



Security Requirements Modeling Tool

SecBPMN2 Elements Reference Guide

(rev 1.0)

For STS-Tool Version 2.1

Table of contents

BPMN 2.0	5
Connections	5
Association	5
Conversation Link	5
Data Association.....	5
Message Flow	5
Sequence Flow.....	5
Events	6
Boundary Event	6
End Event	6
Intermediate Catch Event	6
Intermediate Throw Event	6
Start Event	6
Event Definitions.....	6
Cancel	6
Compensate	7
Conditional.....	7
Error	7
Escalation	7
Link.....	7
Message.....	7
Signal.....	7
Terminate.....	7
Timer.....	7
Gateways.....	8
Complex Gateway	8
Exclusive Gateway	8

Event Based Gateway	8
Inclusive Gateway	8
Parallel Gateway	9
Activities	9
Business Rule Task	9
Choreography Task	9
Manual Task	9
Receive Task	9
Script Task	10
Send Task	10
Service Task	10
Task	10
User Task	10
Data Elements	10
Data Input/Output	11
Input and Output Sets	11
Data Object	11
Data Store	11
Message	11
Error	11
Escalation	12
Signal	12
Resource	12
Variable (Property)	12
Data Type (ItemDefinition)	12
Containers	13
Ad Hoc Sub Process	13
Call Activity	13
Call Choreography	13

Lane.....	13
Pool	13
Sub Choreography	14
Sub Process	14
Transaction	14
Artifacts.....	15
Conversation	15
Group	15
Text Annotation	15
SecBPMN2 Extension.....	15
Security association.....	15
Security Annotations	15
Accountability	16
Auditability	16
Authenticity	16
Availability.....	17
Confidentiality.....	17
Integrity.....	18
Non-repudiation	18
Privacy	18
Bind of duty	19
Separation of duty	19
Non-delegation	19
Security policy associations	20
Walk	20
Negative walk	20
Negative flow	20
The @ wild-card	20
Security policy elements	20

Association of security annotation to call activity.....	21
Call Activity	21
bibliography.....	22

This document describes all SecBPMN2 elements and their visual representation in SecBPMN2 plugin. This document can be used as reference when modelling secure business processes with SecBPMN2 in STS-Tool.

SecBPMN2 is an extension of BPMN 2.0 [1] modelling language. Therefore, this document introduces the BPMN 2.0 elements used in SecBPMN2 than it describes the extension provided in SecBPMN2. For more details on the theoretical background of SecBPMN2 please refer to [2], [3] and [4].


Part of this manual is derived from BPMN 2.0 Modeler¹ user guide, published November 15, 2013.

BPMN 2.0

SecBPMN2 extends BPMN 2.0 collaboration models. Therefore, this section describes the BPMN2.0 elements used in collaboration models, and shows how they are represented in the SecBPMN2 plugin. For a complete specification of BPMN 2.0 please refer to [1].

CONNECTIONS


ASSOCIATION

 An **Association** is used to link information and **Artifacts** with BPMN graphical elements. **Text Annotations** and other **Artifacts** can be associated with the graphical elements. An arrowhead on the **Association** indicates a direction of flow, when appropriate.

CONVERSATION LINK


 **Conversation Links** are used to connect **Conversations** to and from **Participants (Pools)**

DATA ASSOCIATION


 **Data Associations** show a flow of data out of, or in to an **Activity**. An arrow head is used to indicate the direction of the data flow.

Data Associations are used to move data between **Data Objects**, Process Variables, and inputs and outputs of **Activities**, **Processes**, and **Global Tasks**. Process execution does not flow along a **Data Association**, and as a result they have no direct effect on the flow of the **Process**. The purpose of retrieving data from **Data Objects** or Process **Data Inputs** is to fill the **Activities** inputs and later push the output values from the execution of the **Activity** back into **Data Objects** or Process **Data Outputs**.

MESSAGE FLOW

 A **Message Flow** is used to show the flow of **Messages** between two **Participants**. **Pools** in a Collaboration Diagram are used to represent the two **Participants**.

SEQUENCE FLOW

 A **Sequence Flow** is used to show the order in which **Activities** will be performed in a Process or Choreography. A **Sequence Flow** can optionally define a condition **Expression**, indicating that control will be passed down the **Sequence Flow** only if the **Expression** evaluates to true. This **Expression** is typically used when the source of the **Sequence Flow** is a **Gateway** or an **Activity**. A

¹ <https://www.eclipse.org/bpmn2-modeler/>

Sequence Flow that has an Exclusive, Inclusive, or **Complex Gateway** or an **Activity** as its source can also be defined as "default". Such a **Sequence Flow** will have a marker to show that it is a default flow. The default **Sequence Flow** is taken only if all the other outgoing **Sequence Flows** from the **Activity** or **Gateway** are not valid (i.e., their condition **Expressions** are false).

EVENTS

BOUNDARY EVENT



Boundary Events are attached to the borders of an **Activity** and are used to handle conditions (**Event Definitions**) that may have resulted during execution of the **Activity**.

END EVENT



As the name implies, the **End Event** indicates where a Process will end. In terms of **Sequence Flows**, the **End Event** ends the flow of the Process, and thus, will not have any outgoing **Sequence Flows** and no **Sequence Flow** can connect from an **End Event**. An **End Event** may have one or more triggers (**Event Definitions**), which are passed back to an invoking or containing Process (if any).

INTERMEDIATE CATCH EVENT



The **Intermediate Catch Event** is used to handle some kind of condition (**Event Definition**) that has occurred within the process or in an external process.

INTERMEDIATE THROW EVENT



The **Intermediate Throw Event** is used to report some kind of condition (**Event Definition**) to an invoking or containing Process. The receiving Process should be designed so that it is prepared to handle the event, either with a **Start Event**, **Intermediate Catch Event** or a **Boundary Event**.

START EVENT



As the name implies, the **Start Event** indicates where a particular Process will start. In terms of **Sequence Flows**, the **Start Event** starts the flow of the Process, and thus, will not have any incoming **Sequence Flows** and no **Sequence Flow** can connect to a **Start Event**. A **Start Event** may have one or more event triggers (**Event Definitions**) which cause the Process to be initiated.

EVENT DEFINITIONS

Event Definitions determine the behavior of **Events**. An **Event** may have zero or more **Event Definitions**.

CANCEL

This type of **Event Definition** is only allowed when used within a **Transaction** Sub-Process. It is used to "roll back" the effects of the Transaction.

COMPENSATE

Compensation is similar to a [Cancel Event](#) in that it is used to reverse the effects of one or more [Activities](#). In this case, the [Activities](#) may not be contained in a [Transaction](#) Sub-Process, so each [Activity](#) needs to be compensated separately.

CONDITIONAL

This type of [Event](#) is triggered when a condition becomes true. The [Event Definition](#) contains the condition [Expression](#).

ERROR

An [Error Event Definition](#) is used to throw or catch an [Error](#). The [Error](#) payload is associated with a variable owned by the [Event](#). See [Error](#) below, for more details.

ESCALATION

An [Escalation Event Definition](#) is used to throw or catch an [Escalation](#). The [Escalation](#) payload is associated with a variable owned by the [Event](#). See [Escalation](#) below, for more details.

LINK

A [Link](#) is a mechanism for connecting two sections of a [Process](#). [Link Events](#) can be used to create looping situations or to avoid long [Sequence Flow](#) lines. [Link Event](#) uses are limited to a single [Process](#) level (i.e., they cannot link a parent [Process](#) with a [Sub-Process](#)). Paired [Intermediate Events](#) can also be used as “Off-Page Connectors” for printing a Process across multiple pages. They can also be used as generic “Go To” objects within the [Process](#) level.

MESSAGE

A [Message Event Definition](#) can be used to either send or receive a [Message](#). The [Message](#) payload is associated with a variable owned by the [Event](#). See [Message](#) below, for more details.

SIGNAL

A [Signal Event Definition](#) is used to throw or catch a [Signal](#). See [Signal](#) below, for more details.

TERMINATE

This type of [End Event](#) indicates that all [Activities](#) in the [Process](#) should be immediately ended. This includes all instances of multi-instances. The [Process](#) is ended without compensation or event handling.

TIMER

The Timer [Event Definition](#) acts as a delay mechanism based on a specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the [Event](#).

GATEWAYS

A **Gateway** must have multiple incoming or multiple outgoing **Sequence Flows** (i.e., it must either merge or split the process flow). The Gateway Direction property determines its behavior; this property can be one of the following

- **Unspecified** - the **Gateway** may have both multiple incoming and outgoing **Sequence Flows**
- **Mixed** - the **Gateway** must have both multiple incoming and outgoing **Sequence Flows**
- **Converging** - the **Gateway** must have multiple incoming **Sequence Flows**, but may have only one outgoing **Sequence Flow**
- **Diverging** - the **Gateway** may have only one incoming **Sequence Flow**, but must have multiple outgoing **Sequence Flows**

COMPLEX GATEWAY



The **Complex Gateway** can be used to model complex synchronization behavior. An **Expression** is used to describe the precise behavior. For example, this **Expression** could specify that three out of five incoming **Sequence Flows** are needed to activate the **Gateway**. The outgoing paths that are taken by the **Gateway**, is determined by conditions on the outgoing **Sequence Flows** as in the split behavior of the **Inclusive Gateway**.

EXCLUSIVE GATEWAY



A diverging **Exclusive Gateway** (decision) is used to create alternative paths within a Process flow. This is basically the "diversion point in the road" for a Process. For a given instance of the Process, only one of the paths can be taken. A decision can be thought of as a question that is asked at a particular point in the Process. The question has a defined set of alternative answers. Each answer is associated with a condition **Expression** that is associated with a Gateway's outgoing **Sequence Flows**.

A default path can optionally be identified, to be taken in the event that none of the conditional **Expressions** evaluate to true. If a default path is not specified and the **Process** is executed such that none of the conditional **Expressions** evaluates to true, a runtime exception occurs.

A converging **Exclusive Gateway** is used to merge alternative paths.

EVENT BASED GATEWAY



The **Event-Based Gateway** represents a branching point in the Process where the alternative paths that follow the **Gateway** are based on events that occur, rather than the evaluation of expressions using Process data (as with an **Exclusive** or **Inclusive Gateway**). A specific event, usually the receipt of a **Message**, determines the path that will be taken. Basically, the decision is made by another **Participant** based on data that is not visible to a Process, thus requiring the use of the **Event-Based Gateway**.

INCLUSIVE GATEWAY



A diverging **Inclusive Gateway** (inclusive decision) can be used to create alternative but also parallel paths within a Process flow. Unlike the **Exclusive Gateway**, all condition expressions are evaluated. The true evaluation of one condition expression does not exclude the evaluation of other condition expressions. All **Sequence Flows** with a true evaluation will be traversed. Since each path

is considered to be independent, all combinations of the paths **may** be taken, from zero to all. However, it should be designed so that at least one path is taken.

A default path can optionally be identified, to be taken in the event that none of the conditional **Expressions** evaluate to true. If a default path is not specified and the **Process** is executed such that none of the conditional **Expressions** evaluates to true, a runtime exception occurs.

A converging **Inclusive Gateway** is used to merge a combination of alternative and parallel paths.

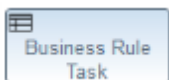
PARALLEL GATEWAY



A **Parallel Gateway** is used to synchronize (combine) parallel flows and to create parallel flows. A **Parallel Gateway** creates parallel paths without checking any conditions; each outgoing **Sequence Flow** is passed control upon execution of this **Gateway**. For incoming flows, the **Parallel Gateway** will wait for all incoming flows before triggering the flow through its outgoing **Sequence Flows**.

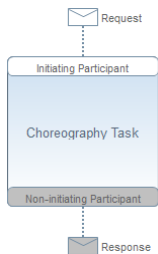
ACTIVITIES

BUSINESS RULE TASK



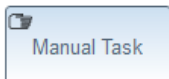
A **Business Rule Task** provides a mechanism for the Process to provide input to a Business Rules Engine and to get the output of calculations that the Business Rules Engine might provide.

CHOREOGRAPHY TASK



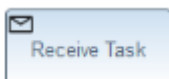
A **Choreography Task** is an atomic **Activity** in a Choreography Process. It represents an Interaction, which may be one or two **Message** exchanges between two **Participants**.

MANUAL TASK



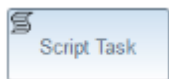
A **Manual Task** is a task that is not managed by any business process engine. It can be considered as an unmanaged task, in the sense that the business process engine does not track the start and completion of such a task. An example of this could be a paper based instruction for a telephone technician to install a telephone at a customer location.

RECEIVE TASK



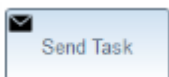
A **Receive Task** is a simple task that is designed to wait for a **Message** to arrive from an external **Participant** (relative to the Process). Once the **Message** has been received, the task is completed.

SCRIPT TASK



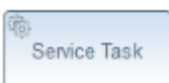
A **Script Task** is executed by a business process engine. The modeler or implementer defines a script in a language that the engine can interpret. When the task is ready to start, the engine will execute the script. When the script is completed, the task will also be completed.

SEND TASK



A **Send Task** is a simple task that is designed to send a **Message** to an external **Participant** (relative to the Process). Once the **Message** has been sent, the task is completed.

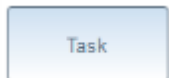
SERVICE TASK



A **Service Task** is a task that uses some sort of service, which could be a Web service or an automated application.

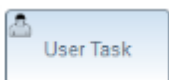
A Service Task defines an **Implementation** which is the underlying technology that will be used to implement the service. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol.

TASK



A **Task** is an atomic **Activity** that is included within a Process. A **Task** is used when the work in the Process is not broken down to a finer level of Process detail.

USER TASK



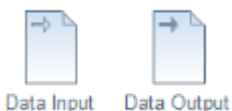
A **User Task** is a typical workflow task where a human actor performs the task with the assistance of a software application. The lifecycle of the task is managed by a software component (called the "Task Manager") and is typically executed in the context of a Process. The **User Task** can be implemented using different technologies, specified by the **Implementation** attribute. Besides the Web service technology, any technology can be used. A **User Task** for instance can be implemented using WSHumanTask by setting **Implementation** to "http://docs.oasis-open.org/ns/bpel4people/ws-humantask/protocol/200803".

DATA ELEMENTS

This section describes all of the possible data elements that can be manipulated by Activities in a Process, both visual and non-visual.

The BPMN 2.0 spec also provides for a "Data State", which is a state of the data contained in a data element. The definition of these states, e.g., possible values, and any specific semantics are out of scope of the BPMN 2.0 spec. Therefore, BPMN adopters can use this element and the BPMN extensibility capabilities to define their own states.

DATA INPUT/OUTPUT



Activities and Processes often required data in order to execute. In addition they may produce data during or as a result of execution. Data requirements are captured as **Data Inputs** and **Data Outputs**.

INPUT AND OUTPUT SETS

An **Activity** or **Process** may be designed such that it can execute with differing sets of input data. This allows the process architect greater flexibility in designing the process, since not all data may be available or even required when executing an **Activity**. For example, some information (e.g. a city name) can be derived or looked up (by, e.g. city postal code) if not provided to the **Activity** at runtime. This is similar to the concept of method overloading in Java or C++.

BPMN 2.0 introduced the concept of **Input Sets** and **Output Sets** to support this concept. Essentially, **Data Inputs**, when grouped together (the “**Input Set**”), define a valid set of data inputs for an **Activity**. **Data Outputs** grouped together (the “**Output Set**”) define the set of data items produced when the **Activity** completes. These **Input** and **Output Sets** also allow you to specify which data items are optionally available on input or optionally created on output, and which items are required or mandatory.

Input and **Output Sets** can also define mutual dependencies; an **Input Set** can indicate which **Output Sets** are produced, and an **Output Set** can indicate the **Input Sets** needed to produce the output data.

DATA OBJECT



Data Objects are the primary construct for modeling data within the Process flow. A **Data Object** has a well-defined lifecycle and structure. A **Data Object** can appear multiple times in a Process diagram, each of which references the same **Data Object** instance. These references are used to simplify diagram connections.

DATA STORE



A **Data Store** provides a mechanism for **Activities** to retrieve or update stored information that will persist beyond the scope of the Process. The same **Data Store** can be visualized, through a **Data Store Reference**, in one or more places in the Process.

MESSAGE



A **Message** represents the content of a communication between two **Participants**.

ERROR



An **Error** represents the content of an **Error Event** or the Fault of a failed **Operation**.

ESCALATION



An **Escalation** identifies a business situation that a Process might need to react to.

SIGNAL



Signals are triggers generated in the **Pool** they are published. They are typically used for broadcast communication within and across Processes, across **Pools**, and between Process diagrams.

RESOURCE

A **Resource** is an actor or responsible party that participates in an activity. This could be, for example, a specific individual, a group, an organization role or position, or an organization. Resources are assigned to Activities during Process execution time.

A **Resource** may need additional information to complete a task. This could be, for example, a customer's contact information, an order request, invoice, etc. This information is represented as **Resource Parameters**. In an executable **Process**, **Resource Parameters** must be in the form of data items that are accessible within the scope of the **Activity** that owns the **Resource**.

VARIABLE (PROPERTY)

Properties, like **Data Objects**, are data containers. But, unlike **Data Objects**, they are not visually displayed on a diagram. Only **Processes**, **Activities**, and **Events** may contain **Properties**. **Properties** are analogous to “Variables” in the context of a programming language, in that they are used to store, transform and convey data as it is moved through the process. **Properties** have the same properties as **Data Objects**, in that they have a well-defined structure, kind and cardinality.

SecBPMN2 plugin uses the term “Variable” instead of “Property” because it seems more intuitive to the software developer, though they refer to the same SecBPMN2 model element. Also, because the term “Property” is so ubiquitous it can sometimes cause confusion when discussing the “properties of a **Property**”.

DATA TYPE (ITEMDEFINITION)

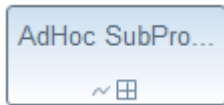
BPMN elements, such as **Data Objects** and **Messages**, represent items that are manipulated during process flows. The set of characteristics that describe these items are known in BPMN 2.0 as **Item Definitions**. However, since SecBPMN2 plugin is designed primarily with the software engineer in mind, it uses the more intuitive name “Data Type”.

Item Definitions can be either “Physical”, such as the mechanical part of a vehicle, or “Informational” such as the catalog of the mechanical parts of a vehicle. This is known as the “Item Kind”. If the item is Informational, then its data structure must also be provided in the **Item Definition**.

It is also possible to define collections of items by setting the “is collection” flag. No assumption is made about the ordering of the items in the collection.

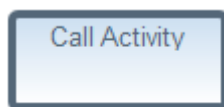
CONTAINERS

AD HOC SUB PROCESS



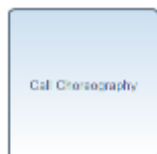
An **Ad-Hoc Sub-Process** contains any number of embedded inner **Activities** and is intended to be executed with a more flexible ordering compared to the typical routing of Processes. The contained **Activities** are executed sequentially or in parallel, they can be executed multiple times in an order that is only constrained through the specified **Sequence Flows**, **Gateways**, and data connections.

CALL ACTIVITY



A **Call Activity** identifies a point in the Process where a global Process or a **Global Task** is used. The **Call Activity** acts as a wrapper for the invocation of a global Process or **Global Task** within the execution. The activation of a **Call Activity** results in the transfer of control to the called global Process or **Global Task**.

CALL CHOREOGRAPHY



A **Call Choreography** identifies a point in the Process where a global Choreography or a **Global Choreography Task** is used. The **Call Choreography** acts as a place holder for the inclusion of the Choreography element it is calling.

LANE



A **Lane** is a sub-partition within a Process (often within a **Pool**) used to organize and categorize **Activities** within a **Pool**. **Lanes** are often used for such things as internal roles (e.g., Manager, Associate), systems (e.g., an enterprise application), an internal department (e.g., shipping, finance), etc. In addition, **Lanes** can be nested or defined in a matrix. For example, there could be an outer set of **Lanes** for company departments and then an inner set of **Lanes** for roles within each department.

POOL



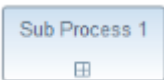
A **Pool** is the graphical representation of a **Participant** in a Collaboration or Choreography and can be a specific **Partner Entity** (e.g., a company) or can be a more general **Partner Role** (e.g., a buyer, seller, or manufacturer). A **Pool** may reference a Process, but is not required to, i.e., it can be a “black box”.

SUB CHOREOGRAPHY



A **Sub-Choreography** is a compound **Activity** in that it has detail that is defined as a flow of other **Activities**, in this case, a Choreography. Each **Sub-Choreography** involves two or more **Participants**. The name of the **Sub-Choreography** and each of the **Participants** are all displayed in the different bands that make up the graphical notation. There are two or more **Participant Bands** and one **Sub-Process** name band.

SUB PROCESS



A **Sub-Process** is an **Activity** whose internal details have been modeled using **Activities**, **Gateways**, **Events**, and **Sequence Flows**. **Sub-Processes** define a contextual scope that can be used for attribute visibility, transactional scope, for the handling of exceptions, of events, or for compensation. A **Sub-Process** can be in a collapsed view that hides its details or in an expanded view that shows its details within the view of the Process in which it is contained.

If a **Sub-Process** is declared as an event handler (the “Is Triggered by Event” flag is set), it is not part of the normal process flow and must be configured as follows:

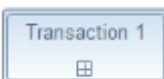
- It may not have any incoming or outgoing **Sequence Flows**.
- It must have one and only one **Start Event** trigger that has one or more of the following **Event Definitions**:
 - Message
 - Error
 - Escalation
 - Compensation
 - Conditional
 - Signal

There are two possible consequences to the parent Process when an Event Sub-Process is triggered:

1. the parent Process can be interrupted
2. the parent Process can continue its work (not interrupted)

This is determined by whether the Start Event that is used has the “Is Interrupting” flag set.

TRANSACTION



A **Transaction** is a specialized **Sub-Process** that is executed atomically, that is, all of its contained activities either execute successfully to completion, or their combined effects (primarily on data) are rolled back.

Transactions must specify a **Transaction Protocol** and the **Method** used to commit or cancel the transaction. The **Protocol** should be set to a technology specific URI, e.g., <http://schemas.xmlsoap.org/ws/2004/10/wsat> for WSAtomicTransaction, or <http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome> for WS-BusinessActivity.

ARTIFACTS

CONVERSATION



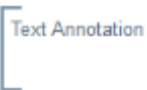
A **Conversation** is an atomic element for a Conversation (Collaboration) diagram. It represents a set of **Message Flows** grouped together based on a concept and/or a **Correlation Key**. A Conversation will involve two or more **Participants**.

GROUP



The **Group** object is an **Artifact** that provides a visual mechanism to group elements of a diagram informally. A **Group** is not an **Activity** or any **Flow Object** and therefore, cannot connect to **Sequence Flows** or **Message Flows**. In addition, **Groups** are not constrained by restrictions of **Pools** and **Lanes**. This means that a **Group** can stretch across the boundaries of a **Pool** to surround diagram elements, often to identify **Activities** that exist within a distributed business-to-business transaction. **Groups** are often used to highlight certain sections of a diagram and do not affect the flow of the Process.

TEXT ANNOTATION



Text Annotations provide additional information to the reader about a BPMN diagram.

SecBPMN2 EXTENSION

This section describes how SecBPMN2 extends BPMN 2.0. The extension consists in a security association and eleven security annotations, which can be used to specify security concepts, and in three sequence flow relations, that can be used to specify security policies, i.e., patterns that can be verified against SecBPMN business processes. For further details please refer to [2], [3] and [4].

SECURITY ASSOCIATION

A security association links a security annotation to STS-ml elements. The association is possible only with a subset of STS-ml elements that depends on the type of security annotation linked.

SECURITY ANNOTATIONS

Security annotations represent security aspects of a business process. They can be included in a diagram by dragging and dropping them from the palette to the diagram, as all other SecBPMN2 elements.

Is possible to specify a set of security properties for each security annotation in a SecBPMN2 diagram. The type of security property that can be specified depends on which SecBPMN2 element the security annotation is linked.

ACCOUNTABILITY

We defined the security aspect represented by this security annotation as the ability of a system to hold users responsible for their actions (e.g. misuse of information). It can be associated to:

- Task
- Pools
- Swimlanes



When an accountability security annotation is associated, with a security association relation, to a task, it is possible to set the list of users to monitor.

AUDITABILITY

We defined the security aspect represented by this security annotation as the ability of a system to conduct persistent, non-by passable monitoring of all actions performed by humans or machines within the system. It can be associated to:

- Task
- Data store
- Pool
- Swimlane
- Message flow



It is possible to set two security properties to auditability security annotations: “Frequency value” and “Frequency period”. They indicate how many times (the former security property) in a span of time (the latter security property) the system is analyzed.

The security property *Frequency* can be specified for tasks, data objects and message flows. If it is set to zero, continuous verification is required.

AUTHENTICITY

We defined the security aspect represented by this security annotation as the ability of a system to verify identity and establish trust in a third party and in information it provides. It can be associated to:

- Task
- Data object
- Data stores
- Pool
- Swimlane



When an authenticity security annotation is associated to a task, a data store or a pool, it is possible to set three security properties: authentication, identification and trust value.

The security property *Identification* can be specified only for tasks. It is a Boolean variable that, if true, specifies that anonymous users should not take part in the execution of the activity.

The security property *Authentication* can be specified only for tasks. It is a Boolean variable that, if true, specifies that the identity of users should be verified.

The security property `trustValue`, can be specified only for tasks. It specifies the minimum level of trust the executor of activity must have

AVAILABILITY

We defined the security aspect represented by this security annotation as the ability of a system to ensure that all system's components are available and operational when they are required by authorized users. It can be associated to:

- Task
- Data object
- Data store
- Message flows



It is possible to set “Availability Level” security property for all availability security annotations. When an availability security annotation is associated to a data object, it is possible to set the “Authorized End Users” security property.

The security property `authUsers`, can be specified for tasks, data objects and message flows. It specifies the user that can access the resource. With the keyword `ALL` it specifies that all users are authorized to request the data object.

The security property `Level` can be specified for tasks, data objects and message flows. It specifies the minimum time percentage that the resource (i.e., activity, data object or message flow, depending on the variant of availability annotation) should be available.

CONFIDENTIALITY

We defined the security aspect represented by this security annotation as the ability of a system to ensure that only authorized users access information. It can be associated to:

- Data object
- Data store
- Message flow



It is possible to set “Writers” and “Readers” security properties for all confidentiality security annotations.

The security property `Readers` can be specified for data objects and message flows. It specifies the set of users that are authorized to read the data object (or receive from the message flow).

The security property `Writers` can be specified for data objects and message flows. It specifies the set of users that are authorized to write the data object do (or send through the message flow).

INTEGRITY

We defined the security aspect represented by this security annotation as the ability of a system to ensure completeness, accuracy and absence of unauthorized modifications in all its components. It can be associated to:

- Task
- Data object
- Message



When an integrity security annotation is associated to a task, it is possible to set “Hardware”, “Software” and “Personnel” security properties.

The security property `Personnel` can be specified only for tasks. It is a Boolean variable that, if true, specifies that the personnel, who executes the task, shall be protected from intentional corruption.

The security property `Hardware` can be specified only for tasks. It is a Boolean variable that, if true, specifies that the hardware, used to execute the task, shall be protected from intentional corruption.

The security property `Software` can be specified only for tasks. It is a Boolean variable that, if true, specifies that the software, used to execute the task, shall be protected from intentional corruption.

NON-REPUDIATION

We defined the security aspect represented by this security annotation as the ability of a system to prove (with legal validity) occurrence/non-occurrence of an event or participation/non-participation of a party in an event. It can be associated to:

- Task
- Message flow
- start/intermediate/end event of type message/error/cancel/compensation/signal



It is possible to set “Execution” security property for all confidentiality security annotations.

The security property `Execution` can be specified for tasks and message flows. It is a Boolean variable that specifies: (i) a proof of execution of the task or a proof of usage of the message flow shall be provided, if set to true; (ii) a proof of non-execution for the task or a proof of non-usage for the message flow shall be provided, if set to false.

PRIVACY

We defined the security aspect represented by this security annotation as the ability of a system to obey privacy legislation and to enable individuals to control, where feasible, their personal information (user-involvement). It can be associated to:

- Task
- Data object



It is possible to set “Sensitive Information” security property for all confidentiality security annotations.

The security property `SensitiveInfo` can be specified for tasks and data objects. It specifies the set of sensitive information to protect.

BIND OF DUTY

We defined the security aspect represented by this security annotation as the ability of the system to require the same person to be responsible for the completion of a set of related tasks. It can be associated to:

- `Pool`
- `Swimlane`



The security property `Dynamic` can be specified only for participants. It is a Boolean variable that, if true, specifies that the set of people, which play the role specified in the participant, change during the execution of the system.

SEPARATION OF DUTY

We defined the security aspect represented by this security annotation as the ability of the system to force two or more different people to be responsible for the completion of a task or set of related tasks. It can be associated to:

- `Pool`
- `Swimlane`

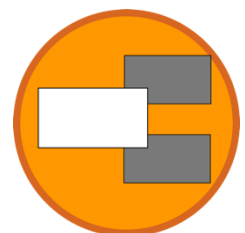


The security property `Dynamic` can be specified only for participants. It is a Boolean variable that, if true, specifies that the set of people, which play the role specified in the participant, change during the execution of the system.

NON-DELEGATION

We defined the security aspect represented by this security annotation as the ability of the system to require that a set of actions is executed only by the users assigned. It can be associated to:

- `Task`



Only `EnfBy` security property can be specified for non-delegation.

SECURITY POLICY ASSOCIATIONS

WALK

Walk relation is represented as a solid black double-headed arrow, shown in Figure 1. It links two tasks and it can be used in with security policies and security policy templates.

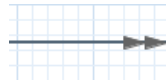


Figure 1 Walk relation

NEGATIVE WALK

The negative walk relation is represented as a solid black double-headed arrow, the heads of the arrow are white. Figure 2 shows an example of the relation. It links two tasks and it can be used in with security policies and security policy templates.



Figure 2 Negative Walk Relation

NEGATIVE FLOW

The negative flow relation is represented as a solid black single-headed arrow, the head of the arrow is white. Figure 3 shows an example of the relation. It links two tasks and it can be used in with security policies and security policy templates.



Figure 3 Negative Flow Relation

THE @ WILD-CARD

In security policy and security policy template editors, is possible to use the @ wild-card as a first character of the name of a element to specify that such element will match any elements of the same type. For example, the security policy in Figure 4, will match any business process in which the task task1 is followed by any task.

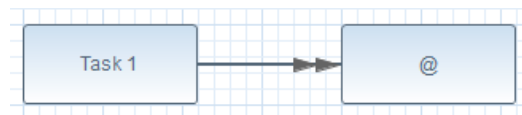


Figure 4 @ Wild-Card example

When a process is not in the boundaries of a participant then it is considered associated to a participant with an @ wildcard, meaning that there will be no restriction in the match of the executor of the process.

SECURITY POLICY ELEMENTS

ASSOCIATION OF SECURITY ANNOTATION TO CALL ACTIVITY

In the security policy template editor is possible to associate security annotations with call activity element. When a security annotation is associated to this element, the annotation is propagated to the elements of the business process. The propagation rule can be set in Propagation rule property of the security annotation specific property, in the property pane, as shown in Figure 5.

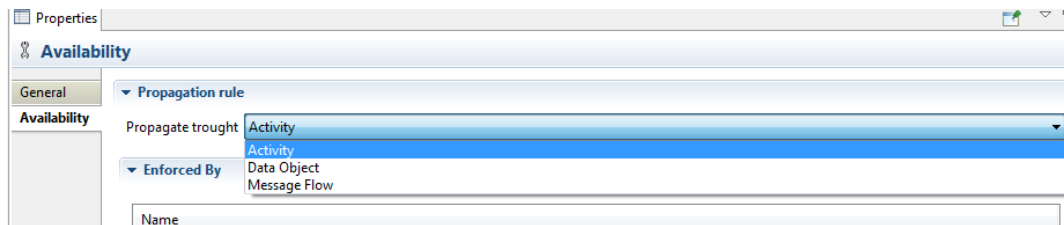


Figure 5 Propagate through example

CALL ACTIVITY

In security policy template is possible to map a goal to a call activity. This is a visualization solution when the security requirement associated to the security policy template, contains a delegation and, therefore, a reference to a goal. Mapping the call activity to a goal will generate a security policy whose call activity will be linked to the process that is mapped to the goal.

BIBLIOGRAPHY

- [1] OMG, «Business Process Model and Notation (BPMN),» 2011.
- [2] M. Salnitri, F. Dalpiaz e P. Giorgini, «Designing Secure Business Processes with SecBPMN,» *International Journal on Software and System Engineering*, 2015.
- [3] M. Salnitri, E. Paja e P. Giorgini, «Mantaining Secure Business Processes in Light of Socio-Technical Systems' Evolution,» *To Appear*, p. 14, 2016.
- [4] M. Salnitri, E. Paja e P. Giorgini, «Preserving Compliance with Security Requirements in Socio-Technical Systems,» *Proceedings of Cyper Seurity and Privacy (CSP) forum*, 2014.