FA approval is received, *Line Item* business entities are created for each corresponding item in the PO. This is denoted as a state transition *action*
3. If t he FA approval is received and after successful creation of the Line Item business entities, the PO *Requester* will need to be notified, modeled as another transition action.

Just for the above portion of the PO lifecycle the services identified are "PO Approval", "Check FA Approval", "Notify Requester" and "Create Line Items". The Service Composition and dependency is shown below.



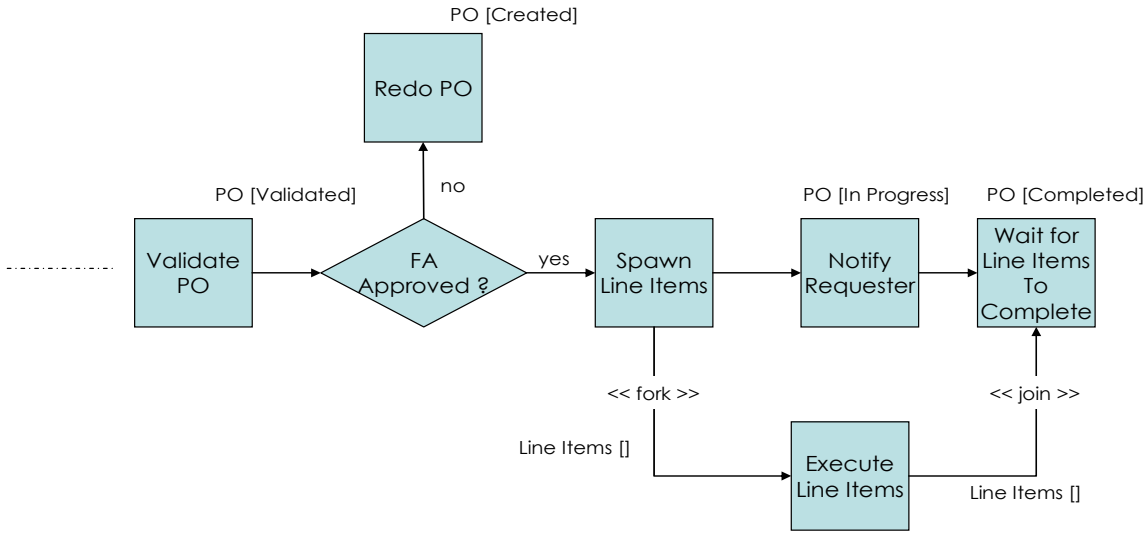
It maybe argued that without the business entity formalism, one could apply sound design skills to discover similar services and their dependencies. However, as business entity states are process milestones, the service that change state can be considered as "business-relevant" (aka a Business Service). The other services identified above are "side-effects" of the state transition. Although needed for completeness of the deisgn, they are just "details". A different company might have different logic/steps towards securing approval than what we show here

Process driven Service Identification and BELA

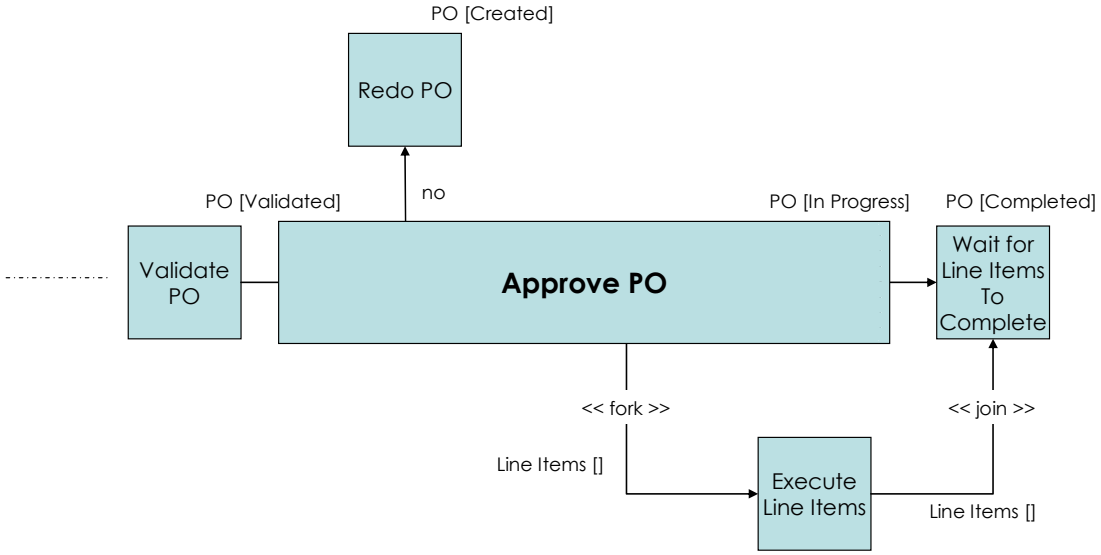
The other, more traditional way of modeling business processes is as a sequence of activities or tasks (*"verbs"*). Information is considered purely from the point of a stateless data structures fed as inputs to and outputs from tasks. One potential drawback of this approach is the absence of a guiding principle to determine the *granularity* of a task. This causes specification creep-in and for complex business processes more often than not results in a "task mine-field". In SOA, process models play an important role to identify candidate services. Each node in the process map is a potential candidate service. However, with improper task granularity the identified services could be unnecessarily huge in number. Maintenance becomes a problem as the process evolves to keep pace with rapid change in business.

The BELA *capability pattern* introduced with SOMA 3.2, overlays the lifecycle of key business entities on process models. This helps to maintain focus on activities of "business value" from the ones that aren't. Applying the technique, a "typical" process can be refactored around entity lifecycles, by grouping tasks that do not change the state of the business entity under (as a sub-process) tasks that do.

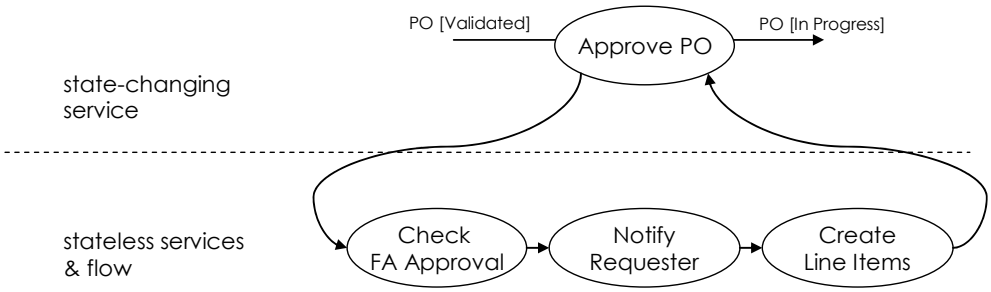
The diagram below shows the activity centric modeling of the Procurement Process we introduced earlier. The square boxes denote the *Tasks*, diamonds are *Decision Boxes,* and few *forks* and *joins* to express parallelism and synchronization. The *Purchase Order* is one of the key business entities and tasks in the process model are annotated to indicate if a state change occurs at the completion of the task. For example, *PO [Validated]* indicates that after the *Validate PO* task is completed, the PO will be put in the state *Validated.*



Applying the BELA technique, we group tasks between state changes as a logical unit of work. In our example, the tasks *FA Approved ?, Spawn Line Items* and *Notify Requester* are grouped together as single logical activity, *Approve PO* that takes the PO from the *Validated* to the *In Progress* state. This grouping is shown in the diagram below.



The services identified from the (BELA) aligned process model produces the candidate services and composition as below. The top level services are the ones that change the state of the entity. In the process the stateless services flow (also a microflow) will be executed before the entity moves into the target state (In Progress)



Thus BELA provides a mechanism to group tasks in process models towards identifying higher level services aligning with state changes of a key business entity. As the state represents a key process milestone, one can argue that these "state-changing" services are indeed business-relevant and at the right level of granularity. Additionally, BELA provides a good validation mechanism for the process model.

Process driven and Entity driven are complimentary

Interestingly, irrespective of the two different approaches to process modeling, business-entity driven or activity driven, one can arrive at the same set of candidate services and service hierarchy by applying BELA to traditional process models.

We argue that the choice of the starting technique is a personal preference, but the entity-driven approach will cut down extra steps. In the absence of any mandate towards process modeling, it would be advisable to straightaway adopt the entity-centric approach to save time and effort.

Usage Scenarios & Best Practices

In the previous sections we illustrated two approaches towards process modeling. We also showed how BELA applied to the activity-centric process model is equivalent to the business entity centric version. We also hypothesized that BELA provides a more scientific approach towards identifying business services. It also helps componentizing the process model by separating business-relevant services (aka the ones that change state) from the more automatable IT services (the ones that are stateless and triggered from state transitions). This also helps modularizing the overall design space. Metaphorically, activity-centric process models can be thought of as "completely procedural" and BELA provides a technique towards creating a (modular) "object-oriented" version with procedural code factored into object methods.

In scenarios where an activity-centric process model exists, it is recommended to apply BELA. In cases, where process models do not exist, it would be a personal choice to first do the traditional process modeling and then apply BELA, or start with the business-entity driven approach right away. However, from a technical standpoint, the entity-

centric approach is most appropriate to lead in with when one or more of the following conditions exist.

- The process is mostly event-driven i.e the sequence of activities are based on external events whose occurrence is unpredictable
- The process involves lots of rework and backflows
- The application has complex UI requirements that are mostly data driven. Typically these kinds of UI's require a conversational UI logic on data & functions.
- The process involves a significant amount of creative and collaborative work.

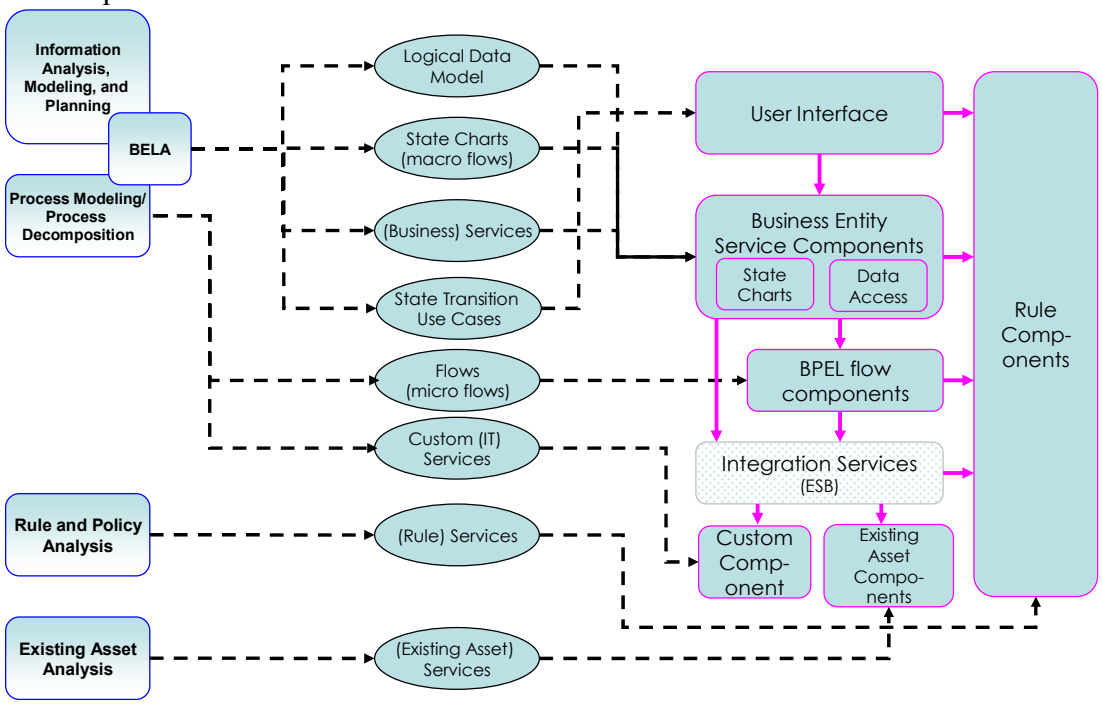
For business executives perspective the following value propositions of BELA are especially attractive:

- Focus on entities drives radical simplification
- Business clarity: from large wall charts to manageable business entity views enabling the business team to engage in the business design, reach consensus quicker and to lead the change
- Comprehensive business requirements from the BELA refactored process model drives less rework downstream
- Process, data, data lifecycle and role based access control designed upstream in the process model
- Fewer iterations – quicker conversions
- Cost reduction and faster cycle time over conventional 'design-develop-deploy' approaches
- Better / stronger connection between business and IT realization results in effective trace ability to bus requirements
- Ability to deploy a complex solution that links design to applications and data
- Reduced time and effort required to make a business change (new offerings to market, improve efficiency / profitability, new supplier and partner relationship…)
- KPIs and business rules automatically generated (entity state = process milestone) in the operational business model for performance mgmt

Business Entity based Service Specification pattern

In the previous sections we covered how the injection of the information perspective to process modeling results in a more complete and rigorous set of business requirements. We also saw how the technique helps in modularizing the design space by separating business-relevant services from IT services. In this section we will show how the same pattern can be translated into an efficient service component design resulting in a modular and componentized solution design. Modularity is key mantra, as mapping the design space around loosely coupled modules increases the potential for reuse. It also forms the cornerstone towards agile delivery as one can go from rapid prototyping to production delivery in an iterative fashion.

The diagram below shows the mapping between the SOMA Service Identification techniques (on the left), the work products they produce (in the middle) and how it can be used to populate the Business Entity Component design pattern [2] as part of SOMA Service Specification.



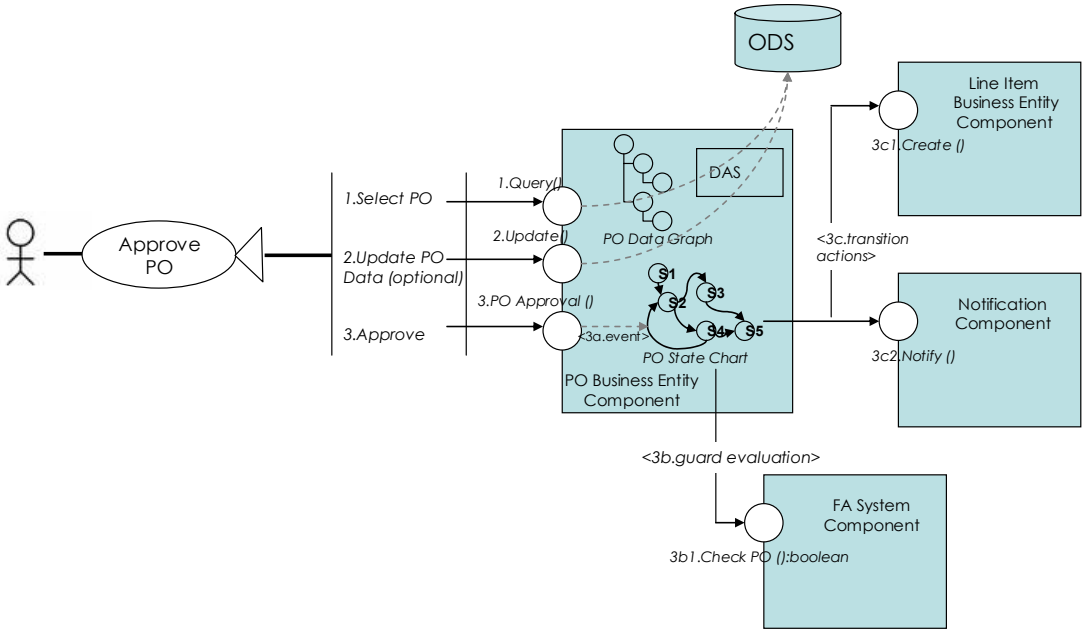
The table below describes the mapping and illustrates with the *PO Approval* example we introduced earlier.

SOMA Activity	**Output**	**Service Component Mapping**	**Comments**	**Mapping from the PO Example**
BELA / Process Modeling & Decomposition	State Charts (macro flows)	Business Entity Service Component : State Charts, Functional Operations	BELA refactors the process models around entity lifecycles. Activities are grouped around state transitions and promoted as a single service. Going by our arguments that since these services help reach process milestones (entity states) they are business relevant	PO State Chart Line Item State Chart Example of Business Service is *Approve PO*

			and candidates to be exposed as a *Business Service*.	
BELA / Information Analysis & Design	Logical Data Model	Business Entity Service Components: Data Access Service Metadata	A Conceptual Information Model will be produced initially as part of SOMA Information Analysis & Design. BELA will add more details as it analyzes the data added to the key business entities through its lifecycle. It also adds structural details like containment, cardinality etc thus mapping the conceptual information model to a *Logical Data Model*	PO-Line Item Data Model Eg. PO has at least 1 or more Line Items. Other attributes eg. total amount etc.
BELA/ Process Modeling & Decomposition	State Transition Use Cases	User Interface	Post BELA each top level activity cause state change in one or more business entity. A set of standalone *state transition use cases* can be derived from these top level activities.	*Approve PO*
BELA / Process Modeling & Decomposition	Flows (micro flows)	Flow Components	The *Flows* are the control flows that tie together the group of activities within a **single** state transition	Choreography of *Check FA Approval, Notify Requester* and *Create Line Items*
BELA / Process Modeling & Decomposition	Custom (IT) Services	Custom Components	The *IT Services* are the lower level (non state-changing) activities or tasks. As part of BELA they will be mapped	*Create Line Items*

			to a particular state transition flow.	
Rule and Policy Analysis	Rule Services	Rule Components	Rules will be used to model constraints and variance in all aspects of the solution design. For Business Entity Service components rules will be mapped to guard conditions on state transitions	*Check FA Approval*
Existing Asset Analysis	Existing Asset Services	Existing Asset Components	A solution more often than not will need to integrate with legacy systems to delegate functional services required by the process. Such legacy systems are identified during Existing Asset Analysis and appropriate functionality identified as *candidate services.* The *integration layer* is responsible to resolving any impedance mismatch - functional (required vs provided) and/or technical (conversation protocols, transport protocols, access control etc.)	*Notify Requester* could Use and existing email Infrastructure.

To further illustrate the use of the pattern, the diagram below shows the component interaction of the various components to implement the *Approve PO* use case. Here we assume a human is approving the PO through some UI screens.



1. **Select PO** – The user firsts selects the PO she wants to work with. The query operation (from the Information Services) is invoked on the PO Business Entity Service Component. The PO Datagraph, associated with the passed in "where" clause is then passed to the Data Access Services (DAS). The DAS then invokes the JDBC mediator which maps the input to an SQL call to retrieve the correct PO instance. The call ends with the PO data being rendered on the screen.
2. **Update PO Data** – The user then updates some of the PO data and sends an *update* call to the PO BESC. As in (1), the call is passed to the DAS which detects the changes and invokes the JDBC Mediator to update the PO with the changes.
3. **Approve PO** – The user finally clicks on the "Approve PO" button on the screen. This call is then routed to *approve()* operation (from the Functional Services) on the PO BESC. The state machine handler picks up the call and selects the appropriate state transition based on the *current state* of the PO and the *approve()* event. The guard condition on the transition is evaluated first by invoking the *Check PO()* operation on the FA System Component. If that passes (returns *true),* then the associated transition action *BPEL flow* is invoked. The BPEL flow invokes the *Notfication Component* to send the email and then invokes the *create()* operation on the Line Item Business Entity Service Component to create the a line item instance for each one in the PO and start their execution. The call returns after the flow executes successfully and the PO is moved to the target state *In Progress.* The next set of activities corresponding to each state transition out of the *In Progress* state is activated through the Human Task Manager.

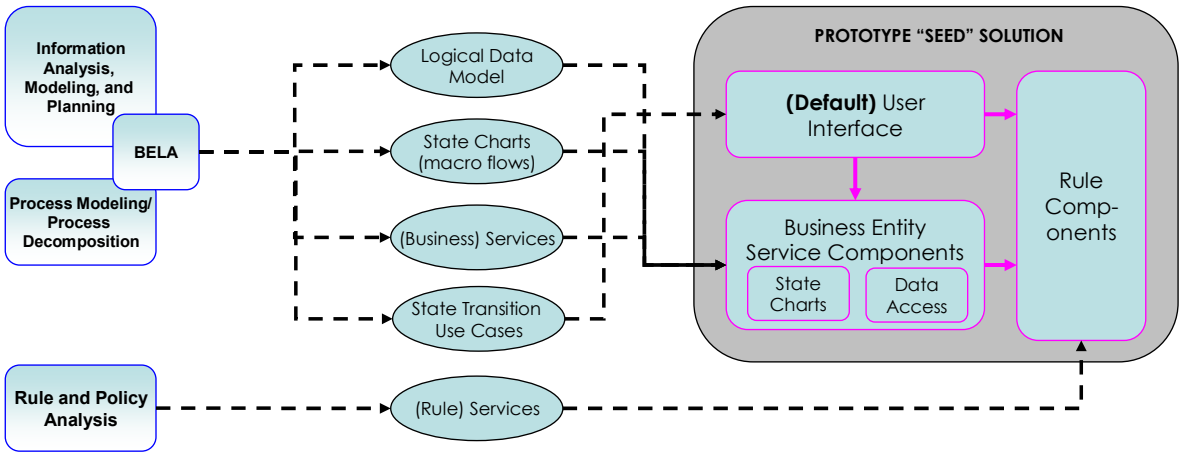
In this section we showed how the Business Entity Service Component pattern can be instantiated from SOMA Identification. At the core the pattern is based on

communicating state machines as in the individual Business Entity Service Components. This provides a great alternative to the current practice of creating a single long-running BPEL whose limitations are well known [reference]. The alternative, BESC based pattern, provides a modularized division of the design space that have tight semantics within, but loosely bound to others. Besides being the best option in certain usage scenarios (eg. lots of rework/backflows, complex & conversational UI requirements, data driven interactions etc.) the modular approach leads to a graceful evolution of deployed solutions with change in business requirements.

Rapid Prototyping

The Business Entity Service Component pattern has another key advantage. Due to its modular design and more rigorous specification, it is possible to generate a rough cut prototype solution right after BELA.

The diagram below shows the minimum set of SOMA activities that are needed to generate a complete prototypical "seed" solution. Essentially performing BELA and Rules Analysis (as pertinent at the business level) provides us with complete semantics to generate the "seed" solution shown on the right. A default user interface (set of screens) can easily be created to drive the state transition use cases



It's a "seed" solution as this forms the core components around which the other components will be added as more SOMA activities complete eg. Existing Assets, Process Decomposition etc. .

References

1. A. Nigam and N. Caswell, ''Business Artifacts: An Approach to Operational Specification,'' IBM Systems Journal 42, No. 3, 428–445 (2003).
2. J. Strosnider, P. Nandi, S. Kumaran, S. Ghosh and A. Arsanjani, "Model-driven Synthesis of SOA Solutions" IBM Systems Journal 47, No. 3, 415-432 (2008).

3. 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).
4. S. Kumaran, ‘‘Model-Driven Enterprise,’’ Proceedings of the Global Enterprise Application Integration (EAI) Summit 2004, Banff, Canada (2004), pp. 166–180.