

Creating Solution Architectures as SV-10c Models in DoDAF Using BPMN 2.0

By Lloyd Dugan of the DCMO ETW Team

In the DoDAF, different views are identified within the architectural framework for containing and relating various architecture objects. The operational view of event traces in a business process is represented as an OV-6c construct, which per Department guidance is to be done as a Business Process Model and Notation (BPMN) model. The Deputy Chief Management Office (DCMO) of the DoD has promulgated modeling guidance and training on how to create OV-6c models using the notational subset of the BPMN v2.0 specification known as the Analytic Conformance Class. This guidance consists of normative modeling approaches based on BPMN 2.0 primitives and patterns defined by the DCMO. Application of these approaches is required of artifacts created for the Business Enterprise Architecture (BEA), which has led to numerous key Business Process Areas (BPAs) being modeled as OV-6c views across the various end-to-end processes of the DoD's Business Mission Area (BMA).

There are many counterpart system views to operational views in the DoDAF, including a system view of event traces in a business system that is supporting one or more business processes. This counterpart is known as the SV-10c view, and it can be used to define the solution architecture of a business system that exists or is proposed that purports to provide such automation support. While an SV-10c is not technically one of the DoDAF views required for defining the BEA, it nonetheless should be done for legacy systems or proposed replacement systems – whether they are custom-built, commercial-off-the-shelf (COTS), government-of-the-shelf (GOTS), or some combination thereof – that claim or assert automation support for one or more business processes and conformance with the governing architecture for the one or more business processes. Alternatively expressed, an SV-10c should be used to show conformance of a business system with a corresponding OV-6c's requirements and architecture.

This DoDAF Journal article has been written to explain how this can be done using current DCMO BPMN 2.0 modeling guidance with some additional guidance related to modeling system behaviors. A principal import of this proposed approach is that BPMN 2.0 should be used to describe the behavior of systems rather than using other modeling languages such as Sequence Diagrams (SDs) from the Unified Modeling Language (UML). With its additional notational elements and behavioral semantics, BPMN 2.0 is far more capable of describing system behaviors than UML SDs. Furthermore, with BPMN 2.0's XML serialization, it is now possible to attribute the modeled elements in a way that richly fills out the XML export of a BPMN 2.0 model. This export can be translated into Resource Description Framework (RDF) format using the BPMN 2.0 ontology created by the DCMO and expressed in Web Ontology Language (OWL) format. Such a semantic expression provides the basis for directly measuring the conformance of a business system with governing architectures similarly expressed in OWL (by way of targeted queries of the architecture data using the query language for semantic data, known as SPARQL).

Modeling Challenges for Solution Architectures

Historically, there have been numerous problems associated with the definition of solution architectures as a means of determining conformance with governing architectures.

- Differences in spans of control – the controlling execution context in which operational or system behaviors occur – between the system’s view of the functionality it is supposed to deliver versus the organization’s view of the same functionality. For example, an organization’s view is process-centric and may recognize more than one span of control, whereas an Enterprise Resource Planning (ERP) system assumes it is the only span of control.
- Models created to describe business systems and to define business system functionality have typically proved lossy with respect to corresponding predecessor models created to describe business processes and to define business process requirements. True model round-tripping – being able to seamlessly trace back and forth from the two different types of models – has proved elusive and technically challenging. For example, shifting from using BPMN 2.0 (or comparable language) to describe a process to using UML SDs (or comparable language) to describe a system has made the transition from analysis-to-design-to-development problematic and reverse engineering impractical.
- Even using BPMN 2.0 can prove tricky since the modeler is able to represent the system as either a performer corresponding to a swim lane (wherein modeled elements are to be executed by the system), a completely abstracted performer based on certain task types (wherein the system has only indirect representation in the model), or a distinct participant separate from the process pool that is represented as a system pool.

There are also the explicit differences between operational and system views within the DoDAF. These differences are highlighted in Table 1 below.

<ul style="list-style-type: none"> • OV-6c (Operational Event Trace) <ul style="list-style-type: none"> – Focus: <ul style="list-style-type: none"> • Describes operational activity • Provides time-ordered examination of resource flows as a result of a particular scenario • Operational thread defined as a set of operational activities, with sequence and timing attributes, and associated resources needed – Intended Usage: <ul style="list-style-type: none"> • Analysis of operational events • Behavioral analysis • Identification of non-functional user requirements • Operational test scenarios <p>http://dodcio.defense.gov/dodaf20/dodaf20_ov6c.aspx</p>	<ul style="list-style-type: none"> • SV-10c (System Event Trace) <ul style="list-style-type: none"> – Focus: <ul style="list-style-type: none"> • Describes system activity • Provides time-ordered examination of interactions between functional resources • System thread is a sequence of functions and data interfaces that honor information needs of participating resources – Intended Usage: <ul style="list-style-type: none"> • Analysis of resource events impacting operation • Behavioral analysis • Identification of non-functional system requirements <p>http://dodcio.defense.gov/dodaf20/dodaf20_sv10c.aspx</p>
---	--

Table 1 – OV-6c vs. SV-10c in DoDAF

The manifestation of these view-driven differences has generally been revealed in confusion and variation in detail for models that purport to describe processes vs. models that purport to describe systems that support those processes. If the latter is simply a more detailed representation of the former, then traceability between the two is difficult to achieve and to preserve. On the other hand, if

the latter is a different a model altogether, then comparability is effectively compromised along with the capability to ascertain architectural conformance. Fortunately, there is a way out of this conundrum.

In the corollary OV-6c guidance provided by the DCMO, various modeling views of the same process space are called out:

- Milestone View – describes major phases of a business process as a simple and “happy path” sequence from start to finish, using only a Start Event, an End Event, and collapsed Subprocesses, with Sequence Flows connecting them all together
- Handoffs View – describes the handoffs from one performer to the next in the process sequence, which is either across Lanes in the same Pool with Sequence Flows or between Pools (and spans of control) via Message Flows
- Decisions View – describes the business logic behind control flow and assignment flow as expressed with diverging/converging data-based or event-based Gateways and Sequence Flows
- Procedures View – describes in rich detail the procedure-level behavior of a process, which is achieved through the use of more advanced Flow Nodes and behavioral semantics of BPMN 2.0.

These different views are intended to provide the modeler with guidance on how to iterate a model from initial cuts to a final version, and to support different “fit-for-purpose” needs of the model to speak to different constituencies and stakeholders.

It is possible to call out an additional view, a System View, that allows an organization’s process-centric view to be preserved as context for defining the System View. This System View will likely require more detail than the Procedures View, but it can still be done using the Analytic Conformance Class elements. However, some additional model element attributes – taken from the Common Execution Conformance Class – are required to fully outfit the BPMN model for a System View, along with some corresponding modeling guidance.

Introducing the System Pool

In a certain sense, a business system mirrors the one or more business processes that it is supporting through automation. It does this by implementing a function that is mediated by a human performer, is reacted to by a human performer, or is the result of something acted upon by a human performer. In software engineering, this is known as an inversion of control or responsibility, where the business system implements at run-time the functionality defined by the system’s interactions with abstracted elements outside the system context. Viewed in this way, a business system represented apart from but in relation to the supported business processes implements a type of delegation pattern that lets the modeled process elements “delegate” the implementation to one or more modeled system elements.

The concept of the System Pool was introduced contemporaneously with the BPMN 2.0 spec via the BPMN 2.0 By Example document (see <http://www.omg.org/spec/BPMN/2.0/>). Using an Incident Management example, the authors described how the concept of the System Pool could be introduced into a BPMN 2.0 collaboration model as the automated support for the business processes in the collaboration, and that this usage did not replace the collaboration but extended it. The principal import

of this approach is that a model preservation strategy – as opposed to a model round-tripping one – is now possible with BPMN. The Incident Management example is adapted and explained below, which is presented as a BPMN collaboration between various participants.

In the Incident Management process collaboration, a problem with a separate (and not shown) business system has generated the “incident” being reported, which is presented to a set of participants charged with fixing the problem. There is a customer facing participant that fields the request, which then hands it off as an opened “trouble ticket” for the Tier 1 Team to attempt to troubleshoot the problem. If the Tier 1 Team cannot fix it, then it is handed off to the Tier 2 Team to fix it. If it cannot be fixed contemporaneously, then the fix is added to the Backlog System for incorporation into the next release of the system that originally suffered from the problem. Into this mix is introduced the Trouble Ticket System as an additional participant in the collaboration that mediates the problem resolution activities.

The first step is to begin with an OV-6c view of the business processes. This is achieved via a BPMN collaboration diagram that calls out the major steps and performers with at least a Handoffs View or Decisions View, or perhaps even a Procedures View, level of detail. This is shown in Figure 1 below.

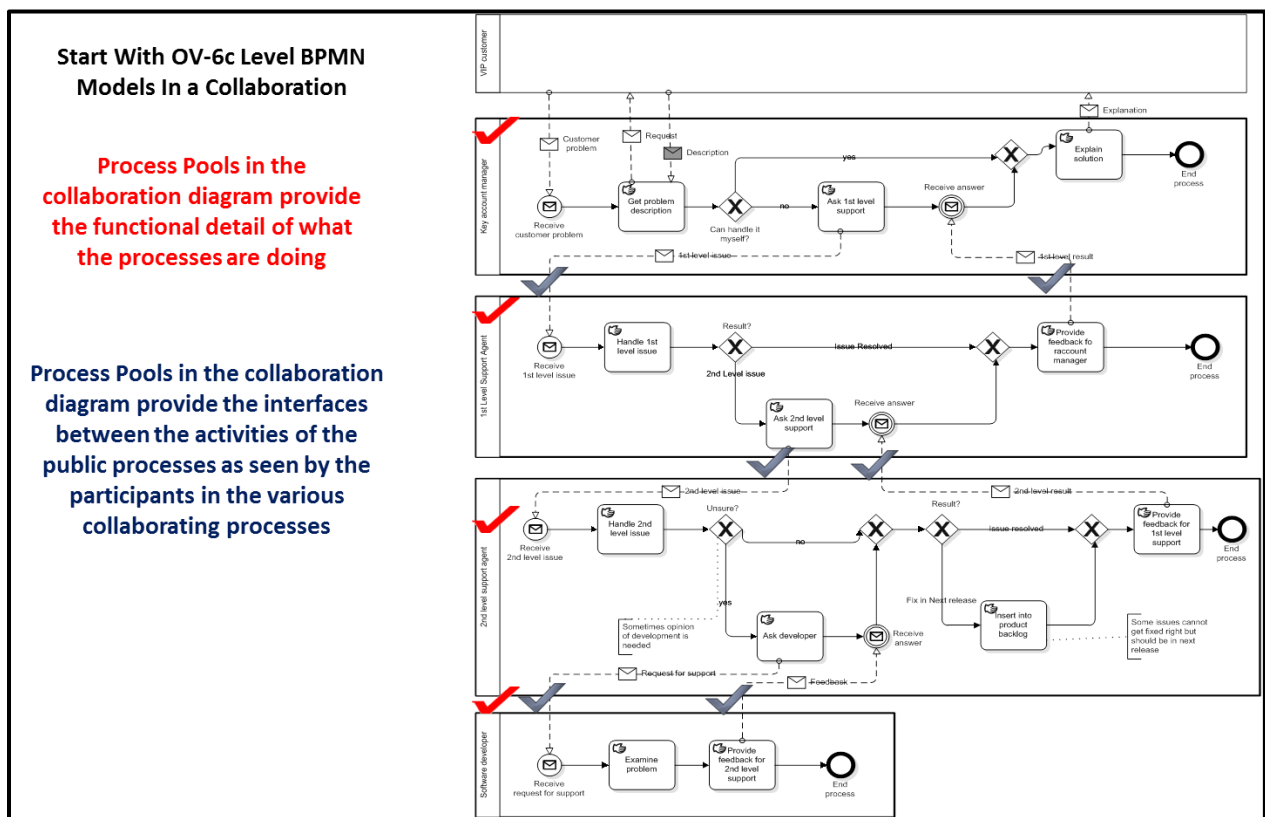


Figure 1 – Process Collaboration as the OV-6c View

(Note that the tasks initially shown in the model are all of the Manual Task type, which is what the original treatment of this approach used. The reason for that usage was that the model was ultimately intended to be executed by an enactment engine, which according to the BPMN 2.0 spec meant that any non-automated task had to be shown as a Manual Task type. However, an OV-6c model is not meant to

be executed in this sense, so in practice it is appropriate instead to use an Abstract Task type (no marker in the box) for most activities or Service Task and Business Rule Task types (as needed) to indicate the intended abstractions. Consequently, the reader should equate the use of the Manual Task type in these models to the use of the Abstract Task type that would typically be the case for OV-6c models.)

The second step is to introduce the System Pool, which is initially represented as a collapsed pool (wherein the inner details of the system's design are not visually represented). As with any external span of control, the other participants in the collaboration interact with the System Pool via Message Flows. These Message Flows into and out of the System Pool represent communicated information proceeding along the interfaces that exist between the other Participants and the System Pool. This configuration is presented in Figure 2 below.

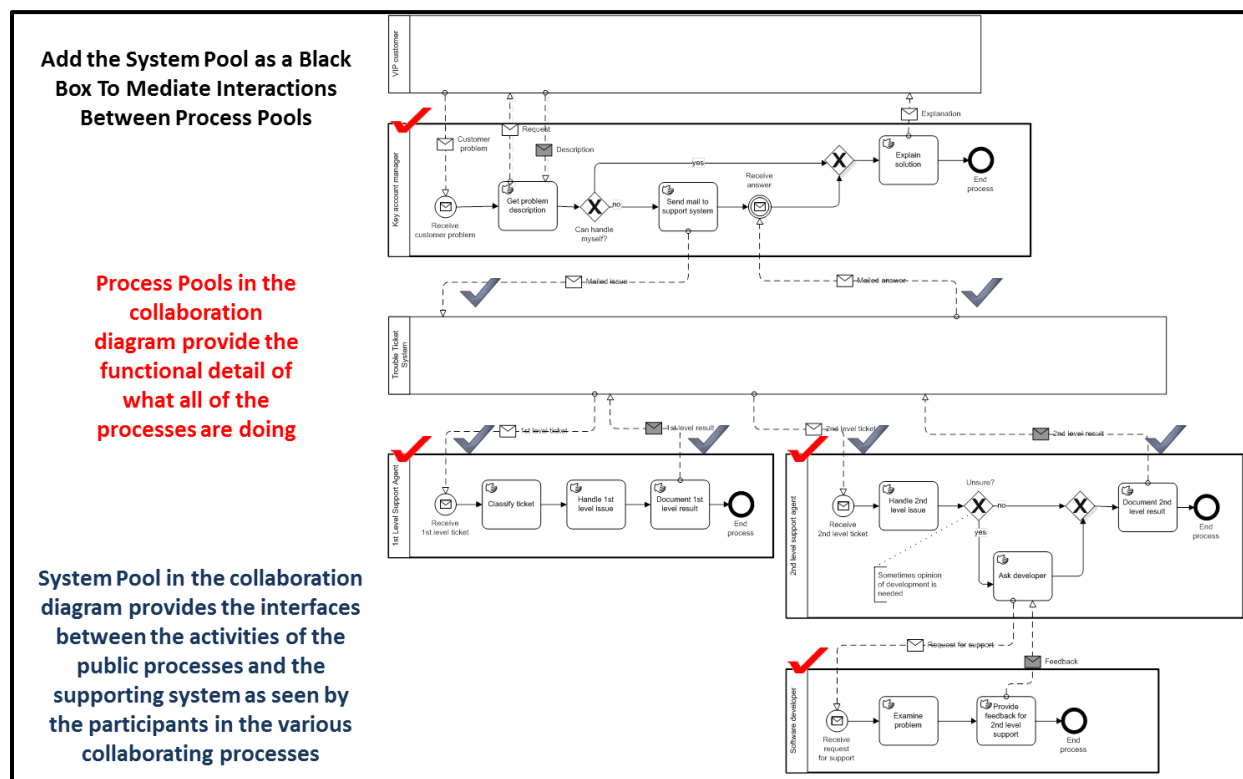


Figure 2 – Introduction of the System Pool

The third step is to then fill out the System Pool with the behavioral detail that describes the functionality of the system. Where in this functionality there is data communicated to or by the system from or to another participant in the collaboration, then Message Flows are connected at the element level on both sides of the communication. The purpose of the Message Flows is thus now clearly revealed as representing the human-to-system (user screens) or system-to-system interfaces (headless interactions via services) that exist within the Trouble Ticket system. This is illustrated in Figure 3 below.

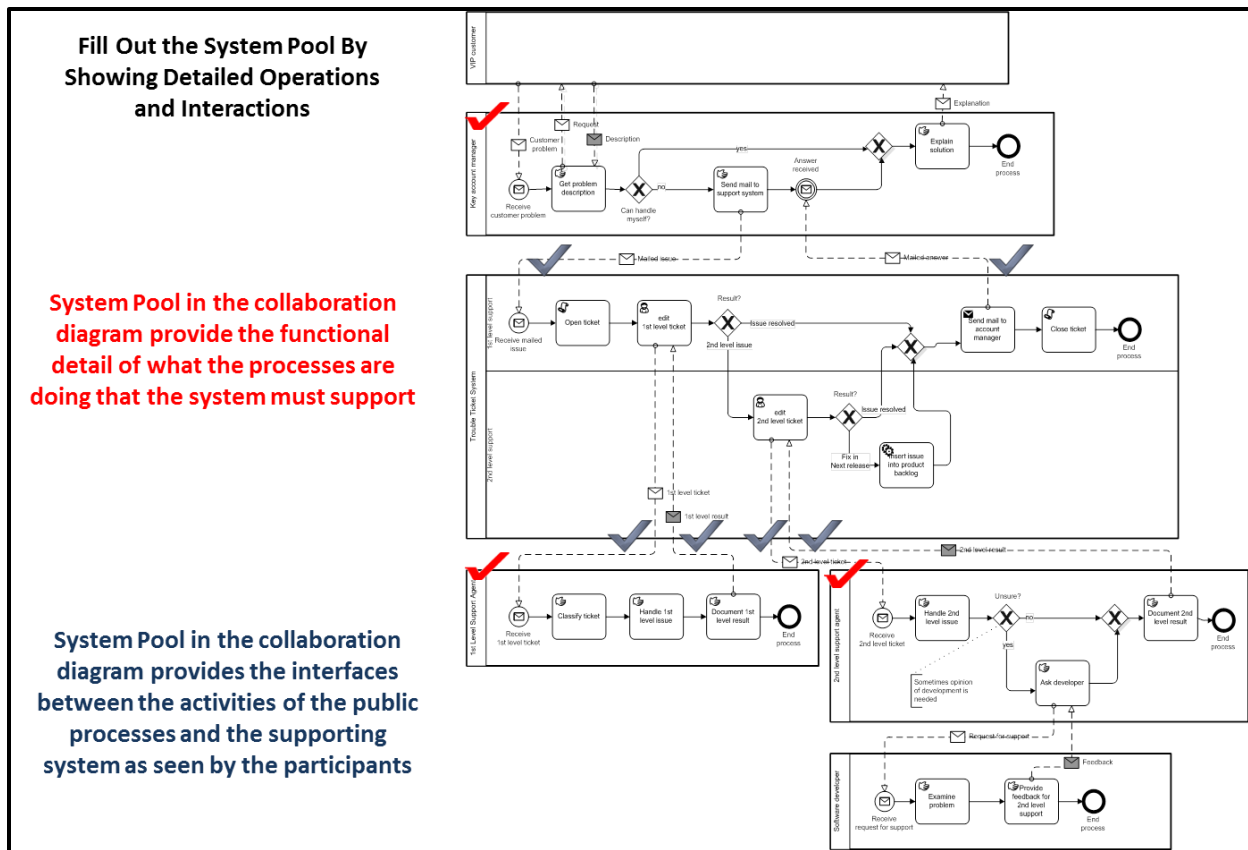


Figure 3 – Description of the System Pool

The final step is to then collapse the other Pools, leaving only the System Pool and the various Message Flows in the process collaboration. At this point, the Message Flows represent the input/output (I/O) requirements of the Trouble Ticket System that must be satisfied. As long as these are honored by the construct in the System Pool, the internal structure of the System Pool can be whatever the modeler uses for the design. Once that internal structure is finalized, the other Pools can be dropped off altogether and the System Pool itself can be handed off to a development team to build out based on the information contained in this system design statement. This is illustrated in Figure 4 below.

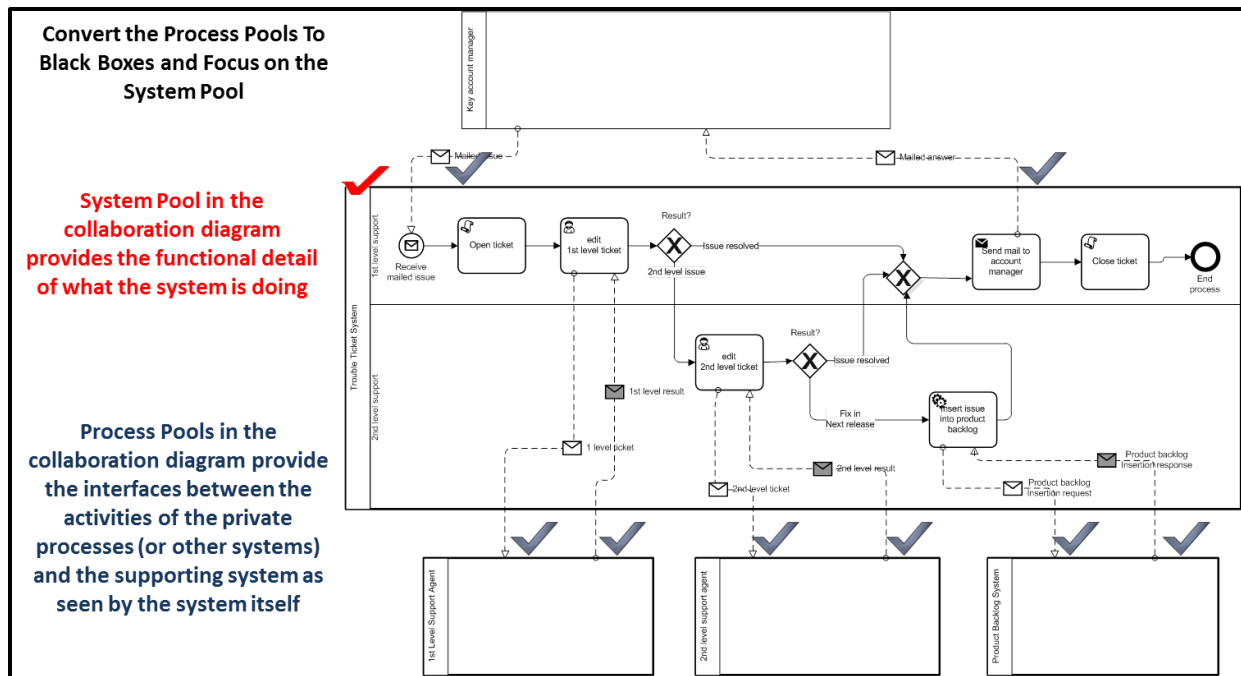


Figure 4 – Focusing on the System Pool

The System Pool can represent any type of automated solution, including an ERP system. Multiple System Pools can be similarly introduced to show how multiple business systems support one or more business processes (as opposed to a single system like an ERP supporting multiple processes).

Thus represented, the process model now has the information about the supporting system that is needed to enable measurement of its satisfaction of requirements, expressed as the informational requirements and output of the other participants. In addition, to the extent that these Message Flows map to Information Exchange Elements as architecture objects defined within an architecture (e.g., the BEA) created per an architectural framework (e.g., the DoDAF), then data as defined schemas or classes are thus mapped to the system that consumes or produces them. This mapping is similarly valid with regard to Data Objects represented in the System Pool, meaning they too can be mapped to defined schemas or classes as transient data consumed, created, or transformed by aspects of the system. The benefit to this mapping approach is that the system being designed automatically inherits the same set of schemas or classes already defined for (and thus applicable to) the modeled business processes.

Designing the System

Additional modeling guidance beyond what is required of the OV-6c modeling is needed to ensure that the system design is sufficiently descriptive and unambiguous in its intended behavior such that a developer could build and/or configure from the design's corresponding executable constructs. As part of that guidance, it is useful to recognize the numerous analogs to system concepts that are present in BPMN 2.0. This mapping is presented in Table 2 below.

System Concept	Equivalent BPMN Representation
Execution context and framework	<ul style="list-style-type: none"> Pool defines the controlling framework for the system Lanes segment system resource (performer) responsibilities System interfaces are realized as boundary-crossing moments with other participants
Control flow and routing logic	<ul style="list-style-type: none"> Sequence flow and conditional expression logic define order of execution Gateways and branches define inbound and outbound execution paths
Procedural and modular logic	<ul style="list-style-type: none"> Service task, send task, receive task, and business rules task define specific and germane operations that are executed via invocation User task defines an operation that is executed via one or more user interfaces Script task defines an operation that manipulates data that is executed internally Subprocess and Call Activity define a set of operations that are executed separately Start events and end events define the start and end conditions for execution Intermediate events define operations based on event type that are executed Boundary events add operations to the attaching activities that are executed
Data I/O declaration	<ul style="list-style-type: none"> Data objects as input or output with corresponding data sets as input or output, respectively, by way of corresponding data associations Messages and signals define data input or output as data payloads
Integration interfaces (APIs)	<ul style="list-style-type: none"> Service task, send task, receive task, business rules task, and user task define service interfaces to abstracted and/or external components Message event and signal event define service interfaces with other systems

Table 2 – Mapping BPMN 2.0 Concepts To System Concepts

Given this level of equivalency, modeling guidance is critically needed about how to use BPMN 2.0 concepts to represent system-level behaviors. Part of this guidance requires deep understanding and application of first principles of modeling, which can be broken down into two core concepts:

- **Decomposition**
 - By performer, which ensures separation by those entities responsible for performing atomic tasks (i.e., tasks responsible for a single outcome based on initial conditions)
 - By functional purity, which ensures that atomic tasks are homogenous with respect to executed operations and/or to processed data
 - By state transitions, which ensures that all meaningful changes in the central process object moving through the process sequence are the direct result of atomic tasks
- **Abstraction**
 - By using a subprocess (a Collapsed Subprocess or Embedded Subprocess), which encapsulates more detailed behavior in a self-contained child process that is performed within the context of the parent process
 - By using abstraction (principally the Service Task or the Business Rule Task), which displaces business logic away from the core process being represented and towards a separate participant as the provider of requested services (a Service Provider or Business Rules Engine, respectively)

- By using reusable components (a Call Activity for a Global Task or a Global Process), which displaces business logic away from the core process being represented and towards a standalone and separate construct that can be invoked.

Through the appropriate application of these first principles of process modeling, the appropriate granularity of modeled systems can be achieved. This granularity is an emergent property that results from the interaction of the degree of coupling between modeled elements (with dependent operations and data) and the degree of cohesion across modeled elements (that fits them together into a whole). These concepts are explained in Figure 5 below.

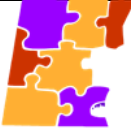



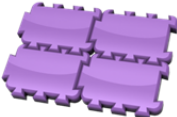
Cohesion	Coupling
 <p>Cohesion is a quality of relatedness that is driven by shared and/or common characteristics of the constituent elements</p>  <p>Cohesion conveys a sense of how efficiently the constituent elements all fit together that is often more evident once actually put together</p> <p>What To Look For: Cohesion in a process model is measured by the affinity that modeled elements have with respect to each other for performing a single function or set of related functions, based on their operational characteristics (functionality executed, data operated on, constraints applied, etc.).</p>	 <p>Loose coupling has few and simple interdependencies</p>  <p>Something in between has aspects of both types</p>  <p>Strong coupling has many and complex interdependencies</p> <p>What To Look For: Coupling in a process model is measured by the type and number of dependencies between modeled elements (e.g., data inputs and outputs), including dependencies with respect to external entities with which modeled elements interact.</p>

Figure 5 – Cohesion and Coupling as Elements of Granularity

The desired granularity can be achieved through the use of the following guidance with respect to tasks and events, which amounts to making sure that each task or event maps to a single operation. This is summarized in Figure 6 below.

Element	Granularity Guidance
Tasks	<p>Any activity in an SV10c Level model that:</p> <ul style="list-style-type: none"> - Is a task: should <u>execute one operation</u> for the activity within the process sequence, commensurate with its task type and any data input and data output associations - Is a subprocess: should effect the passing of control from the parent process to the child (and back) as its <u>one operation to be executed</u> for the activity within the process sequence, commensurate with its subprocess type and any data input and data output associations - Is a call activity: should execute call invocation as its <u>one operation to be executed</u> for the calling activity <p>An aborting boundary event attached to an activity in an SV10c Level model extends the functionality of that activity but the action of the event constitutes a <u>single operation for execution</u> within the process sequence, commensurate with any data output associations</p> <p>A non-aborting boundary event attached to an activity in an SV10c Level model extends the functionality of that activity but the action of the event <u>constitutes a single operation for execution</u> within the process sequence, commensurate with any data output associations</p>
Events	<p>A catching or starting event in an SV10c Level model constitutes a single operation for execution within the process sequence, commensurate with any data output associations</p> <p>A throwing or ending event in an SV10c Level model constitutes a single operation for execution within the process sequence, commensurate with any data input associations</p> <p>Note: This covers all event types except for the timer event, which is <u>by definition a single operation for execution</u> with the process sequence (i.e., a temporal condition is acknowledged)</p>

Figure 6 – Granularity Guidance By Flow Element Type

The summary effect of the application of the granularity guidance is easily summed up to the following:

- Granularity of the Process, Process Object, and Process Value should all generally match
- Modeled elements at higher levels should exhibit higher cohesion and lower coupling
- Modeled elements at lower levels should exhibit lower cohesion and higher coupling
- Level of granularity should be roughly equivalent across a process level.

The resulting granularity should represent an executable construct or, alternatively expressed, a construct that is sufficiently unambiguous such that a developer can build from it or an enactment engine can be configured to implement it.

As a system design, there are certain patterns that are optimal in terms of realizing consistent processing of transactions and preserving the integrity of those transactions during processing. One such pattern is ACID (defined below), which is the ideal nature of an individual task at the parent level of system designed in BPMN 2.0. Another pattern is BASE (also defined below), which can be the nature of implementing components that should be encapsulated into lower-order expressions (Subprocesses or Call Activities) or abstracted out (using a Service Task or a Business Rule Task). For example, the implementation of a User Task can consist of several screens (participating in a series of related navigational flows) that exhibit BASE-like properties at the end of each screen experience, but the set of screens are represented as a single User Task because at the point the collection exhibits ACID properties. The nature of these two patterns is described in Table 3 below.

<p><u>ACID:</u></p> <p>Tasks should typically be more ACID-like with respect to usage within the BPMN process sequence of the system design</p>	<p>A: <u>A</u>tomtic – task is a single operation task C: <u>C</u>onsistent – task behavior is consistent I: <u>I</u>solated – task results are isolated D: <u>D</u>urable – task results are durable</p>
<p><u>BASE:</u></p> <p>Tasks that are more BASE-like should typically be collected into abstraction Task types or into Subprocesses or Call Activities</p>	<p>B + A: <u>B</u>asically <u>A</u>vailable as an operation S: <u>S</u>oft state for the data object E: <u>E</u>ventual consistency of the intended result</p>

Table 3 – ACID vs. BASE Design Characteristics

In general, modeled Tasks should individually exhibit ACID properties, at least at the main or top-level representation, but the system design as a whole does not (and often will not) collectively exhibit ACID properties. The specific characteristics to seek for a given system design depend on the specific needs of the system being designed in BPMN, so these patterns are provided to aid in that determination.

Execution design patterns that can also be followed consist of different ways of processing transactions regardless of whether they are ACID or BASE:

- Idempotent – Process reacts only once for a specific instance of the submitted trigger, no matter how many times the same trigger is sent
- State Machine – Process manages the state of the transaction through a series of stateless moments in the Tasks
- Data Access Management – Cache access service vs. database access service is invoked via a Service Task.

Implementing these patterns amounts to selecting specific implementing component types, which should be a consciously-directed choice of the Solution Architect as opposed to an unintended result.

Usage and Attribution of Task Types

In using BPMN to design a system, the concept of the Task needs to be more fully understood. In BPMN 2.0, the Task type has specific behavioral meaning and, in some cases, defines the type of abstraction involved and the nature of the component that executes it. This is defined in Figure 7 below.

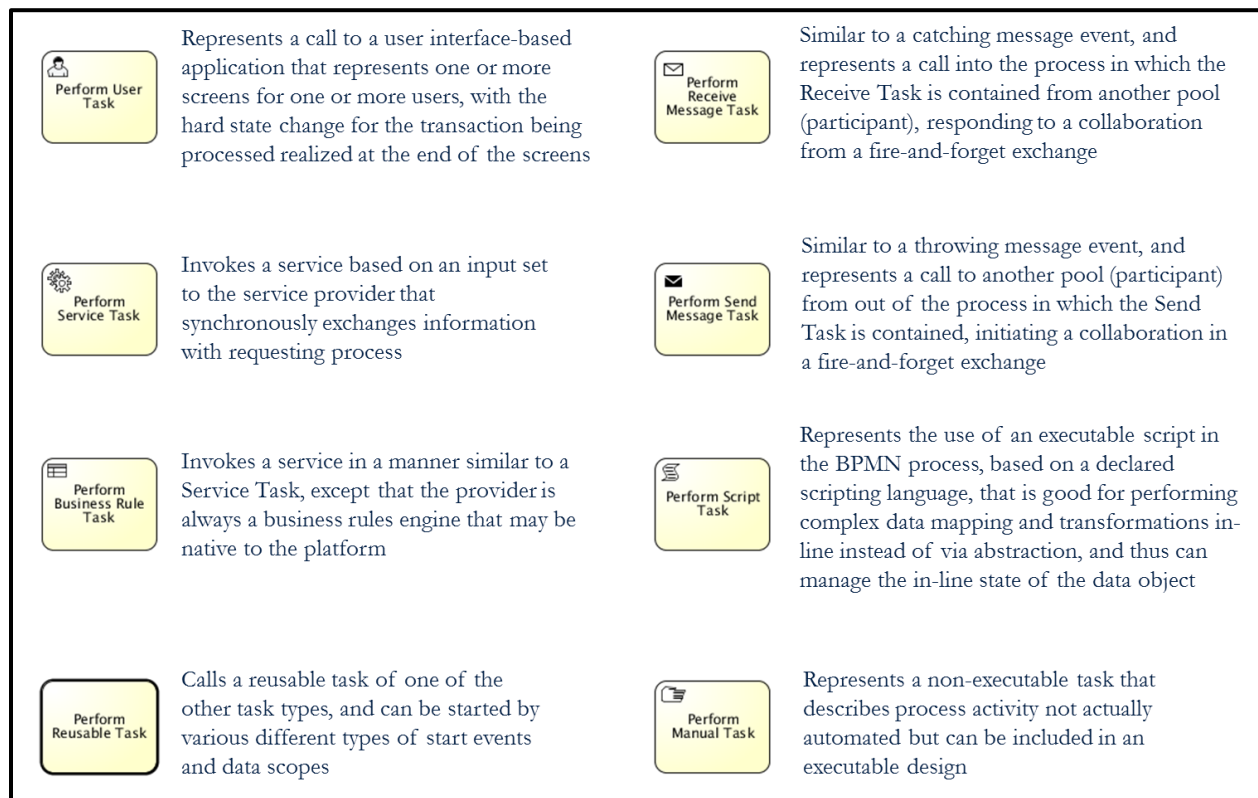


Figure 7 – Specific Nature of BPMN Task Types

To make the most of using specific Task types, it is necessary to capture or define the Operation Name for a Task that is an abstraction and is referencing an implementing component. It is also necessary to capture or define the Implementation Reference for the implementing component, which can be expressed as one of the following:

- Web Service – typically a Simple Object Access Protocol (SOAP)-style web service that leverages relevant application programming interface (API) defined standards for web services
- Unique Resource Identifier (URI) – typically a Representational State Transfer (REST)-style web service that leverages industry accepted API standards for web services
- Other – typically a custom API that is proprietary for the service provider or is required by the requester.

Note that the capability of the modeling tool may constrain how or if this level of attribution can be done, but these attributes – taken from the Common Execution Conformance Class – are available for use within the BPMN XML.

Usage and Attribution of Task Types

The concept of Messaging in BPMN 2.0 similarly needs to be more fully understood. In BPMN 2.0, the use of Messages, Message Flows, Message Tasks, and Message Events have specific behavioral meaning and identify the conversations that occur between participants as part of the process collaboration. In particular, the key message exchange patterns (MEPs) of Synchronous Request/Response and Asynchronous Request/Response can be represented in BPMN, though it requires a modeling convention and attribution scheme to do so. Request/Response is where a request is made by a participant of another participant that may generate a response from the latter to the former.

If this exchange is Synchronous, then the moments of Request and Response are blocking with respect to further processing, and thus are best suited for transaction processing of very short duration. The Request has an input payload, and the response has a concurrent output payload. This yields two Messages and Message Flows: the Request that is initiating and invoked as a call to the service provider, and the Response that is not initiating and is not a separate call to the service provider. The detailed nature of a Synchronous Request/Response MEP is outlined in Figure 8 and Figure 9 below.

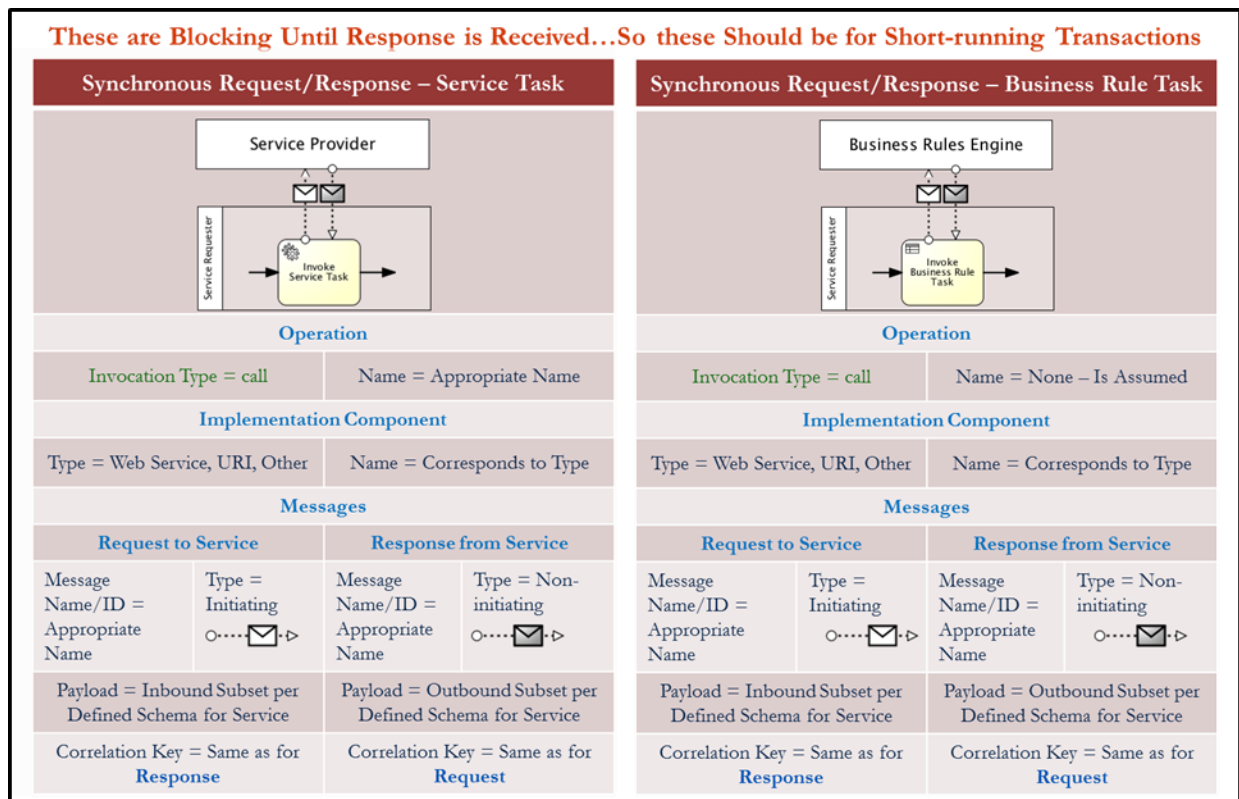


Figure 8 – BPMN Modeling Convention and Attribution Scheme, Synchronous Request/Response

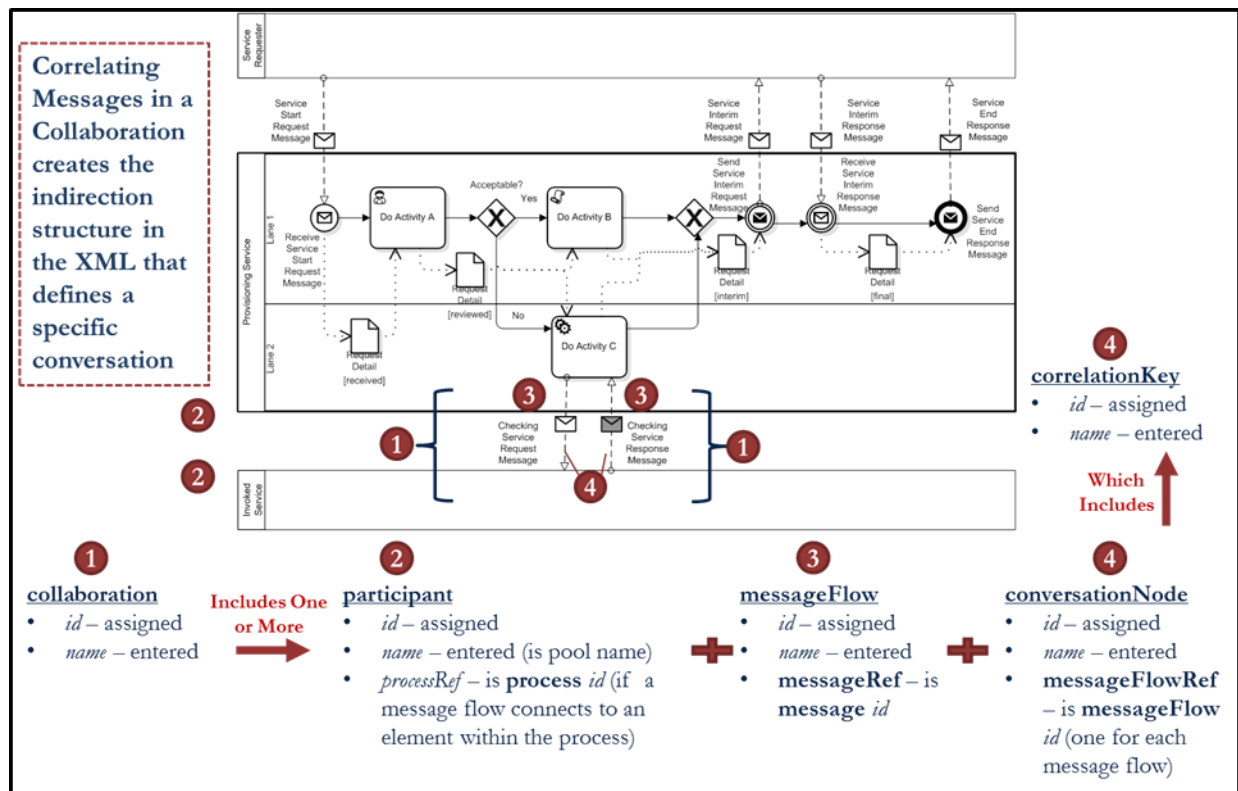


Figure 9 – Synchronous Request/Response Example

If the exchange is Asynchronous, then the moments of Request and Response are not blocking with respect to further processing, and thus are best suited for long-running transaction processing. This also yields multiple Messages and Message Flows: the Request that is initiating and invoked as a call to the service provider, and the Response that is also initiating because it is a call back to the service provider. If there is not Response expected, then the MEP involved here is also known as Fire-and-Forget (or One-Way) since the Request message is sent out and the call made, and that is all that is done. The detailed nature of an Asynchronous Request/Response MEP is outlined in Figure 10 and Figure 11 below.

These are not Blocking Until Response is Received...So these Should be for Long-running Transactions

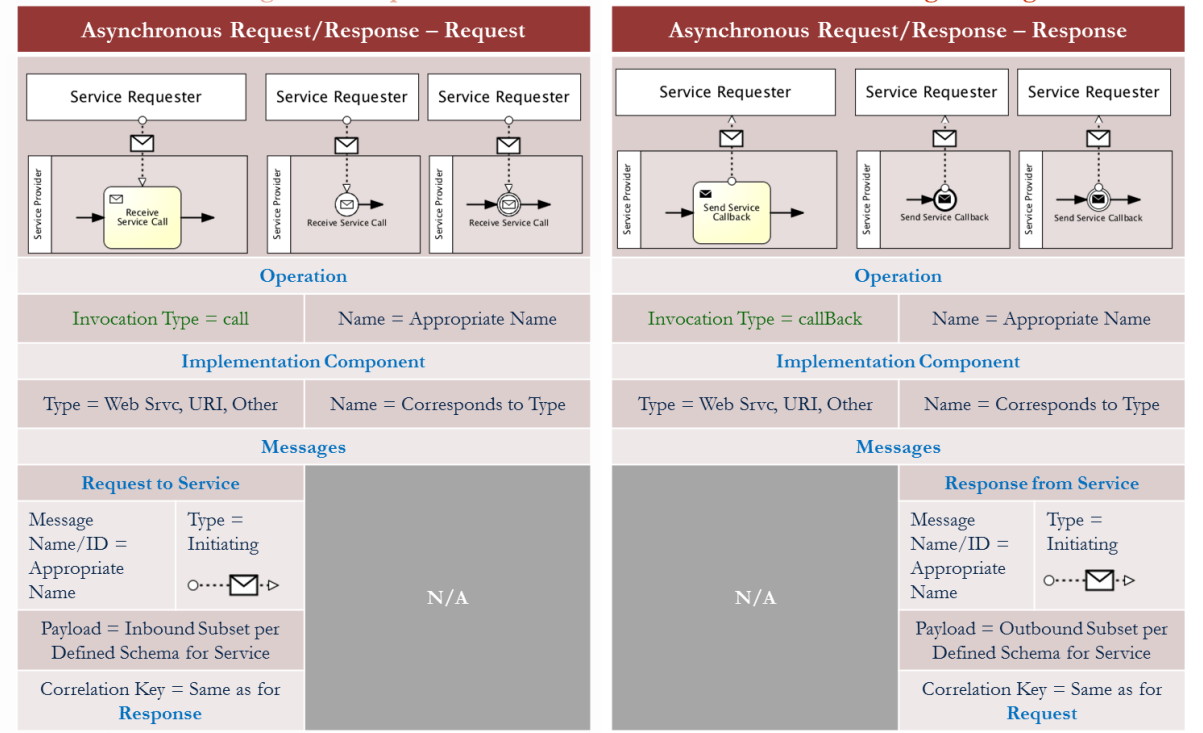


Figure 10 – BPMN Modeling Convention and Attribution Scheme, Asynchronous Request/Response

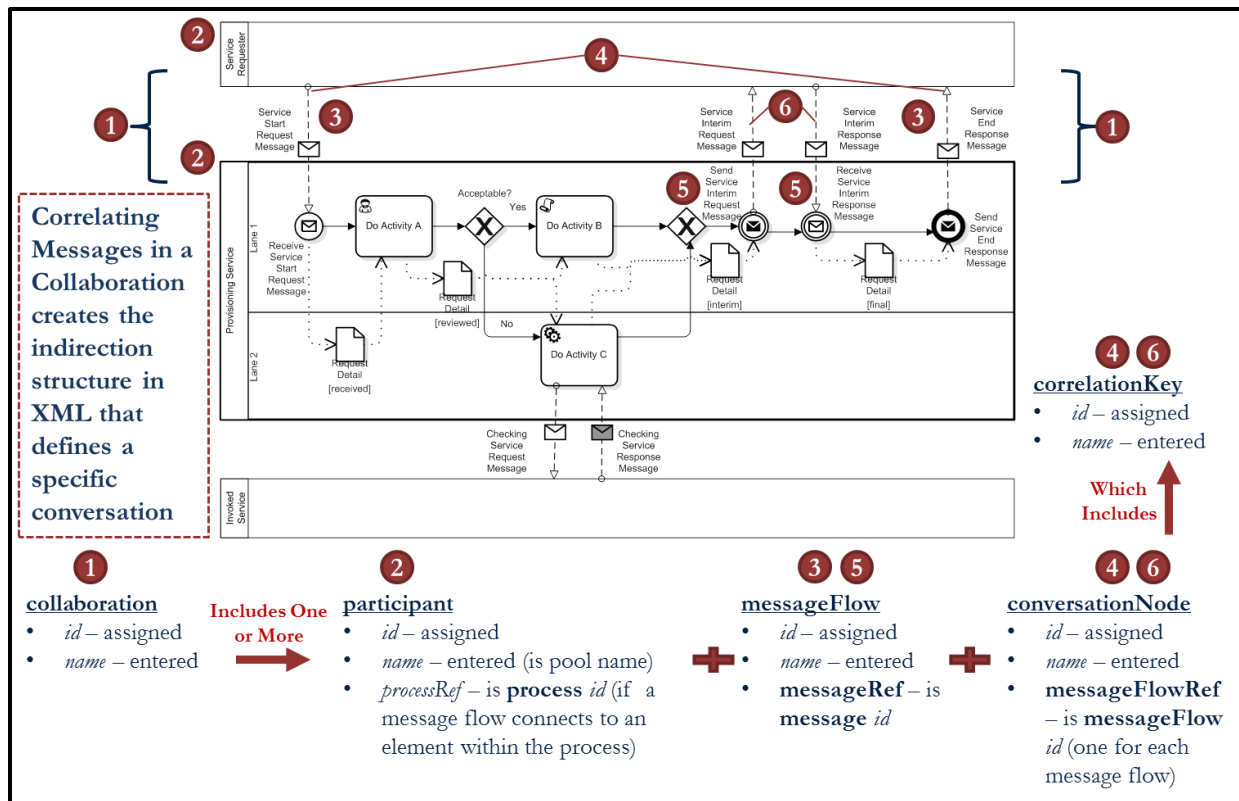


Figure 11 – Asynchronous Request/Response Example

To make the most of representing MEPs, it is necessary to capture or define the Operation Name for the Message Task that is an abstraction or the Message Event that is a service endpoint. It is also necessary to capture or define the Conversation and Correlation Key for the involved Messages. Note that the capability of the modeling tool may constrain how or if this level of attribution can be done, but the these attributes – taken from the Common Execution Conformance Class – are available for use within the BPMN XML. (The other elements represented are already part of the Analytic Conformance Class.)

In addition, the nature of the MEP can also be characterized by the API style that appropriately applies. These API styles match up with the Implementation Reference types mentioned earlier, and are defined more fully at <http://www.servicedesignpatterns.com>:

- Message-based APIs for SOAP-style web services for appropriately-named operations of some complexity
- Resource-based APIs for RESTful services for simply-named operations of Create, Read, Update, and Delete (CRUD) operations (e.g., see POST, GET, PUT, and DELETE)
- Custom remote procedures call (RPC) APIs for proprietary services for specifically-named operations to effect non-standard interactions.

Implementing these API styles amounts to selecting specific implementing component types, which should be a consciously-directed choice of the Solution Architect as opposed to an unintended result.

Summary and Conclusion

A System Pool that is created using the approach described in this paper can be isolated and exported as corresponding BPMN XML. This BPMN XML can be translated into OWL per the BPMN 2.0 ontology created by the DCMO. Combined and mapped to other ontologies, notably domain ontologies that were created pursuant to the naming of core DoDAF Meta Model (DM2) concepts of Capabilities, Activities, Resources, and Performers (aka CARP), this ontological representation of the system's functionality can be measured against the applicable governing architectures using specific SPARQL queries crafted to determine the presence or absence of conforming relationships.

Through such a means, the architectural conformance of proposed or existing (legacy) systems can be ascertained automatically and unequivocally. It is proposed here that DoD systems be designed in this manner and submitted to conformance checking as part of ongoing investment review of IT systems.