Integrating BPMN and SoaML: Part 2. Concepts that guide integration

Jim Amsden (jamsden@us.ibm.com)
Senior Technical Staff Member
IBM

18 November 2014

This article, part 2 of a 3 part series, explores the concepts that guide the integration of BPMN and SoaML. The concepts include how each language approaches encapsulation, contracts (or interfaces), structure and behavior.

View more content in this series

Introduction

The first article in this series, Integrating BPMN and SoaML, Part 1. Motivation and Approach introduced the value proposition for integrating these standards, and showed one way to do this. Integrating the standards allows modelers to:

- Use their complimentary capabilities for modeling behavior and structure to address a broader range of problems
- Support additional stakeholder views
- Provide a means of integrating process and service-centric approaches to development to leverage their unique capabilities.

Part 2 explores the concepts that could be used to guide integration of these standard modeling languages. The concepts include how each language approaches encapsulation, contracts (or interfaces), structure and behavior. Understanding these concepts provides a context for commonality/variability analysis between the standards in order to inform the actual integration. This analysis might provide clarity on the meaning of concepts in each modeling language by explaining the concepts from the perspective of the other language.

Review of Part 1: Motivation and approach

BPMN and SoaML integration should support the construction and use of services through the separation of the architecture of services from the processes that define, implement or use them. With this integration modelers can:

- Identify SoaML Capabilities (or candidate services) from BPMN processes

- Identify, specify and implement services using SoaML
- Use BPMN to define a method to implement an Operation of a SoaML ServiceInterface provided by a Participant
- Invoke a service operation, defined by a SoaML ServiceInterface and provided by a SoaML Participant, as a ServiceTask in a BPMN process
- Share information specifications (classes, data types, message types) between BPMN and SoaML
- Use SoaML to define lifecycles for active data objects in BPMN that can respond to business events

Part 1 also introduced different approaches to BPMN and SoaML integration including no integration, model interchange, and model integration. Model integration is the preferred approach because it focuses more on how these modeling languages can be used together and complement each other to provide a greater value proposition. Each modeling language can be used for what it does best, and the information linked and shared between them. For example, modeling language integration allows vendors to create tools that use both languages together in a standard way, if desired. Model bridging is similar to meta-model integration but is done using a separate mediator/bridge meta-model that results in less coupling between BPMN and SoaML. This allows them to be developed and used separately, too.

Concepts guiding integration

In order to understand how to integrate BPMN and SoaML, it is useful to understand how each specification addresses a set of core concepts. This facilitates understanding the strengths and weakness of each specification and how to use the strengths together, while avoiding the weaknesses.

The core concepts are:

Encapsulation
: A description of the encapsulating element or component including the potential interactions with other prototypical components

Contract
: The encapsulation and specification of the potential interactions or interfaces between components and the agreements the components are expected to adhere to

Structure
: The Internal structure of an encapsulating component including the assembly of other components

Behavior
: A Representation of a component's internal behavioral implementation or orchestration, including how the component uses and/or provides services according to the agreed upon contract

Encapsulation and contract

Encapsulation and contract are best addressed together since encapsulation separates content in different logical units and contracts specify the connections between those logical units.

Encapsulation and contract primarily deal with separation of concerns and commonality/variability analysis – separating out the things that are different and the things that change so they can be handled independently. Measures of encapsulation and contract quality often involve measures of cohesion and coupling. High cohesion and minimal coupling tends to lead to better reuse and solutions that are more resilient, easier to design, implement, test, and maintain.

Encapsulation defines a unit of containment and an interface to that unit of containment. Contracts specify and encapsulate the interactions between encapsulated units. The term component is generally used to describe the unit of encapsulation. Contracts that describe interactions between components involve message exchange patterns that include:

- The specification of the data or information exchanged (the message content)
- The grouping of data exchanged into logical, cohesive units (messages)
- The valid sequences, protocols or choreographies that constrain the order of those messages (the message exchange patterns)
- Acceptable qualities of service and value that result from the exchange

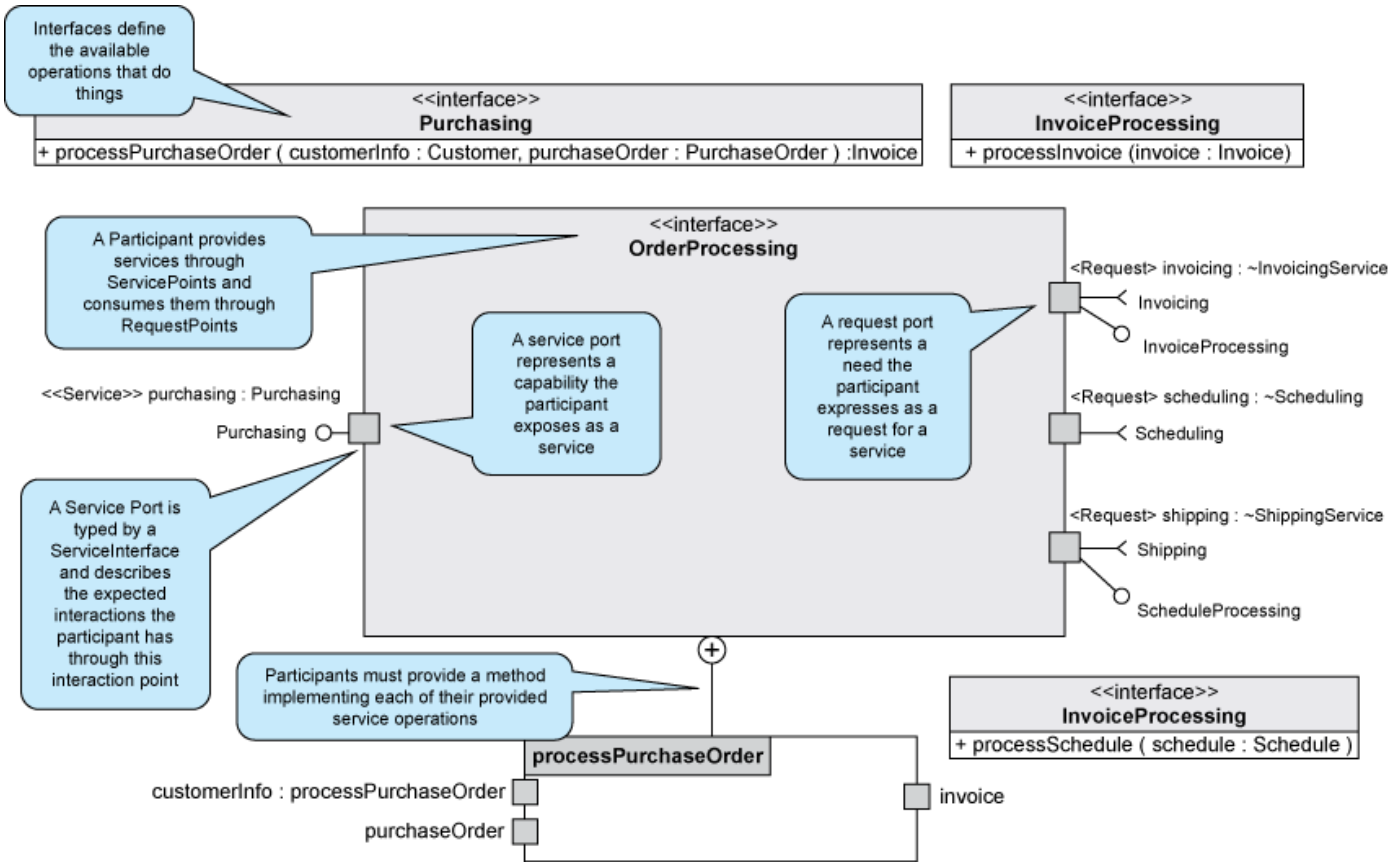
SoaML

The unit of encapsulation in SoaML is the **participant**. Participant is an extension of UML Component which defines a component that provides and/or uses services, independent of how the services are implemented, how they might be used by other participants, or how requested services are used.

Participants have service ports that represent the services they provide and request ports for the services they use from other participants.

Figure 1 shows a typical "black-box" view of a Participant that is independent of any other participant. Information about the processPurchaseOrder method is in the Behavior section of this article.

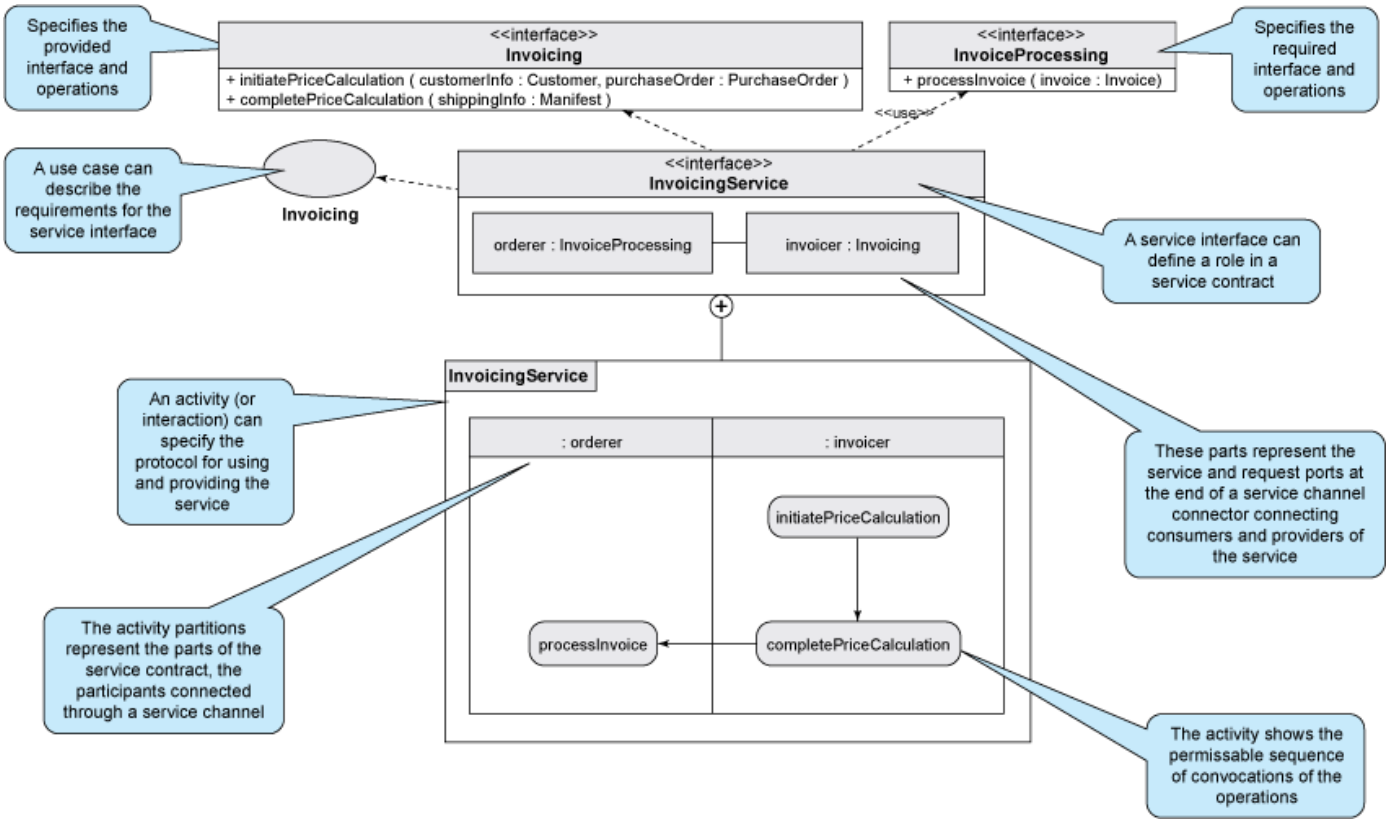
Figure 1. Participant as the unit of encapsulation in SoaML



A ServiceInterface, shown in Figure 2, is used to define the type of a service or request port and specifies the details of the message exchange patterns.

- **Messages exchanges** are defined by the operations and receptions of the provided and required interfaces and their input and output parameters.
- **Message grouping** is defined by the ServiceInterface itself. ServiceInterfaces are used to type service and request ports.
- **Message choreography** is defined by the owned behaviors of the ServiceInterface. The owned behaviors specify the protocol for using and providing the service operations. SoaML also supports ServiceContract as a more flexible means of specifying message choreography involving potentially many participants.

Figure 2. ServiceInteface in SoaML



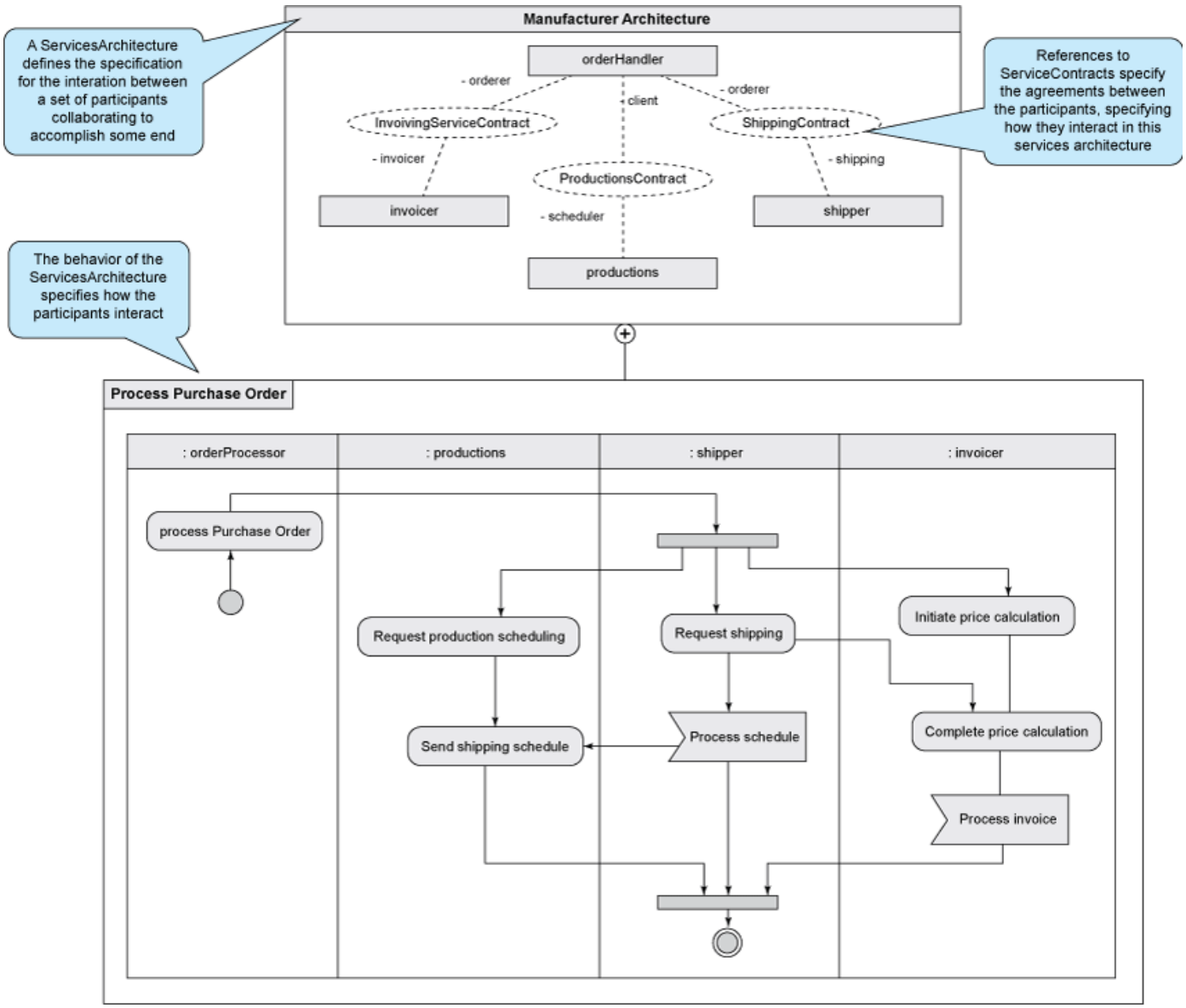
SoaML (and UML) supports both message- and remote procedure call (RPC) oriented service interactions. Message-oriented interaction is done through Receptions in provided and required Interfaces that define the Signals used to model the exchanged message content. Activities use SendEventAction and AcceptEventAction to actually send and receive the events. Operations in Interfaces and CallOperationAction and AcceptCallAction in UML Activities support RPC-oriented interactions. These are actually normalized in UML run semantics through CallEvent, which is sent by a CallOperationAction.

At a higher-level, a SoaML ServicesArchitecture describes the relationships between a set of Participants playing roles and interacting according to agreed upon ServiceContracts to achieve some desired result. See the developerWorks article "Using SoaML services architecture" for further details.

The Manufacturer ServicesArchitecture

As shown in Figure 3, the Manufacturer ServicesArchitect shows how orderhandler, invoice, productions and shipper components collaborate based on agreements captured in ServiceContracts in order to process a customer order, produce the products and ship them. The Process Purchase Order Activity describes the order of the activities in the collaboration, and shows how the service operations are intended to be used to achieve the desired result.

Figure 3. ServicesArchitecture and ServiceContract in SoaML



BPMN

In BPMN, the unit of encapsulation is also the Participant. This is represented by a Pool. A BPMN Collaboration diagram describes the messages exchanged between the participants.

BPMN uses a number of concepts to describe the interactions between participants and pools including orchestration, collaboration, conversation and choreography.

Orchestration
: The sequencing of Activities that represent the sending or receiving of a message, the use of a service operation, or a task performed by a participant

Collaboration
: The specification of messages exchanged between participants

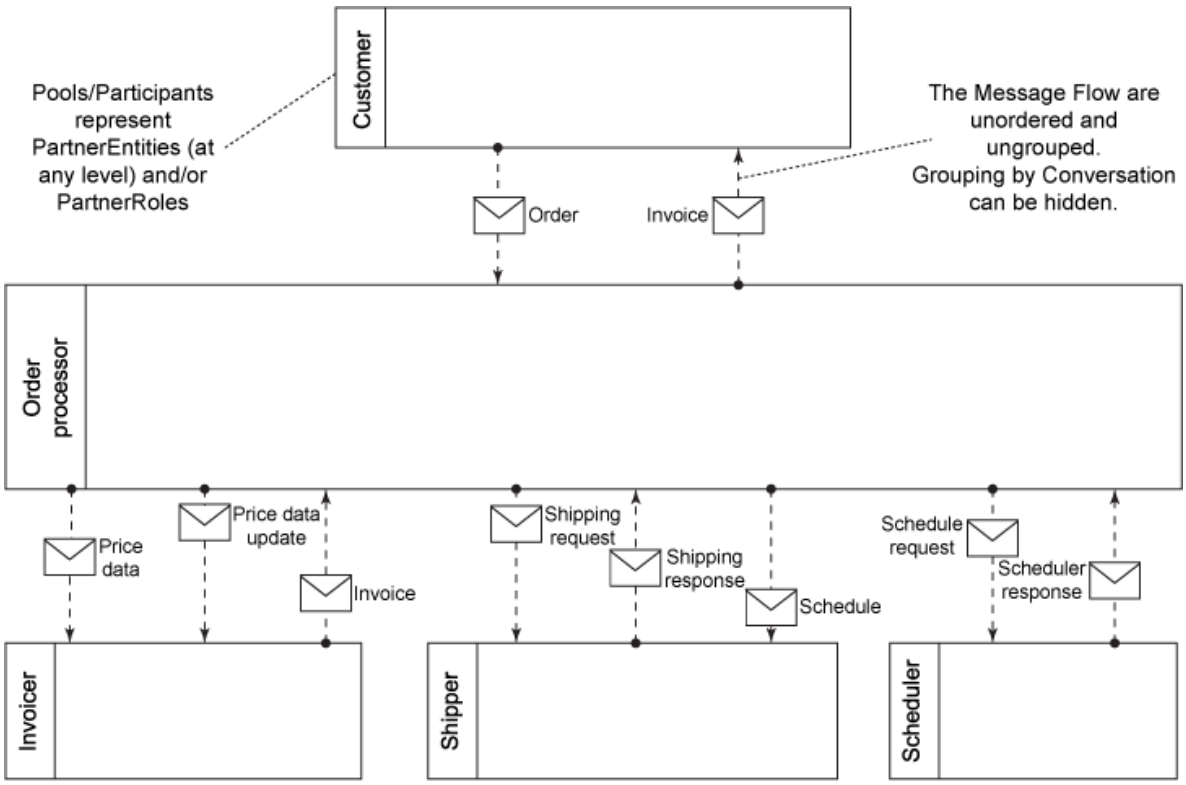
Conversation
The grouping of those messages into logical units for the purpose of correlating the interactions between particular instances of participants at runtime

Choreography
The specification of valid sequences of messages between participants

Service Interface
The specification of service tasks that are provided by one participant and used by another.

Figure 4 is a collaboration diagram. It is used to provide an ungrouped view of the messages exchanged between the participants.

Figure 4. BPMN collaboration

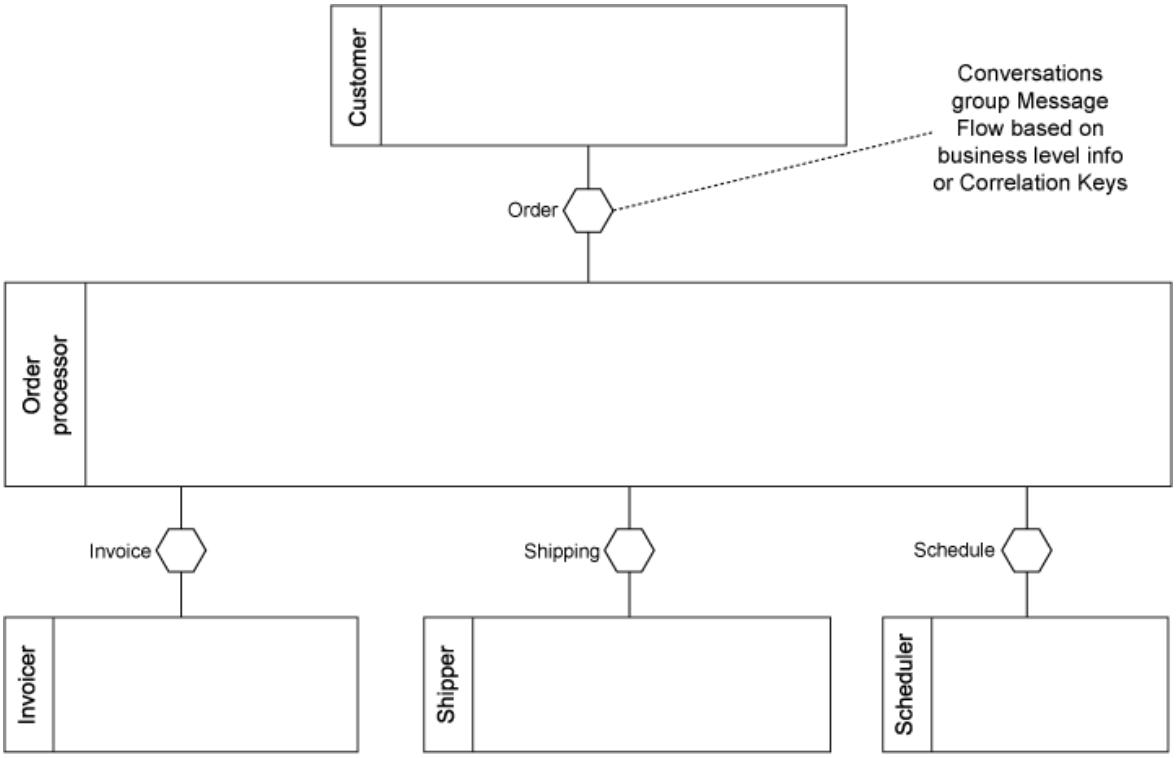


BPMN also supports service tasks. Service tasks can refer to an operation of a service interface. This allows BPMN to also support message-based or RPC-based interaction styles. The meaning of the service task is to call an operation that is defined by the service interface. This is the same as a CallOperationAction in UML Activities. BPMN does not specify how these service interfaces relate to message correlation, conversations, or choreography. Instead, it treats them as different approaches used to model similar concepts. BPMN also does not distinguish how different participant pools interact through provided and used service interfaces. Rather this is done through binding information on the services themselves.

In BPMN, messages are grouped using Conversation. The Conversations also provide the ability to specify correlation information that can be used to distinguish different instances of the participant in some run environment. BPMN does not distinguish between type and instance. It treats a participant pool as a prototypical instance. This has some implications if you reuse

participants in other collaborations. For example, it is not clear whether the contexts in which a participant is reused are inclusive or exclusive. In practice this doesn't matter that much, as modelers can assume that a participant could be involved in any or all of the interactions in which it is included in BPMN models. Figure 5 shows a conversation in BPMN. The model doesn't necessarily specify which participants are necessary for any particular context and that has to be left to additional documentation.

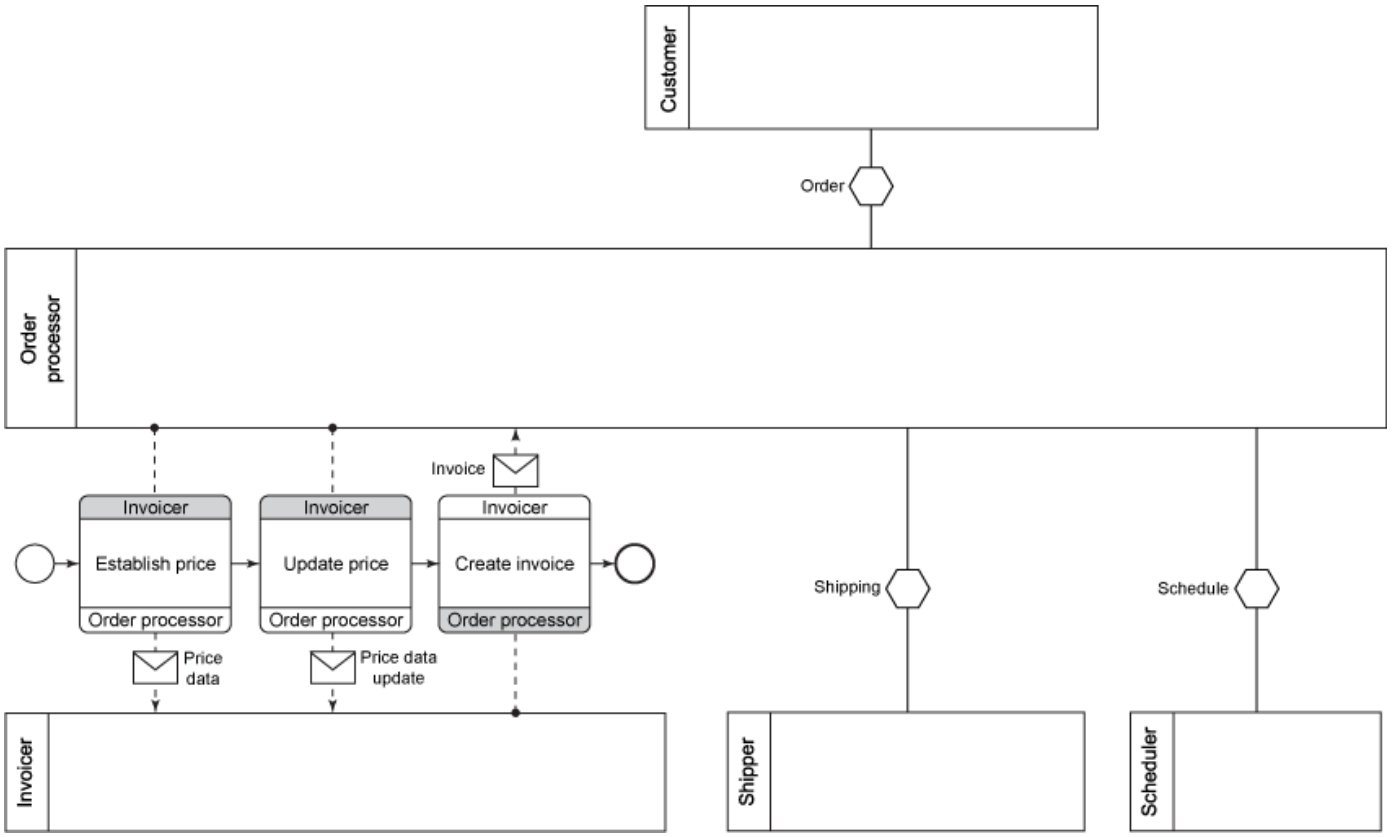
Figure 5. BPMN conversation



The messages contained in a Conversation are not shown on a Conversation diagram. They are however listed in the model through other mechanisms (such as properties views).

A Choreography diagram shows message choreography in BPMN. A BPMN choreography shows the valid sequences of messages exchanged between the participants. Figure 6 shows how to sequence message patterns that are part of a conversation.

Figure 6. BPMN choreography



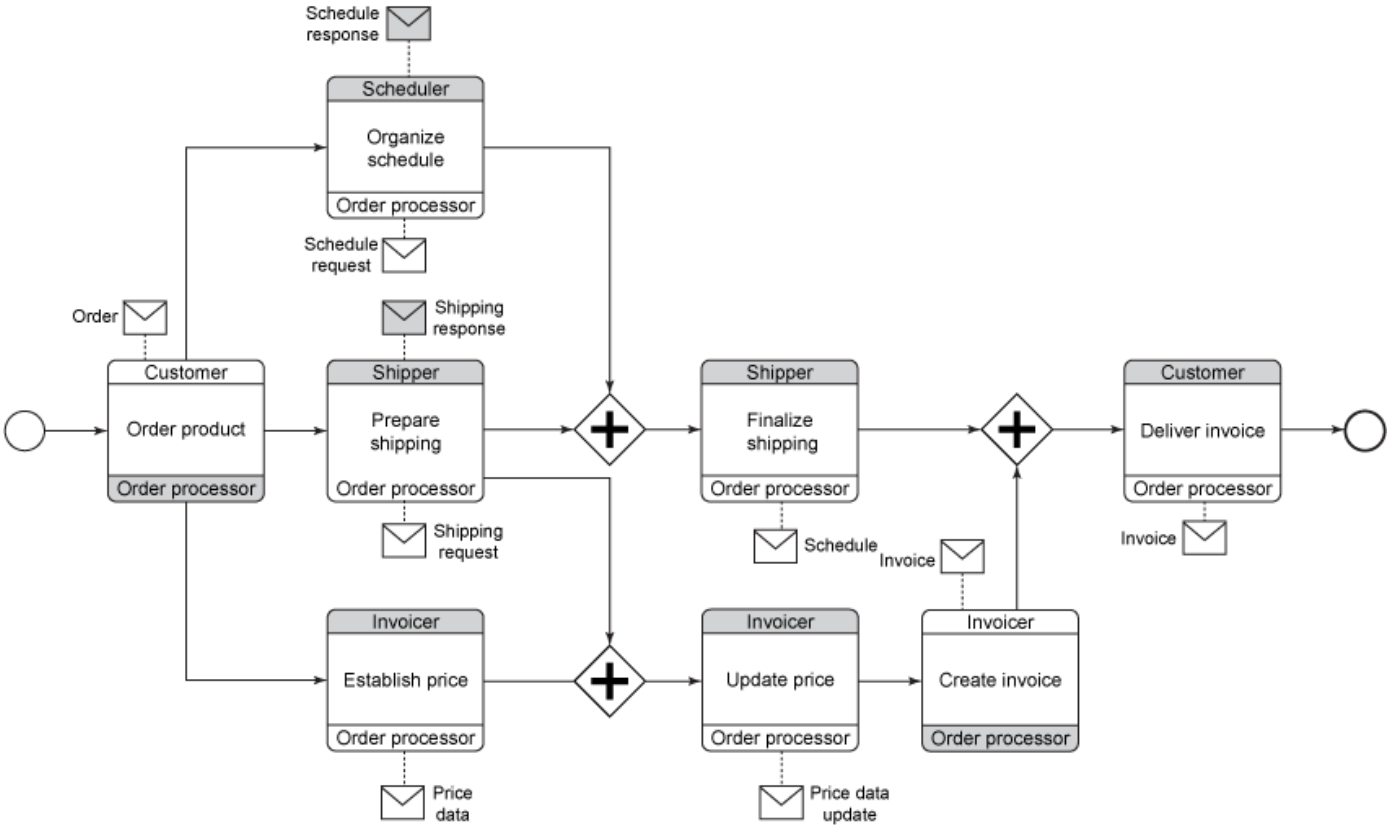
A conversation that groups a set of correlated messages can have an associated choreography that specifies the message sequence or protocol.

BPMN does not currently have any notation that shows messages exchanged, message grouping in conversations, message protocols using choreography in the same diagram That information is available in the BPMN model and can be shown through tool-supported properties and navigation between related diagrams.

A BPMN choreography can also show the overall sequencing of the messages between all participants as they collaborate to achieve some desired result.

Figure 6 shows the choreography of the entire purchase order process, not the choreography of a particular interaction between participants involving a particular conversation. This combined choreography can be developed early in the modeling process to identify the key participants and how they interact. It plays a similar role as the SoaML ServicesArchitecture to show the intended interactions between participants according to their agreed upon service contracts, and the overall activities required to achieve the desired result.

Figure 7. Use choreography to describe architecture



In summary, message patterns in BPMN are modeled as:

- **Messages exchanges** are defined by a collaboration diagram showing messages between pools representing participants
- **Message grouping** is defined by a conversation diagram that groups messages shown on collaboration diagrams in order to correlate interactions between specific instances of participants in an execution environment
- **Message choreography** is defined by a choreography diagram that shows the sequencing of the messages between participants involved in a conversation

Structure

Structure defines how a system is assembled from parts. It describes how encapsulated components can be constructed from other connected encapsulated components according to defined contracts. Structure defines how parts are assembled and connected so that interactions between them can occur. Structures can be static, assembled as the result of intentional design. Or structures can be dynamic, discovered and assembled and disassembled at runtime in some environment and driven for some purpose.

Structure is defined at either the class level or the instance level. For the class level, structure is defined as the description of a set of related elements. At the instance level it is defined as connections between parts representing instances of some defined classes. These definitions are the source of some modeling confusion. In the past, UML 1.x did not clearly distinguish between

type and instance in the meta-model. So modelers would often use class diagrams to specify a context. The classes and associations on the diagram identified the structure of connected parts in that context. This had problems because diagrams are views, not models, and conveyed implied rather than explicit semantics. It also had implications for reuse. Since connections between parts were defined at the class level with associations, all instances of a class had to include all associations from all diagrams in the model semantics. This limits the ability to reuse the class for other purposes without potentially pulling in unused associations.

UML 2 addressed this issue by introducing a StructuredClassifier that explicitly defines the notion of connected parts that are typed elements. However, the practice of using class diagrams to represent prototypical instances, and to use dependency "wires" to show assembly of parts at the class level, still persists and is also directly supported by UML 2. The rest of this article attempts to be precise about the distinction between a type and the instances of that type used in the structure of an assembly for some specific purpose.

SoaML

Structure in SoaML is modeled as the internal structure of a participant. The parts of a participant are typed elements that represent references to instances of other participants. The parts are connected or wired together through ServiceChannels. ServiceChannels connect the service and request ports of the parts in a manner that is (optionally) consistent with a ServicesArchitecture.

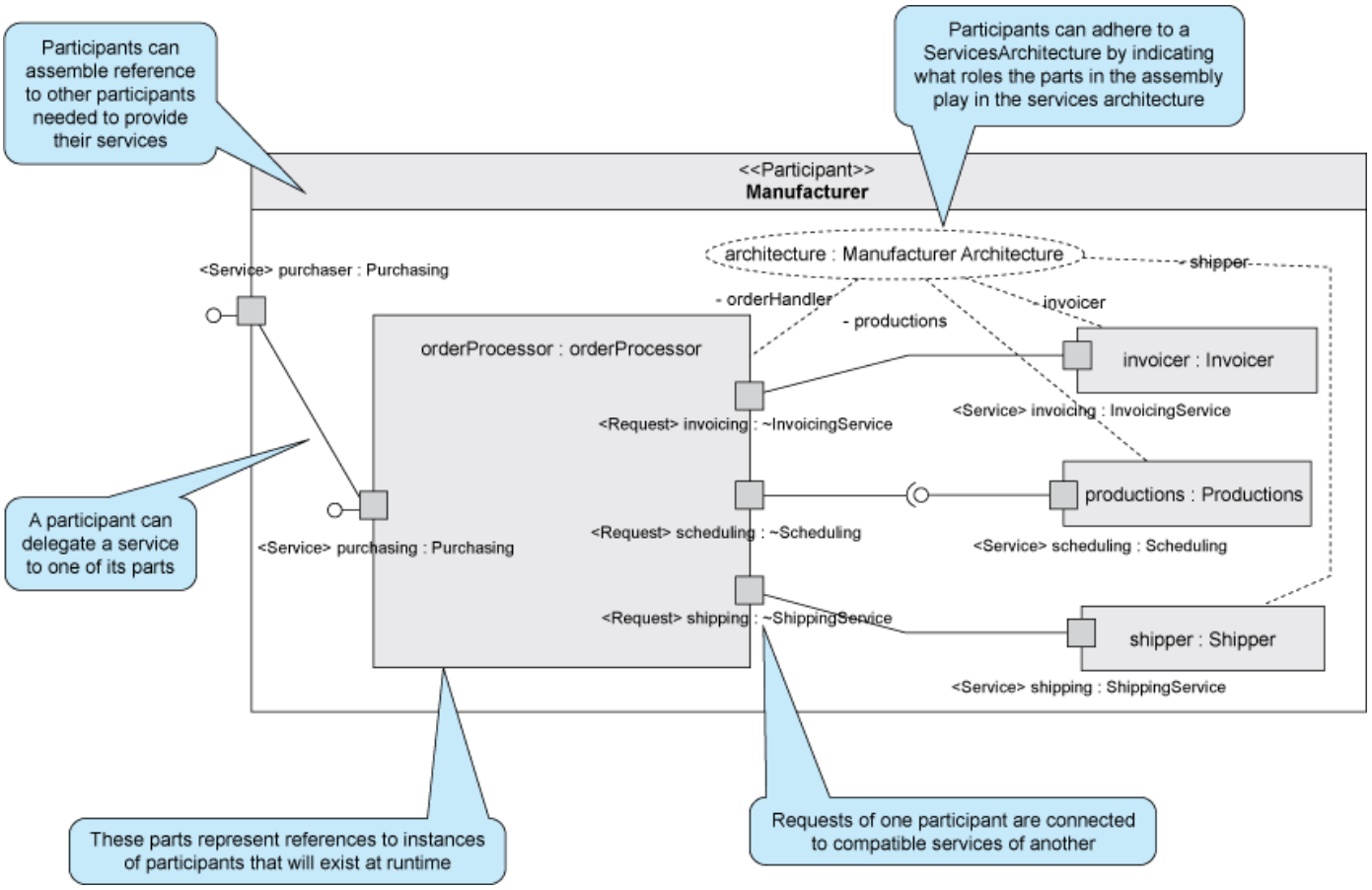
In Figure 8, the Manufacturer Participant has an internal structure that consists of orderProcessor, invoice, productions, and shipper parts that are connected by service channels. The service channels connect the provided and used services. They also provide a channel through which the service requests can flow.

A SoaML Participant can utilize its internal structure as a means of providing and using services. In Figure 8 the Manufacturer participant provides the Purchasing service by delegating it to one of its parts using a delegation connector.

The architecture CollaborationUse in Figure 8 depicts an instance of the Manufacturer Architecture, which was described in the Encapsulation and contract section. The role bindings indicate how the Manufacture participant adheres to that services architecture. Each part must be type compatible with the role it is bound to in the services architecture. This provides traceability between design and implementation in the model.

It is possible to utilize a class diagram and dependency wires to show similar information. The dependency wires would be between the service and request ports of the participant classes. This indicates the design intent for the service interchanges and is not the preferred approach, so it is not described in further detail.

Figure 8. Service assembly

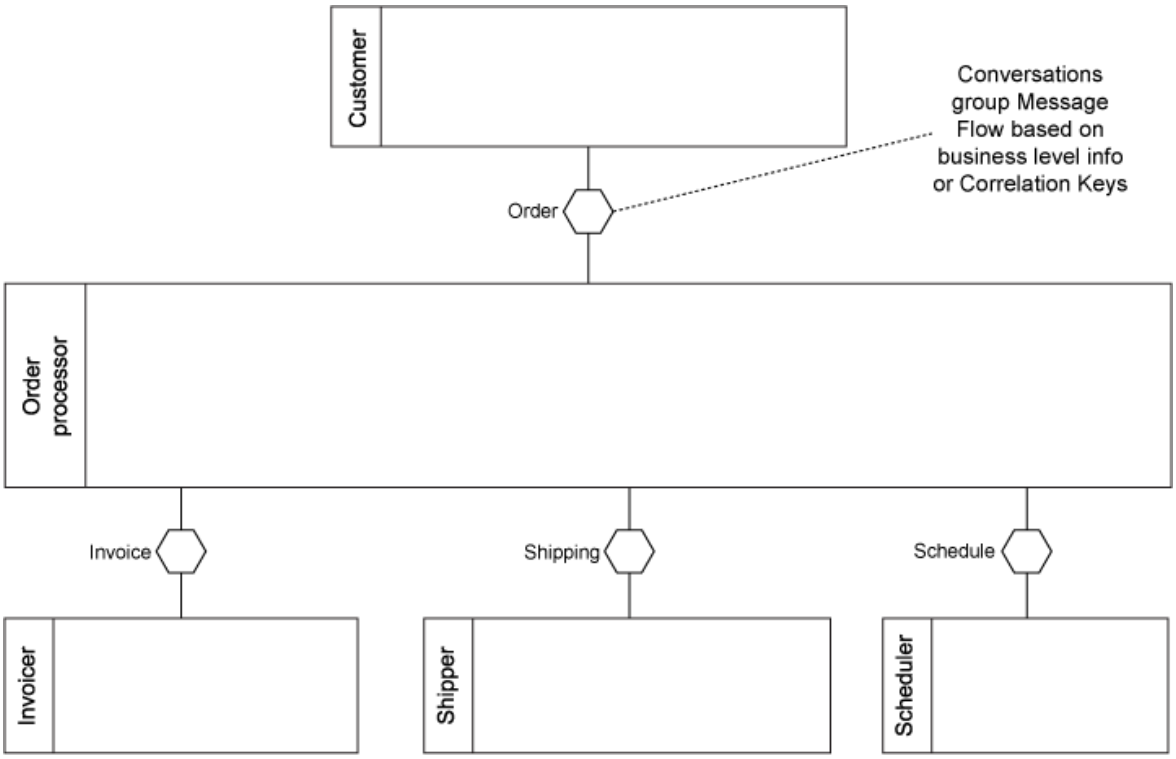


BPMN

BPMN does not distinguish between type and instance. BPMN defines process components and their implementations. The relationships between these process components are modeled using conversations, possibly elaborated by collaborations and choreography. Conversations can include correlation information used at runtime to distinguish different instances of participants in potentially long-running processes. This correlation information is not available for defining typed elements in diagrams.

In Figure 9 the conversation diagram provides the context for the assembly of participants required to process an order. Each participant pool represents a prototypical instance that could exist in some runtime environment where the process actually runs

Figure 9. Process assembly using BPMN conversation



Example of the limitations

The Scheduler participant in Figure 9 provides Schedule capabilities used by the Order Processor participant as shown in the conversation that connects them. A Scheduler could also be reused to schedule classes for students at a university, for example. That use could be shown on another conversation diagram. What does this mean? Does it mean that any Scheduler must be connected to an Order Processor and a University Class in order to function properly? Of course this is not the case. But, a Scheduler might require services from other participants in order to function. And these connected participants are required for all uses of a Scheduler. The BPMN models can't distinguish between these two situations because BPMN doesn't distinguish between type and instance.

In practice this is not a big problem. You can create different diagrams, or separate BPMN models, for the Scheduler participant. These indicate which conversations and choreographies are required for it to function in a particular context. You can create other BPMN models to show different usages of the Scheduler for different purposes. Modelers distinguish these difference usages through the different models and their contexts and have little problems distinguishing between what is required for all instances and what is required in a particular context.

This is a case where BPMN and SoaML do not overlap and SoaML provides complimentary capabilities. BPMN can describe the prototypical instances in conversation diagrams, and SoaML can define the specific assemblies of those instances in a participant's internal structure. SoaML and BPMN could therefore be combined, similar to the relationship between W3C Service Component Architecture (SCA) composites and Business Process Execution Language (BPEL) partner references.

Behavior

UML and BPMN can describe behaviors for various purposes including:

- Elaborating requirements
- Ordering sequences of activities to accomplish some result
- Describing how an entity changes its state over time
- Describing actual, anticipated or desired sequences of activities
- Implementing and using services

Behavior is further classified as executing behavior - activities performed by an object itself, and emergent behavior - sequences of activities that result from the interactions of one or more participating objects.

Behaviors can be viewed from different perspectives. Public processes represent observable behavior as seen from the outside. Internal processes represent those behaviors that implement and use services. Internal processes are often hidden in order to avoid exposure of private, proprietary, or competitive information.

SoaML

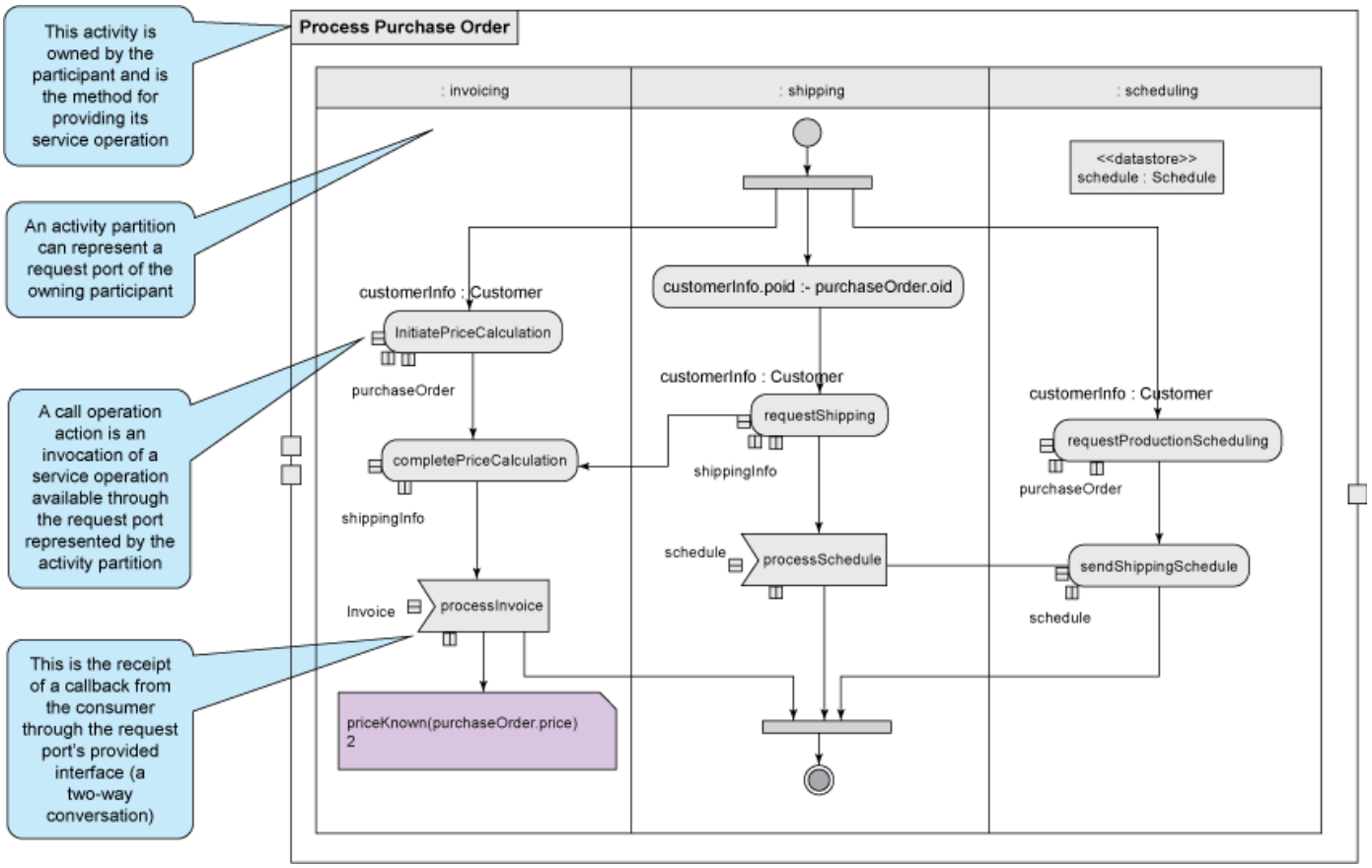
UML, and therefore SoaML, has a number of different kinds of behaviors including:

- Activities
- Interactions
- State machines
- Opaque behaviors (e.g., code in some language including UML Action Language)

These different approaches are generally semantically equivalent, and represent different notations that modelers or stakeholders might find convenient. For the purposes of integration with BPMN, the focus of this article is UML activity models.

Figure 10 shows a typical UML activity that is owned by a participant, and is the method for an operation provided by that participant. The section Encapsulation and contract shows the OrderProcessor participant owned a processPurchaseOrder behavior, omitted to hide unnecessary details. That behavior is shown in detail in figure 10.

Figure 10. UML activities implement and use services in SoaML



An ActivityPartition in UML can represent anything, just like lanes in BPMN. However, UML2 defines a number of special activity partitions that have specific semantics. The partitions shown are "part" partitions because they represent ports of the owning classifier. This means that the CallOperationActions in the partitions must have their target What object they are going to be invoked onInputPin set to the part the partition references. In this case, that is the request port of the participant. This diagram omits the details of the target input pin because the information is more easily shown in the activity partition.

The CallOperationActions in the activity partitions call service operations provided by other participants. These will eventually be connected to the request ports through a service channel (a UML Connector). These called operations are specified in the required interfaces of the port, indicating what operations the activity might call through that port.

In longer running, two-way conversations, the activity might have to respond to callback functions, or operations defined by the service interface that must be provided by the participant using the service. In Figure 10, the Invoicing service provides operations for calculating invoices, but requires any user of this service to actually process the invoice. This is modeled using an AcceptCallAction in the invoicing partition. The target input pin for this action is also the invoicing port. This action is executed when a participant provides the Invoicing service then calls back the order processor in order to process the invoice.

UML activities also allow input pins and other actions to refer to the activity parameter nodes and any data stores in the activity. For example, the schedule data store is referenced on the input pin of the processSchedule AcceptCallAction.

BPMN

BPMN supports a number of different capabilities for process modeling that can be used to support different stakeholder needs for understanding and reasoning about processes, and different modeler needs for capturing and elaborating process details. A more services oriented view of these different BPMN capabilities has orchestration similar to SoaML behaviors. Collaboration, conversation and choreography are views of different aspects of the encapsulation and contracts that formalize the interactions between participants.

Orchestration
: Defines the sequences of activities and tasks in a process that implements and possibly uses services

Collaboration
: Describes the messages exchanged between consumers and providers as they interact to achieve some result

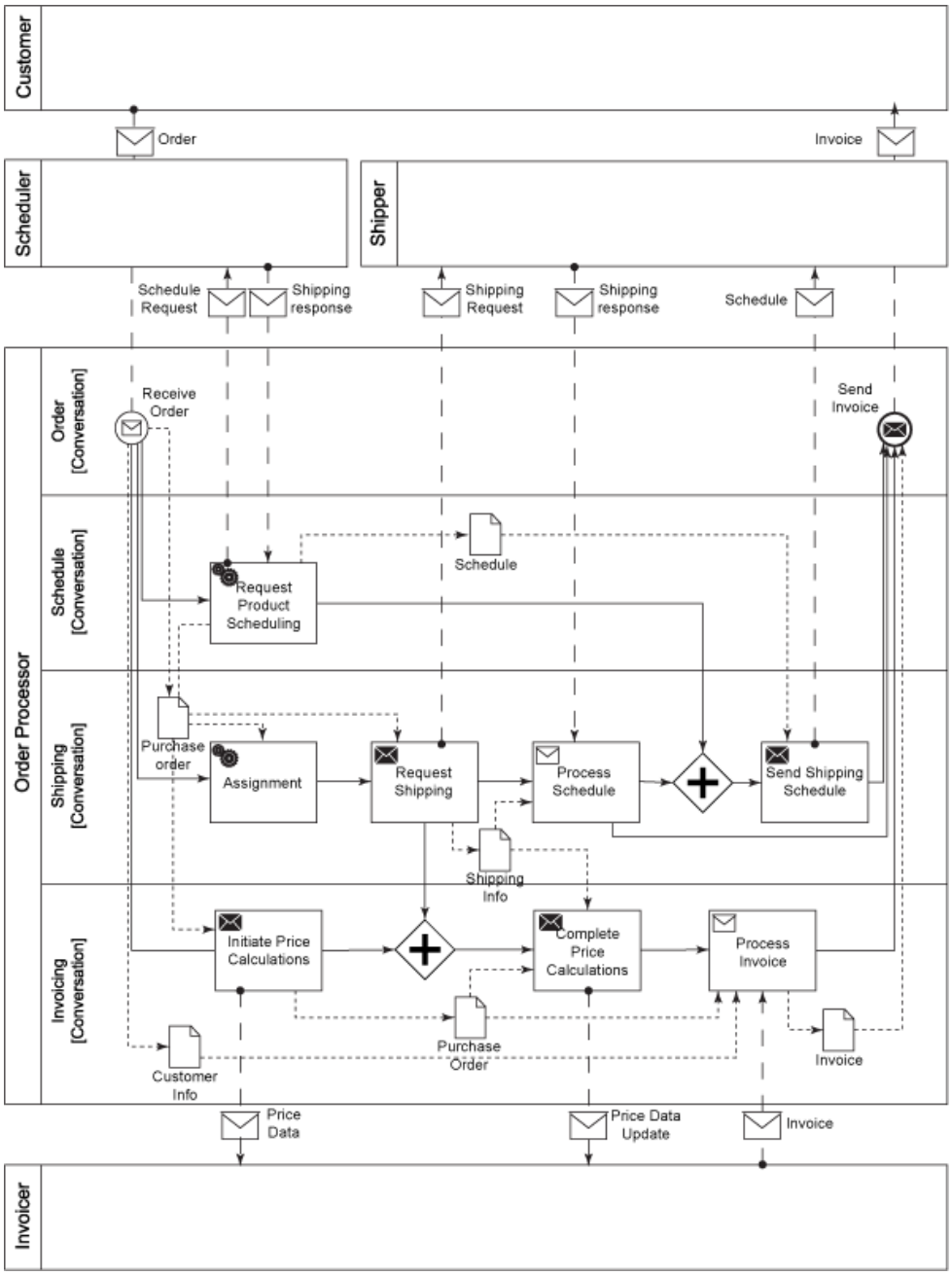
Conversation
: Describes the organization of the messages, encapsulating the interactions between participants, and providing information needed to identify and correlate the particular participants involved.

Choreography
: Specifies the valid interactions between collaborating participants including the sequencing of the message exchange patterns.

BPMN provides many diagrams that capture behavior concepts. Figure 11 shows a collaboration diagram elaborating the Order Processor participant shown in the previous collaboration and communication diagrams. It shows the internal processes of the Order Processor participant (represented by a Pool), and the messages that are exchanged with other participants. Service Task Initiate Price Calculations represent the invocation of a service operation. The Price Data message indicates the service task parameters.

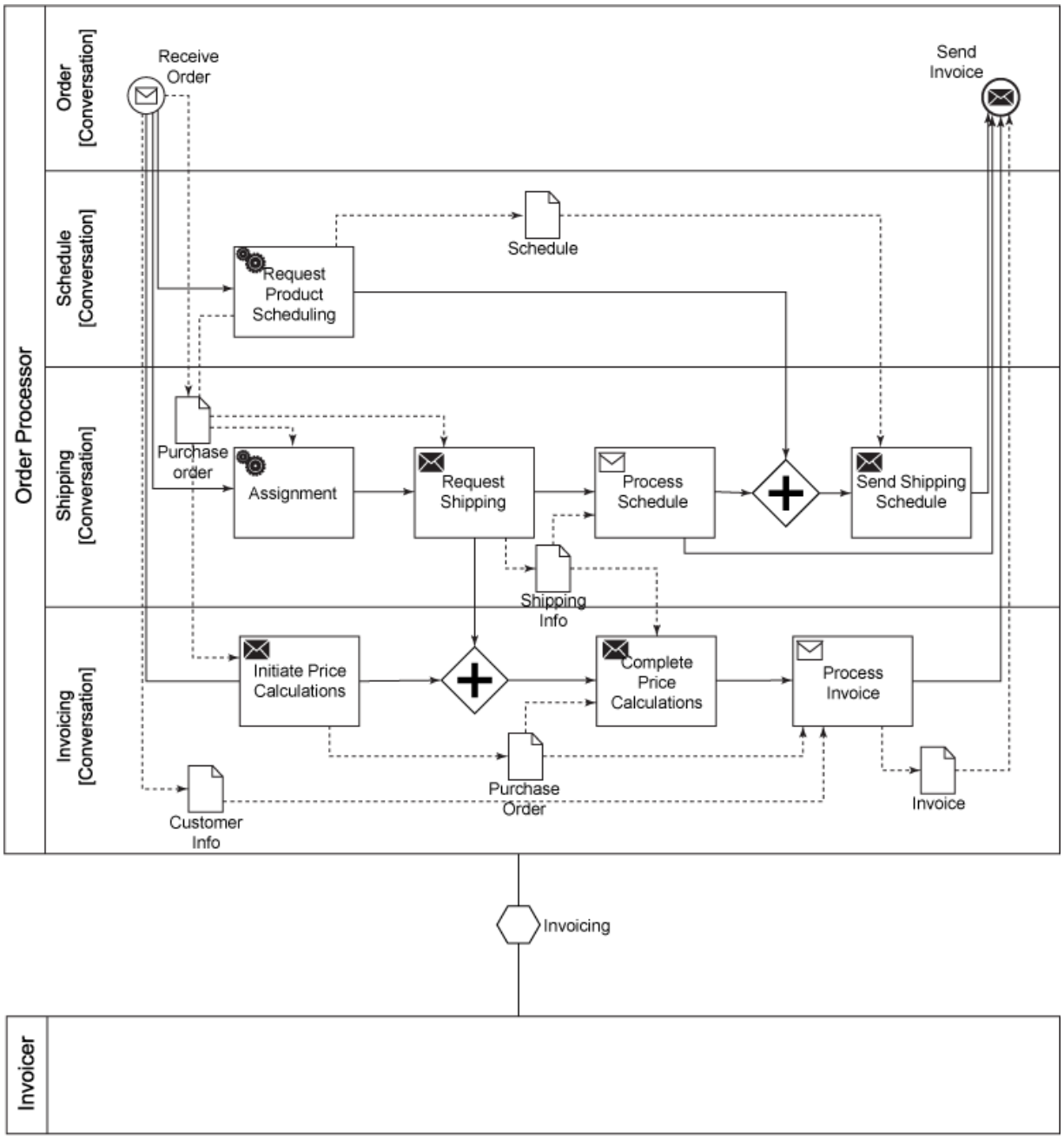
Similar to UML ActivityPartictions, BPMN Lanes in a pool can represent anything. In Figure 11, the lanes in the Order Processor represent the conversations through which the messages are exchanged, and in accordance with the choreographies that describe those conversations. The Conversation defined for the Collaboration can be used to create the Lanes.

Figure 11. Processes implement and use services in BPMN



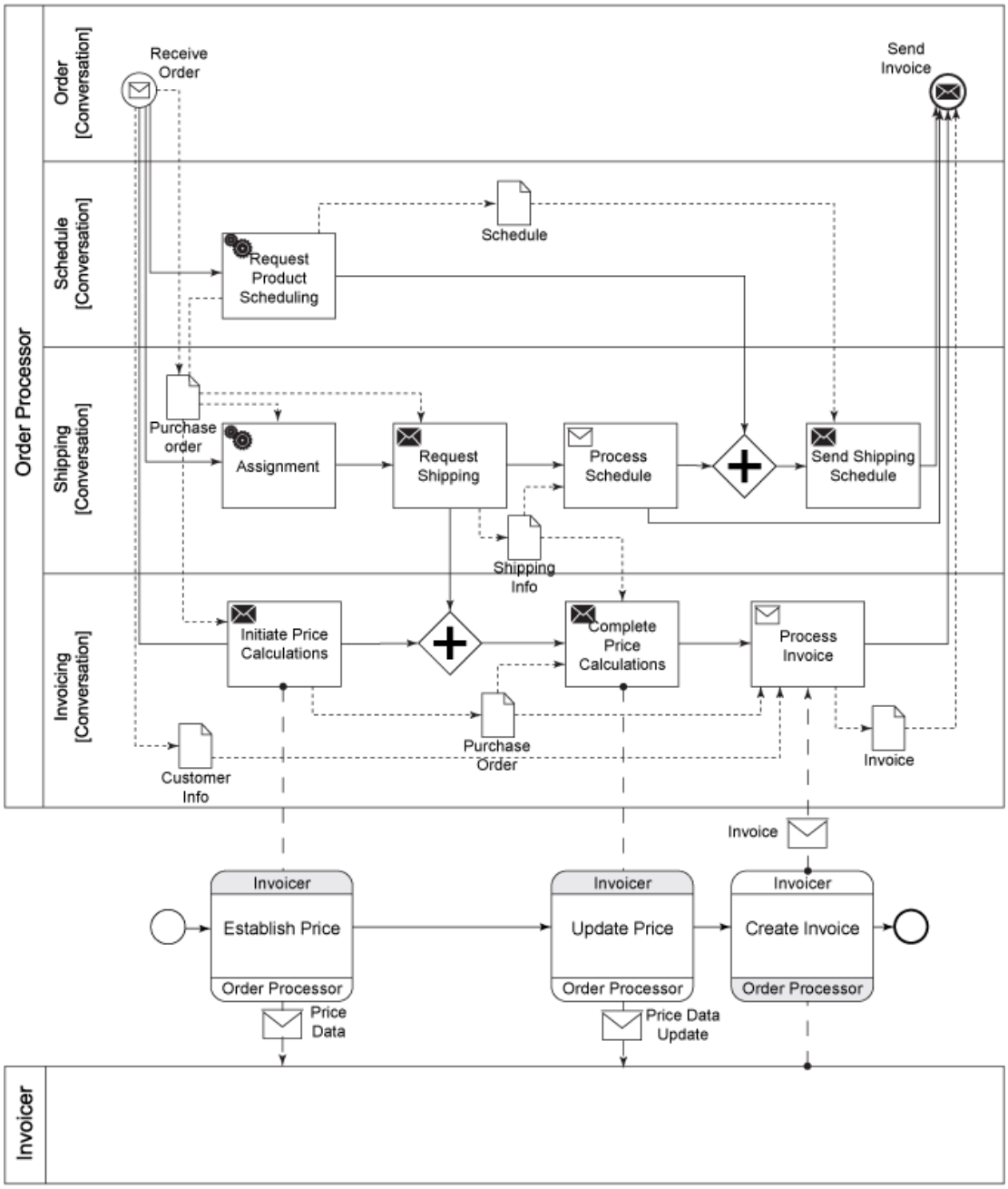
For example, the Order Processor calls the service Initiate Price Calculations through the invoicing Conversation by sending a Price Data message and receiving an Invoice Response message. All Activities in that lane could be considered associated with the interactions between an Order Processor and an Invoicer according to the Invoicing conversation. As shown in Figure 12, you can see that the order of these activities is consistent with the choreography of the messages defined for that communication.

Figure 12. Associating lanes with conversations in BPMN



The Service Tasks in the invoicing swim lane refer to an operation of an Interface. The operation has parameters that are its input and output messages. These messages must somehow correspond to the messages defined by the collaboration and conversation that describes the interaction between the participants. Figure 13 shows an expansion of the Invoicing conversation shown in Figure 12 as a choreography that captures the sequencing of the exchanged messages.

Figure 13. Connecting BPMN conversation and choreography



The associations between the lanes, activities in the lane (including service tasks and their defining interfaces), messages, conversations and choreographies are not shown on any (single) BPMN diagram in this article. However, this information can be captured in model element properties. The use of lanes to represent conversations is not formally specified in BPMN, but is a useful modeling convention.

Conclusion

BPMN and SoaML are two important standards adopted by OMG. There is significant overlap between these standards, each providing a particular emphasis. BPMN focuses on process

models, starting with simple orchestration of activities in a business process, and expanding to support collaboration between processes, and then expanding again to support choreography of the interactions between processes. SoaML focuses on services architectures and the encapsulation of interactions between participants in a services architecture using service contracts to model service-level agreements (SLA). This article explores SoaML and BPMN support for concepts such as encapsulation, contracts, structure and behavior in order to understand the similarities and differences between the languages. The next article, Integrating BPMN and SoaML Part 3: Mapping BPMN and SoaML, will demonstrate:

- How to map the common elements across the languages
- How to use this mapping to guide the use these two modeling languages, either separately or together, on the same or related projects
- How to do so in a way that leverages their unique, complimentary features while avoiding redundancy resulting from their overlap.

Resources

Learn

- Integrating BPMN and SoaML: Part 1. Motivation and Approach (Jim Amsden, developerWorks, July 2014): First article in this series, introduced the value proposition for integrating these standards, and showed one way to do it.
- "Modeling SOA: Part 1. Service identification" (Jim Amsden, developerWorks, October 2007): First in a series of five articles about developing software based on service-oriented architecture (SOA).
- Modeling SOA: Part 2. Service specification (Jim Amsden, developerWorks, October 2007): The second in a series of five articles about developing software based on service-oriented architecture (SOA).
- Modeling SOA: Part 3. Service realization (Jim Amsden, developerWorks, October 2007): The third article of this five-part series explains how SOA-based Web services are actually implemented. The service implementation starts with deciding what component will provide what services. After these decisions have been made, you can model how each service functional capability is implemented and how the required services are actually used. Then you can use the UML-to-SOA transformation feature included in IBM Rational Software Architect to create a Web service that can be used in IBM WebSphere Integration Developer to implement, test, and deploy the completed solution.
- Modeling SOA: Part 4. Service composition (Jim Amsden, developerWorks, October 2007) The fourth article of this five-part series covers how to assemble and connect the service providers modeled in "Part 3. Service realization" and choreograph their interactions to provide a complete solution to the business requirements. It also shows how this service participant fulfills the original business requirements.
- Modeling SOA: Part 5. Service implementation (Jim Amsden, developerWorks, October 2007) The fifth article in this series shows how to create an actual implementation that is consistent with the architectural and design decisions captured in the services model.
- Service-oriented modeling and architecture: How to identify, specify, and realize services for your SOA (Ali Arsanjani, developerWorks, November 2004) is about the IBM Global Business Services' Service Oriented Modeling and Architecture (SOMA) method.
- IBM Business service modeling (Jim Amsden, developerWorks, December 2005) This article describes the relationship between business process modeling and service modeling to achieve the benefits of both.
- "Design and develop a more effective SOA, Part 1: Introducing IBM's integrated capabilities for designing and building a better SOA" (developerWorks, May 2011): First article in a five-part series on IBM's commercial solution for service-oriented systems design and development. This article begins by discussing some of the promises and issues associated with moving to a service-oriented approach for IT systems. It then provides high-level descriptions of best practices and tools for realizing the benefits and overcoming the issues.
- OMB BPMN Specification and OMG SoaML Specification: Learn more about these specification by downloading them in PDF format.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download the trial version of IBM Rational Software Architect.
- Download IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.
- Rational Software Architect, Data Architect, Software Modeler, Application Developer and Web Developer forum: Ask questions about Rational Software Architect.

About the author

Jim Amsden

Jim Amsden is an IBM Senior Technical Staff Member and Solution Architect. Jim works on a broad range of solutions incorporating IBM products for municipal strategic planning; solution analysis, design and construction; operational efficiency and monitoring; and business intelligence - tying these together into a comprehensive offering supporting the IBM Smarter Cities initiative. Jim has spent many years developing standards and products for modeling to understand and manage complex systems. He holds a Masters degree in Computer Science from Boston University and a Bachelors degree in Mathematics from the University of Maine. His interests include semantic web technologies and information integration and sharing, Enterprise Architecture, contract based development, agent programming, business-driven development, JEE, UML, and service-oriented architectures. He is co-author of Enterprise Java Programming with IBM WebSphere and an author of the OMG SoaML Specification, a standard for modeling Service Oriented Architectures. Jim has also done extensive research and development in the area of mathematical models for power generation optimization for paper mills. He is a TOGAF-Certified Enterprise Architect.