Processes

Process Modeling

In the previous chapter, we have seen that BPEL provides a set of standard constructs to define the behavior of an orchestration at run-time. These constructs are specified in XML and do not have a graphical representation. However, integration platforms that are based on (or at least inspired in) BPEL often provide development tools where an orchestration can be designed by resorting to graphical elements (or shapes) that represent those constructs.

Clearly, BPEL is a standard that is geared towards execution, by defining orchestrations as a series of Web service invocations. Given a BPEL orchestration, which is essentially an XML document, it is possible to have an execution engine that parses the XML and runs each activity according to the behavior of each BPEL element, as defined in the BPEL standard [19]. In this case, we could say that such execution engine is BPEL-compliant. However, such engine can work only if a fully specified BPEL orchestration is provided. In other words, the BPEL orchestration is an executable model of some business process that has been fully characterized, to the point that it can be run by an execution engine. This is the reason why BPEL stands for Business Process Execution Language.

In practice, in order to arrive at such executable model, it is necessary to understand and design the business process. This usually involves a significant amount of effort, as a process model is created and then changed and refined over several iterations. For this purpose, it is necessary to have appropriate tools— namely, a graphical process modeling language—to represent the process in an intuitive way, so that it can be easily understood and manipulated by business analysts. When finished, such design model (as opposed to executable model) will serve as the blueprint for system integrators to implement the process as a set of one or more services and orchestrations.

Just like there is a standard (i.e., BPEL) that defines the constructs that can be used for execution, there is also a standard to define the graphical constructs that can be used for designing business processes. This standard is called BPMN (Business Process Model and Notation) and it has succeeded in gathering the support of most IT vendors. There are, of course, other process modeling languages, such as Petri nets and EPCs, but here we are interested in the fact that BPMN is not only a standard, but is also a language that shares at least part of its conceptual foundations with BPEL. This makes of BPMN a useful tool to design process models which can be translated into executable orchestrations.

The purpose of this chapter is not to provide an exhaustive presentation of all BPMN features,1 but to describe the typical structure of BPMN process models and to highlight the similarities between BPMN constructs and BPEL constructs. The correspondence is not one-to-one, but the concepts that underlie some BPMN constructs are very similar to the original purpose of some BPEL constructs. In most cases, it will be possible to figure out a way to implement a given BPMN model with BPEL constructs. This is quite interesting from a practical point of view, since it becomes possible to bridge the gap between the process models developed by business analysts (typically, using BPMN) and the integration solutions that are required to implement those business processes (e.g., using BPEL).

Elements of Process Model

Activities

In BPMN it is possible to decorate activities with an icon to provide some more information about how the activity is to be performed. Figure 11.2 shows the different types of activities available in BPMN 2.0. These can be describing as follows:

Fig. 11.2 Activity types

• A service task is an automated activity that consists in the invocation of some service or application.
• A send task is an activity that consists in sending a message to an external participant. In Fig. 11.1 there was an activity called “Send purchase order” which could have been represented as a send task.
• The receive task is the counterpart of the send task. Basically, it represents an activity whose main purpose is to receive a message. The activity is completed only when the message has been received.
• In addition to the regular receive task, BPMN 2.0 includes the concept of instantiating receive, which is equivalent to the concept of activating receive in orchestrations
• The manual task is intended to represent an activity that is to be performed without IT support. This could be any action in the physical world that is not monitored or supported by an IT system.
• The user task represents an activity that is assigned to some user. Typically, this task will be sent as a work item to the user’s worklist, and the execution engine will be waiting for an output or completion message before resuming the process.
• The script task contains a series of instructions that are to be carried out by the engine that will be executing the process. When the engine reaches the script task, it will execute the code contained therein. For this purpose, the script must be written in a language that the engine is able to interpret and execute.
• Finally, the business rule task is used to invoke business rules. This is equivalent to the call rules shape that was briefly mentioned earlier. Basically, business rules can be used to perform calculations or make decisions based on user-defined parameters. The reason why these rules are not embedded in the process is that they can be changed at any time according to business requirements. The business rule task is a means to invoke an external business rules engine that will evaluate the rules and return the results back to process. The process can then use these results to decide how the process should proceed.

Loops and Multi-Instance Activities

Fig. 11.3 Loop activities

Figure 11.3 shows how these concepts can be represented in BPMN. On the lefthand side there is a loop activity, meaning that the activity will run an arbitrary number of times before the process can

proceed to the next activity. The number of iterations is determined by some condition that is to be evaluated at run-time. This condition can be included as a attribute of the activity, but it does not have a graphical representation in BPMN.

The next two shapes, in the middle and on the right-hand side of Fig. 11.3, represent the multi-instance concept. Here, the activity is seen as being instantiated multiple times, where each instance is independent from every other. These instances may run in parallel or in sequence, with the sequential multi-instance being a recent addition in BPMN 2.0.

Although it may seem that the loop activity and the sequential multi-instance may be the same, there are some subtle differences between the two. In the loop activity, there is an exit condition that is evaluated after each run. The loop may continue or be exited depending on the particular circumstances that occur when the condition is being evaluated. On the other hand, for the sequential multi-instance the number of instances is known at the start of the activity, and the process can only proceed when all of those instances have been completed.

Usually, the multiple-instance activity, either in parallel or sequential form, is used when there is a collection of objects to be processed independently of each other. In this case, each instance of the activity is intended to handle a different object. The loop activity, on the other hand, is a means to keep an activity running until some condition is true. This may not necessarily involve a different object in each iteration. In fact, the loop may be run over the same object until the state of that object changes, or some other condition becomes true.

Sub-processes

Another interesting possibility is to define an activity as a subprocess. This means that an activity becomes a placeholder for some process logic that one may want to insert at that point in the process. Figure 11.4 illustrates how a subprocess may appear in a BPMN process model. There are two forms: either collapsed or expanded. If collapsed, the subprocess looks like a regular activity except for the plus sign (i.e., “C”) indicating that it contains additional process logic. If expanded, the subprocess shows the logic that is contained inside it. Such logic must follow the same design principles as a top-level process, so usually it contains a start event, a sequence of activities, and an end event. It is only when the subprocess reaches its end event that the parent process can proceed to the next activity.

Just like an ordinary activity, a subprocess may be executed multiple times in the flow of the parent process. In particular, the subprocess can be executed as a loop, as a parallel multi-instance, or as a sequential multi-instance, as shown in Fig. 11.5. In case the subprocess is expanded, it should keep its decoration (i.e., the loop sign), as shown for the expanded loop subprocess in Fig. 11.5.

A particular type of subprocess that has a much different behavior from the rest is the ad-hoc subprocess. This is a kind of subprocess that is not bound to the typical, well-structured behavior of a sequence flow. Basically, an ad-hoc subprocess contains a set of activities that can be executed in any order. In particular, there is no restriction on when each activity can begin and end, so the ad-hoc subprocess can be also regarded as a block where everything can run in any order, including in parallel.

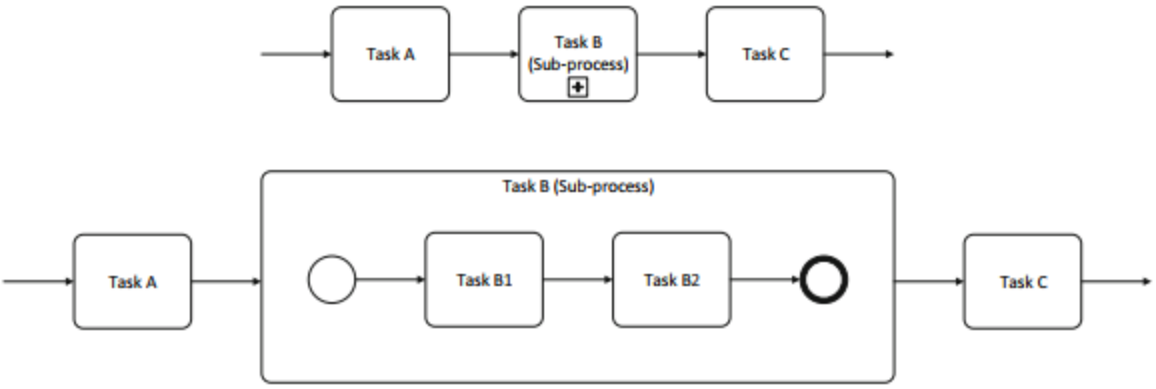


Fig. 11.4 Subprocess in collapsed and expanded forms

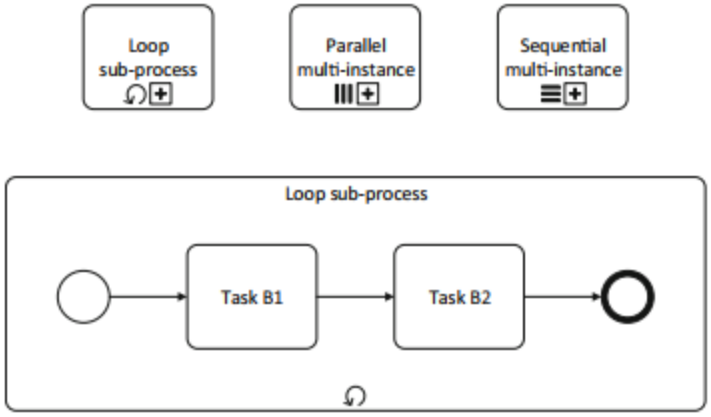


Fig. 11.5 Loop subprocesses

Gateways

At a certain point in the process of Fig. 11.1 there is a decision between two branches. If the product is available, it will be dispatched directly from the warehouse; otherwise, it will be ordered from a supplier. The element that was used to represent this decision is a gateway.

Each gateway represents a different behavior and has its own symbol. A gateway without a symbol is assumed to be an exclusive gateway. This means that one and only one branch must be chosen. To make things clearer, the exclusive gateway can also be drawn with a symbol (an "X" that stands for XOR, i.e., exclusive-OR). Therefore, the exclusive gateway has two possible representations.

The logical counterpart of the exclusive gateway is the parallel gateway. This means that all branches are to be followed in parallel. Usually, regardless of the type of gateway that is being used, each gateway that splits the flow in multiple paths is matched by another gateway of the same type that merges those paths back into the main flow. Here, parallel gateway has been used, but the same principle applies to other gateways too.

In the case of the parallel gateway, the merging gateway is especially important because it works as a

synchronizing merge, i.e., the process will not move on to the next activity until all parallel branches coming into the merging gateway have completed. For the exclusive gateway, this merging works in a different way: as soon as one branch is complete, the process can proceed. Since, in the exclusive gateway, only one branch can be chosen, it does not make sense to wait for the other branches; these are simply skipped.

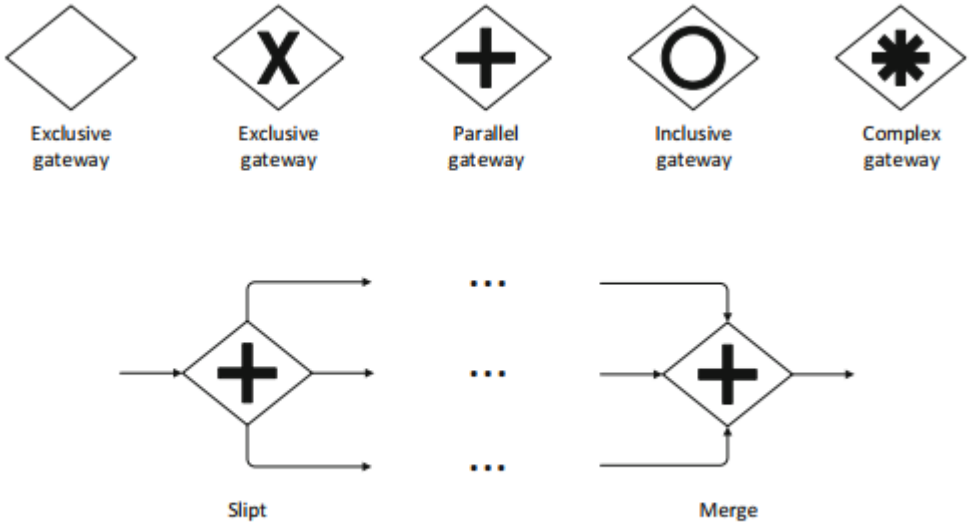


Fig. 11.7 Types of gateways

Exception Handling

In BPMN there are several different ways to represent exceptions, and there are also different ways to include behavior that is specifically targeted at handling those exceptions. The most commonly used constructs to represent exceptions are intermediate events attached to the boundary of activities. Typically, if such an event occurs, the activity is interrupted and the process follows a different path. These attached events are very useful when modeling business processes, but it is not always easy to map them to an execution language such as BPEL, since the flow that is associated with attached events may not follow a nested block structure.

A more traditional solution to the problem of representing exceptions in BPMN are error events. This is a special type of event that, like other intermediate events, can be either thrown or caught, and therefore it is possible, with relative ease, to map these error events to the exception handling mechanisms of BPEL.

On the other hand, escalation events can be seen as a different form of error event. In fact, they do not represent an error in the sense of a system error, but a condition (i.e., a business problem) that occurs during the execution of a business process, and that requires some special handling. In particular, an escalation event means that someone with higher responsibility (such as, e.g., a supervisor) will be called to intervene, or at least will be notified. This is quite useful when modeling business process in organizations with some form of hierarchical structure. However, the semantics that are associated with escalation events do not have much impact from an execution point of view. In this regard, an escalation event is not much different from an error event, except that it can be non-interrupting.

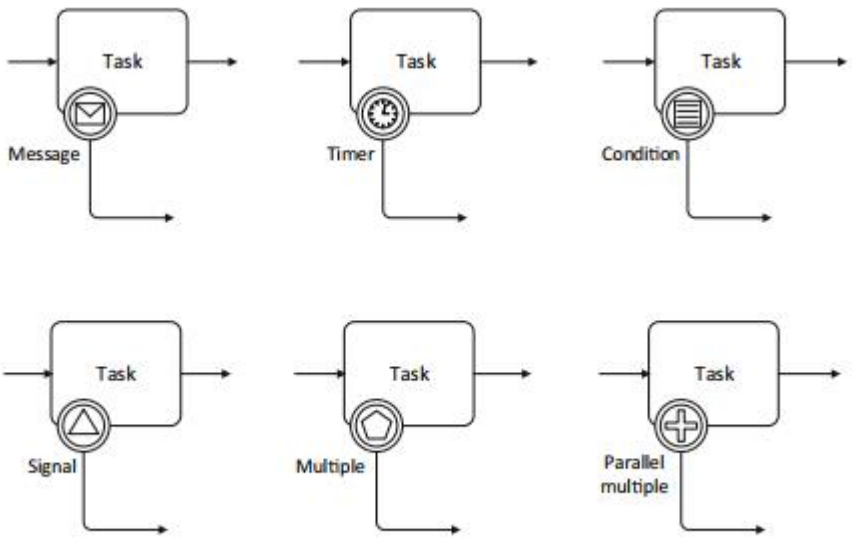


Fig. 11.13 Interrupting intermediate events

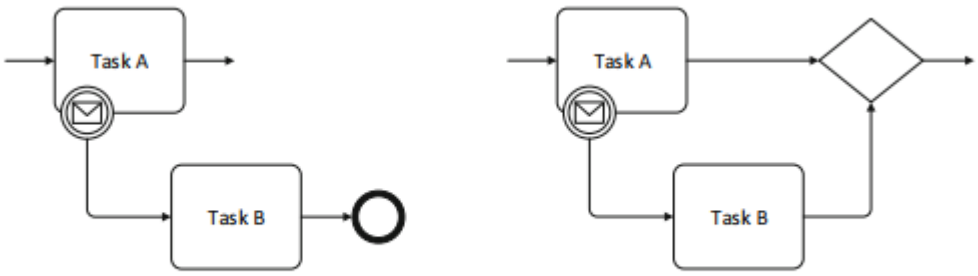


Fig. 11.14 Possible routings for the exception flow

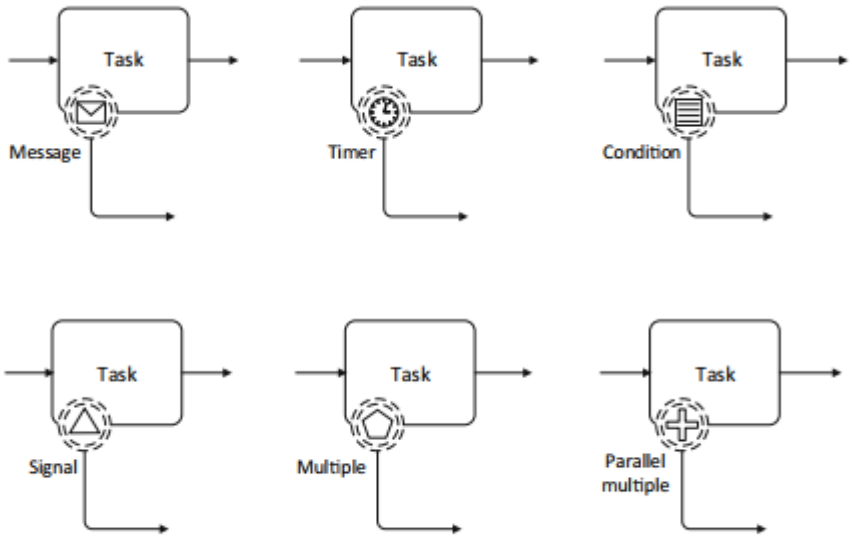
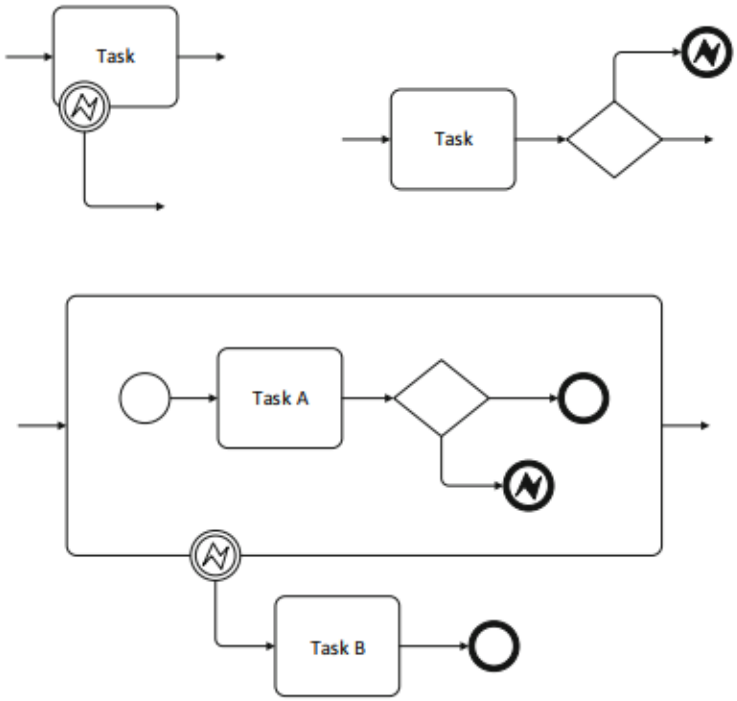


Fig. 11.15 Non-interrupting intermediate events

Fig. 11.16 Possible uses of the error event



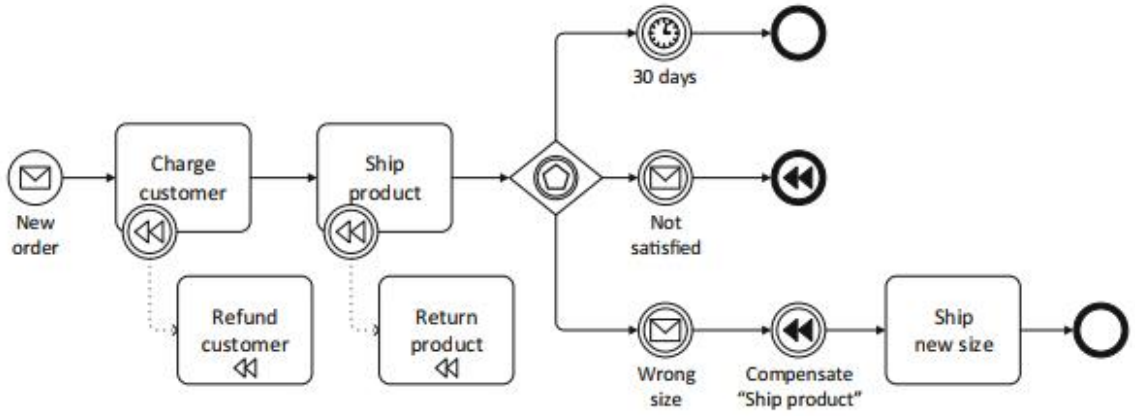


Fig. 11.25 Use of throw compensation events

Inter-Organizational Processes

Namely, a process can be defined using a modeling language such as BPMN, and it can be implemented through an execution language such as BPEL, provided that the necessary services are in place. These services may be the lowest-level services that represent the underlying systems and applications, or they can be higher-level services that are built as compositions of lower-level ones, and these compositions can also be implemented through BPEL.

Figure 12.1 shows an abstract example of a choreography involving two organizations. Here, organization A has an internal process, and organization B has its own internal process as well. These are the private workflows in this scenario. On the other hand, there is a public workflow that consists in the following message exchanges: M1 from A to B, M2 from B to A, M3 from A to B, and M4 from B to A. This sequence of message exchanges is the choreography between these two organizations, and their internal processes have been designed in such a way as to comply with the role of each organization in the choreography.

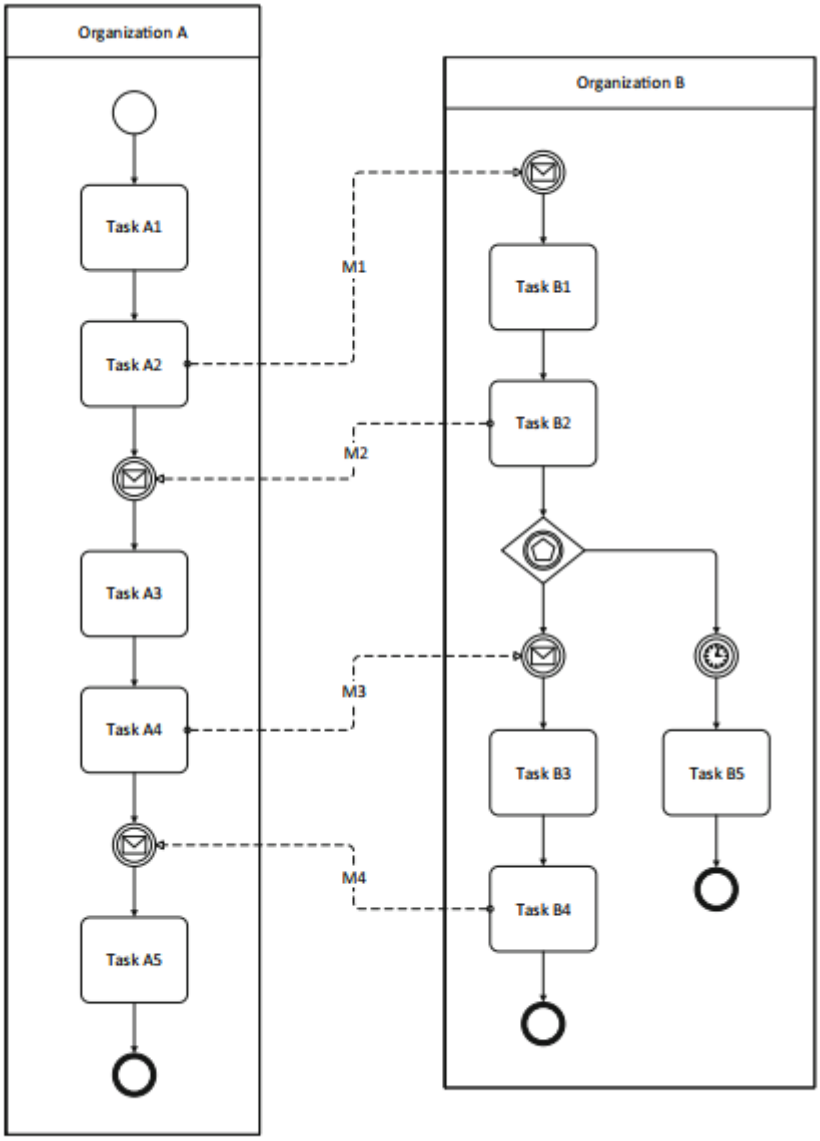


Fig. 12.1 A choreography as a set of inter-organizational message exchanges

Naturally, the internal process of each organization may contain more behavior than what is specified in the choreography. For example, tasks A3 and A5 in organization A, as well as tasks B1 and B3 in organization B, do not appear to contribute directly to the choreography; rather, they are private activities that have to do with the purpose of each internal process. Also, the internal process at organization B includes an additional possibility: the possibility that M3 is not received within a certain time frame, as indicated by the intermediate timer event to the right of the event-based gateway. In this case, task B5 will be performed, and the process of organization B will end without waiting further for M3. After that point, if organization A tries to send M3, organization B will no longer be ready to receive it. This illustrates just how a choreography may not work properly if partners do not have a precise agreement as to what is their expected behavior.

In addition to the choreography, there are other concerns that must be taken into account in an inter-organizational scenario. One of such concerns has to do with the format or structure of the messages to be exchanged. Within an organization, it is possible to define the message schemas to be used in a given orchestration, according to the services and applications that are to be invoked in that orchestration.

However, in an inter-organizational setting it becomes very difficult to achieve an agreement on the message format, since each business partner will have its own processes and requirements. For example, consider message M1 in the choreography of Fig. 12.1: who should define its format, is it organization A which produces it, or is it organization B who consumes it? Organization A can argue that it can only produce the message in certain format, while organization B can argue that it cannot handle the message unless it is in another, different format.

The solution to this problem is to rely on well-accepted standards. Fortunately, there are standards such as EDI (Electronic Data Interchange) which specify the format for a wide range of documents (such as invoices and purchase orders) that are typically exchanged between organizations. Rather than trying to impose some proprietary format on business partners, it makes more sense to ask for compliance with a common standard such as EDI. This way an organization that goes through the effort of implementing a standard message format will at least know that it will become interoperable not only with the current business partners but also with any potential partner that adheres to the same standard. In general, having the message formats defined by an independent third party facilitates business relations by relieving organizations from having to set up their own formats.

Another important concern that applies to inter-organizational processes is security. When business partners exchange messages over the network, they would like to ensure that such messages are authentic, that they have not been modified while in transit, and also that they have not been read by anyone else other than the intended recipient. This is usually attained through the use of digital certificates and public-key cryptography. Here, too, the role of a trusted third-party— specifically, a certification authority (CA)—is essential in order to provide each business partner with its own digital certificate, and to allow a message recipient to check that the incoming message comes from a trusted source.

Security

At the lowest level of an inter-organizational exchange there is the need to establish a secure connection between both ends. In particular, the message that flows across such connection should be encrypted so that it cannot be read or tampered with by anyone else except the receiver. In addition, the message should be signed by the sender, so that the receiver can verify that the message is authentic, i.e., that it comes from the supposed source. Both of these features can be achieved through public-key cryptography, with the advantage that public-key cryptography, as opposed to symmetric-key cryptography, does not require that both parties have met or exchanged some encryption/decryption key before.

In symmetric-key cryptography, the same key is used for both message encryption and decryption. Both parties have access to this key and they trust each other in keeping that key—i.e., their shared secret—as safe as possible. During message exchange, the sender uses the key to encrypt the message, and the receiver uses the same key to decrypt it. However, for this to work, both sender and receiver must have previously exchanged, by some means, the symmetric key that they will use for encryption/decryption. This creates a problem that must be solved through some other security mechanism, and it is also an obstacle to initiating an interaction between two business partners who have not met before.

In public-key cryptography, every entity that is to engage in some exchange has a pair of keys: one is the public key and the other is the private key. The private key, as the name indicates, is to be kept private and not to be shared with anyone. The public key, on the other hand, can be shared with everyone else, and it can even be published on some central repository. Each pair of public and private keys works in

such a way that whatever is encrypted with the public key can only be decrypted with the corresponding private key, and vice versa (i.e., whatever is encrypted with the private key can only be decrypted with the corresponding public key). This is why the public and private keys are called a "pair," i.e., because they work in combination and they do not work with any other private/public key.

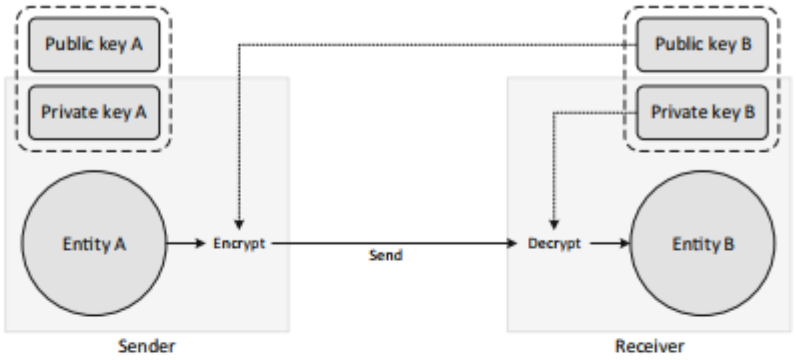


Fig. 12.2 Encryption with the receiver's public key

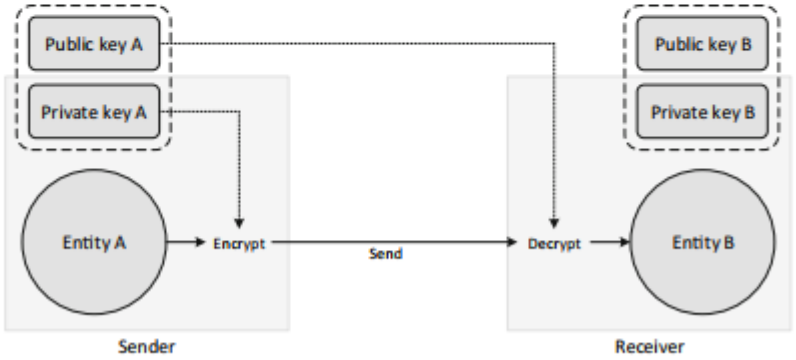


Fig. 12.3 Authentication with the sender's public key

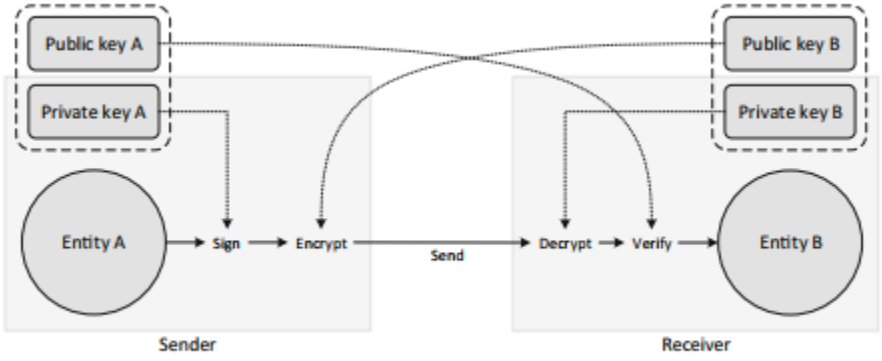


Fig. 12.4 Signing and encryption

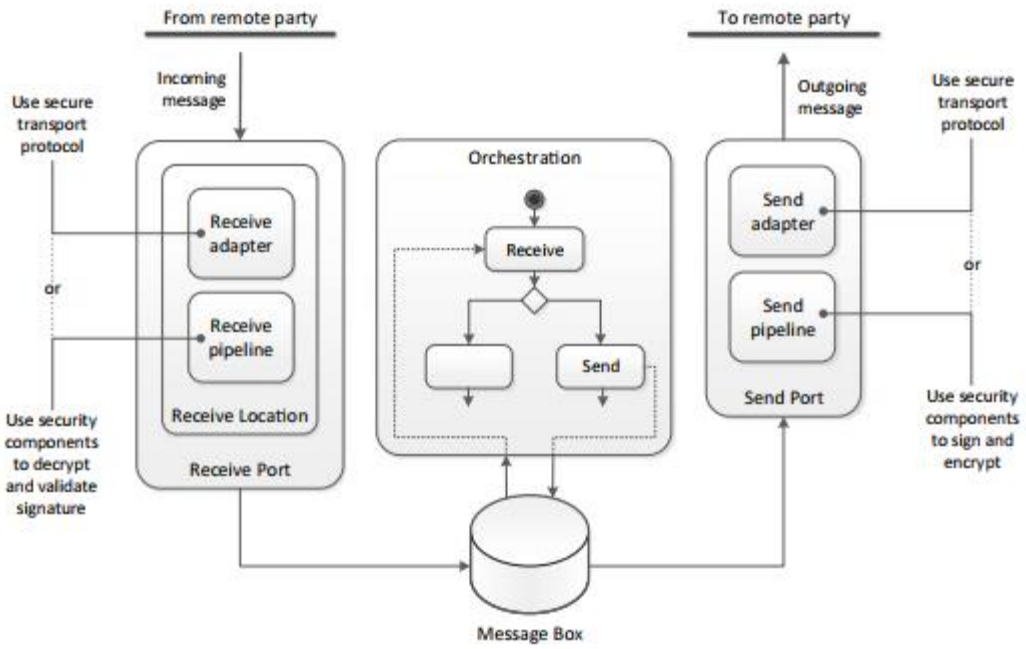


Fig. 12.6 Security in BizTalk solutions

Security in Integration

Public-key cryptography is the basis for many secure communication protocols available today. For example, SSL and it successor TLS are commonly used protocols for the exchange of data between a Web server and a Web browser (i.e., a client), and both of these protocols use public-key cryptography (with server authentication and possibly client authentication as well) during the initial setup of a session between client and server. Besides the Web, these security protocols can also be used in other scenarios, for example to secure the communication between an e-mail client and an e-mail server, or between e-mail servers.

In practice, there are implementations of SSL, TLS, and other security protocols available for general use, so many integration platforms already include those protocols. For the systems integrator, incorporating security mechanisms in an integration solution is often as simple as selecting the appropriate transport protocols, or including some predefined components for message encryption and authentication. Figure 12.6 illustrates how security mechanisms can be incorporated in an integration solution, using BizTalk as an example. In Sect. 2.3 we had already seen that each port includes an adapter and a pipeline, and optionally a transformation map as well. In Fig. 12.6, only the adapter and the pipeline have been represented in each port, since this is where security mechanisms can come into play.

Basically, communication with a remote party can be made secure in two different ways: either at the level of the adapter or at the level of the pipeline. At the level of the adapter, a port can be configured to use a secure transport protocol. For example, instead of using HTTP for communication, the adapter can use HTTPS (i.e., HTTP on top of SSL/TLS). This approach applies equally well to receive ports and to send ports, since SSL/TSL provide encryption and the possibility of authenticating both ends through the use of digital certificates. This is also the approach that requires less changes to the integration solution, since it is essentially a matter of port configuration. However, it should be noted that an adapter can provide security only while the message is in transit; to secure the actual content of the message, it is necessary to perform some processing at the level of the pipeline.

Electronic Data Interchange

Assuming that security concerns have been addressed by the mechanisms and technologies described above, the next issue to be considered in an inter-organizational scenario has to do with the actual content or structure of the messages to be exchanged. In particular, the problem of who defines the message format for an inter-organizational exchange becomes an especially sensitive issue, since each business partner will have its own requirements, and in the worst case there may be no consensus as to what format should be used. This can become a major obstacle to setting up the interaction between processes running at different organizations.

Within an organization, when there is a message to be exchanged between a sender and a receiver, it is possible to decide whether it is the receiver who will have to adapt to the message schema produced by the sender, or whether it is the sender who will have to adapt to the message schema consumed by the receiver. Also, there could be a third possibility, which is to have a transformation map between the two schemas. However, none of these options are very practical in an inter-organizational environment, because this would require the development of a new adapter or a new transformation map for every new business partner.

Inter-organizational relationships are supposed to be dynamic. For example, if a company has a supplier for a certain commodity, but suddenly finds that it can buy the same commodity at a lower price from a different supplier, then the company will want to switch to the new supplier immediately, without having to go through the effort of setting up new message schemas, new adapters, or new transformation maps just to be able to interact with the new supplier. Ideally, the trade of a given product or service should be standardized in such a way that it is possible to establish relationships with new business partners without having to introduce significant changes either in the processes or in the supporting infrastructure.

In general, the trade of any product or service involves the exchange of certain business documents such as, for example, a request for quote (when a customer asks for the price of a certain product), a purchase order (when the customer orders the product from the supplier), a delivery note (where the supplier says when the product will arrive), and an invoice (when the supplier requests payment for the product). These and many other types of business documents are well known to any business organization. The problem is that, in practice, each organization has its own internal representation for these documents, and this makes it difficult to automate the exchange of such documents in an inter-organizational scenario.

In an effort to devise a solution to this problem, several standards have been proposed for the representation of business documents. These standards were meant not only to provide a uniform format, but also to facilitate the automated exchange of those documents between systems running at different organizations. Because of this focus on automated processing, such standards are collectively known as EDI (Electronic Data Interchange), and they include the EDIFACT standard by them United Nations, which is used in Europe and Asia, the X12 standard which is used in North America, as well as some other standards that are used in specific industry sectors. Here we will be focusing mainly on the UN/EDIFACT standard.