UN/CEFACT’s Modeling Methodology (UMM):

UMM Meta Model – Foundation Module Version 1.0
Technical Specification
2006-10-06
# Table of Contents

1 About this Document ................................................................. 2
  1.1 Status of this Document ...................................................... 2
  1.2 Revision History ............................................................... 2
  1.3 Document Context ............................................................ 2

2 Project Team ........................................................................... 3
  2.1 Disclaimer ........................................................................... 3
  2.2 Contact ............................................................................... 3
  2.3 Project Team Participants .................................................... 3

3 Introduction ............................................................................ 4
  3.1 Audience ............................................................................. 4
  3.2 Related Documents ............................................................. 4
  3.3 UN/CEFACT's Modeling Methodology (UMM): Overview ......... 5
  3.4 Objectives ........................................................................... 6
    3.4.1 Goals of the Technical Specification ................................. 6
    3.4.2 Requirements ............................................................... 6
    3.4.3 Caveats and Assumptions ............................................... 7
  3.5 Structure of the UMM Foundation Module ......................... 7

4 Dependency on other UMM modules (normative) .................. 9

5 UMM Foundation Module ....................................................... 10
  5.0 Foundation Module Management ........................................ 10
    5.0.1 Conceptual Description (informative) .............................. 10
    5.0.2 Stereotypes and Tag Definitions (normative) ................... 11
    5.0.3 Constraints (normative) ................................................. 13
    5.0.4 OCL methods used in the
       UMM Foundation Module Management (normative) .......... 14
  5.1 Business Domain View ..................................................... 15
    5.1.1 Conceptual Description (informative) .............................. 15
    5.1.2 Stereotypes and Tag Definitions (normative) ................... 16
    5.1.3 Constraints (normative) ................................................. 21
    5.1.4 Example (informative) .................................................. 23
    5.1.5 OCL methods used in all packages of the BDV (normative) .... 24
  5.2 Business Requirements View ............................................. 26
    5.2.0 Sub-Views in the Requirements View ............................... 26
      5.2.0.1 Conceptual Description (informative) ......................... 26
      5.2.0.2 Stereotypes and Tag Definitions (normative) .............. 27
      5.2.0.3 Constraints (normative) ......................................... 29
    5.2.1 Business Process View .................................................. 30
      5.2.1.1 Conceptual Description (informative) ......................... 30
      5.2.1.2 Stereotypes and Tag Definitions (normative) .............. 31
      5.2.1.3 Constraints (normative) ......................................... 33
      5.2.1.4 Example (informative) .......................................... 34
    5.2.2 Business Entity View .................................................... 35
      5.2.2.1 Conceptual Description (informative) ......................... 35
      5.2.2.2 Stereotypes and Tag Definitions (normative) .............. 35
      5.2.2.3 Constraints (normative) ......................................... 36
      5.2.2.4 Example (informative) .......................................... 37
    5.2.3 Partnership Requirements View .................................... 39
      5.2.3.1 Conceptual Description (informative) ......................... 39
5.2.3.2 Stereotypes and Tag Definitions (normative)................................. 42
5.2.3.3 Constraints (normative)................................................................. 44
5.2.3.4 Example (informative)................................................................. 50
5.2.4 OCL methods used in all packages of the BRV (normative).............. 51
5.3 Business Transaction View................................................................. 56
  5.3.0 Views in the Transaction View...................................................... 56
    5.3.0.1 Conceptual Description (informative)...................................... 56
    5.3.0.2 Stereotypes and Tag Definitions (normative)......................... 57
    5.3.0.3 Constraints (normative).......................................................... 58
  5.3.1 Business Choreography View....................................................... 59
    5.3.1.1 Conceptual Description (informative)...................................... 59
    5.3.1.2 Stereotypes and Tag Definitions (normative)......................... 60
    5.3.1.3 Constraints (normative).......................................................... 62
    5.3.1.4 Example (informative)............................................................ 64
  5.3.2 Business Interaction View............................................................. 65
    5.3.2.1 Conceptual Description (informative)...................................... 65
    5.3.2.2 Stereotypes and Tag Definitions (normative)......................... 67
    5.3.2.3 Constraints (normative).......................................................... 73
    5.3.2.4 Example (informative)............................................................ 79
  5.3.3 Business Information View............................................................ 80
    5.3.3.1 Conceptual Description (informative)...................................... 80
    5.3.3.2 Stereotypes and Tag Definitions (normative)......................... 81
    5.3.3.3 Constraints (normative).......................................................... 82
    5.3.3.4 Example (informative)............................................................ 83
  5.3.4 OCL methods used in all packages of the BTV (normative).............. 84
Copyright Statement ............................................................................... 90
1 About this Document

1.1 Status of this Document

This document has completed the Open Development Process (ODP) of UN/CEFACT on 2006-10-06. It is a UN/CEFACT Technical Specification.

1.2 Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Release</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate for 1.0</td>
<td>First Working Draft</td>
<td>2005-08-11</td>
<td></td>
</tr>
<tr>
<td>Candidate for 1.0</td>
<td>Second Working Draft</td>
<td>2006-03-17</td>
<td></td>
</tr>
<tr>
<td>Candidate for 1.0</td>
<td>Final Working Draft</td>
<td>2006-06-20</td>
<td></td>
</tr>
<tr>
<td>Version 1.0</td>
<td>Technical Specification</td>
<td>2006-10-06</td>
<td></td>
</tr>
</tbody>
</table>

1.3 Document Context

The UMM meta model is divided in a set of meta modules. This means the UMM meta model is partitioned into functional levels, ranging from core, minimal functionality to complete functionality. The following partitions levels have been defined for meta modules:

- **Base**: Covers the fundamental principles that are shared across all the other modules.
- **Foundation**: Includes the core concepts of the UMM. Defines all the concepts that are used as part of the minimal methodology to produce a UMM compliant business collaboration model.
- **Specialization**: Multiple specialization modules might define add-on concepts to the foundation. Each specialization module addresses a specialized type of analysis that extends the foundation module at a well-defined extension point for a certain topic. Specialization modules might become candidates for later inclusion into the foundation module.
- **Extension**: Extension modules serve the same purpose as specialization modules. Whereas specialization modules are developed and maintained by UN/CEFACT, extension modules are adding features that are created and maintained by external organization.

This specification defines the foundation module of UMM.
2 Project Team

2.1 Disclaimer
The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this technical specification.

2.2 Contact
Name: Christian Huemer
Company: Vienna University of Technology
Street: Favoritenstrasse 9-11/188
City, state, zip/other: 1040
Nation: Austria
Phone: +43 1 58801 18882
Email: huemer@big.tuwien.ac.at

2.3 Project Team Participants
Project Team Lead: Christian Huemer Austria
Editing Team: Jens Dietrich (Lead Editor) Germany
Birgit Hofreiter Austria
Christian Huemer Austria
Philipp Liegl Austria
Rainer Schuster Austria
Marco Zapletal Austria
Contributors: Steve Capell Australia
Sylvie Colas France
William McCarthy USA
Glenn Miller Canada
Harry Moyer Australia
Nita Sharma USA
Gunther Stuhec Germany
Jörn Guy Suess Germany
Anders Tell Sweden

The Editing Team of this UMM foundation module likes to thank former members of TMG’s Business Process Working Group (BPWG) who have spent enormous efforts in putting the UMM into a stage that we were able to build upon in order to create this foundation module:

Jim Clark USA
Kenji Itoh Japan
Paul Levine USA
Klaus-Dieter Naujok Canada
Dave Welsh USA
3 Introduction

3.1 Audience

A reader of the document MUST have a deep understanding of UML 1.4. She or he MUST be able to understand meta models denoted as UML class diagrams. She or he SHOULD be familiar with the UML 1.4 meta model, at least she or he MUST be able to check back with the UML 1.4 meta model. The reader SHOULD be familiar with OCL 2.0 in order to understand the OCL constraints of this UMM profile – those who are not familiar with OCL are provided with a plain text description of the constraint.

The information described in this manual is aimed at

- advanced business process modelers who check a UML model for UMM compliance (if not supported by a tool)
- advanced business process modelers who train other business process modelers and business process analysts
- software designers who want to produce UML tools providing support for this UMM foundation module
- software designers who want to produce tools to transform UMM compliant business collaboration models into specifications of the IT-layer (ebXML, Web Services, UN/EDIFACT, etc.).
- software designers who want to produce repositories to register UMM compliant business collaboration models

3.2 Related Documents

- **UN/CEFACT**
  - UN/CEFACT Open Development Process
    http://www.unece.org/cefact/cf_plenary/plenary05/cf_05_05e.pdf
  - BCSS: A UML Profile for Core Components
    BCSS LINK
  - Core Component Technical Specification
- **International Organization for Standardization (ISO)**
  - Open-edi Reference Model. ISO/IEC 14662
- **Object Management Group (OMG)**
  - Unified Modeling Language Specification (UML), Version 1.4.2
    http://www.omg.org/docs/formal/05-04-01.pdf
3.3 UN/CEFACT’s Modeling Methodology (UMM): Overview

UN/CEFACT’s Modeling Methodology (UMM) is a UML modeling approach to design the business services that each partner must provide in order to collaborate. It provides the business justification for the services to be implemented in a service-oriented collaboration architecture. Thus, a primary vision of UN/CEFACT is to capture the business knowledge that enables the development of low cost software based on service-oriented architectures (SOA) helping the small and medium size companies (SMEs), and emerging economies to engage in e-Business practices. UMM focuses on developing a global choreography of inter-organizational business processes and their information exchanges. UMM models are notated in UML syntax and are platform independent models. The platform independent UMM models identify which services have to be realized in a service-oriented architecture implementing the business collaboration. This approach provides insurance against technical obsolescence.

The UMM, as described in this document is the formal description technique for describing any Open-edi scenario as defined in ISO/IEC 14662 Open-edi reference model. An Open-edi scenario is a formal means to specify a class of business transactions having the same business goal, such as, purchasing or inventory management. The primary scope of UMM is the Business Operations View (BOV) and not the Functional Service View (FSV) as defined in ISO/IEC IS 14662. The BOV is defined as “a perspective of business transactions limited to those aspects regarding the making of business decisions and commitments among organizations”, while the FSV is focused on implementation specific, technological aspects of Open-edi. The commitments of the BOV layer are reflected in the choreography of the inter-organizational business process and its information exchanges. At the FSV layer this choreography must be implemented by a set of composite services. It follows, that UMM on the BOV layer defines what the business is about and technologies on the FSV layer define how to implement the business by a service-oriented architecture.

This version of the UMM consists of three views each covering a set of well defined artifacts:

- Business Domain View (BDV)
- Business Requirements View (BRV)
- Business Transaction View (BTV)

Business Domain View (BDV): The BDV is used to gather existing knowledge. It identifies the business processes in the domain and the business problems that are important to stakeholders. It is important at this stage that business processes are not constructed, but discovered. Stakeholders might describe intra-organizational as well as inter-organizational business processes. All of this takes place in the language of the business experts and stakeholders. The business domain view results in a categorization of the business domain (manifested as a hierarchical structure of packages) and a set of relevant business processes (manifested as use cases). The result may be depicted in use case diagrams.

Business Requirements View (BRV): The goal of the BRV is identifying collaborative business processes between different business partner types and describing the requirements regarding these collaborative business processes. In order to identify collaborative business processes the static descriptions of the internal business processes discovered in the BDV are described in more detail and are analyzed regarding their dynamic behavior and their relationship to each other. Based on this analysis the relevant “real-world”-concepts in the domain of the collaboration are identified. This is done by focusing on business entities, which are “real-word” things having business significance and a shared among the business partners involved in the collaboration. The requirements of aligning the states of these business entities between the business partners are documented by business collaboration use cases and by business transaction use cases.

Business Transaction View (BTV): The BTV represents the view of the business process analyst who transforms the requirements into a choreography of information exchanges. Currently, the overall
choreography of a business collaboration is defined by an activity graph called business collaboration protocol. In a future version other alternatives may be developed. The business collaboration protocol choreographs the flow among business interactions. This flow depends on the states of business entities. Currently, a business interaction is always defined by a business transaction. Other alternatives may be developed in future versions. A business transaction defines a simple choreography of exchanging business information between two authorized roles and an optional response. A business transaction identifies the business actions of each partner responsible for sending and receiving the business information. These actions correspond to the services that must be implemented on each business partner’s side in a service-oriented collaboration architecture. The business information exchanged corresponds to the input/output of these services. The choreography among the business transactions – described by the business collaboration protocol in UMM – is easily mapped to machine-readable choreography languages (such as BPEL, WS-CDL, BPSS).

An execution of a business transaction usually results in the change of state of one or more business entities. Thus, the information exchanged in a transaction should be limited to the minimum information needed to change the state of a business entity. Nevertheless, UMM allows the definition of an information exchange in a document-centric approach – even if this is not recommended. A business transaction leads to synchronized states of the business objects at both partners participating in a business transaction. Inasmuch, a business transaction is a unit of work that allows roll-back. A business transaction has a number of quality of service (QoS) parameters that represent security and timing requirements. These are specified in tagged values.

3.4 Objectives

3.4.1 Goals of the Technical Specification

The goals of this specification are:

- To define the semantics of well-formed UMM business collaboration models.
- To define the validation rules for UMM compliant business collaboration models.
- To clarify the basic concepts that a UMM-compliant business collaboration model is based on.
- To provide an unambiguous definition for UMM business collaboration models that allows a unambiguous mapping to artifacts for deployment in a service-oriented architecture. Note, that the mapping itself is not part of UMM.
- To define a UML profile for the UMM foundation module that allows UML tool vendors to customize their tools to be UMM compliant. Better tool support will lead to a growing UMM user base.

3.4.2 Requirements

This specification is guided by the following key requirements derived from the above goals:

- The UMM foundation module defines only those modeling concepts that are considered as fundamental to deliver a UMM compliant model. This means it delivers concepts to structure the domain (in the business domain view), to gather requirements for collaborative business processes (in the business requirements view) and to provide a choreography of business information exchanges (in the business transaction view). Additional advanced modeling concepts shall be covered in specialization and extension modules.
- The UMM foundation module is directed towards the business operational view of Open-edi. This means it is independent of certain implementation technologies used in SOAs like Web Services and ebXML or whatever comes up in the future. However, the UMM compliant business collaboration models must be defined in a way that allows a mapping to an implementation technology of choice. Such a mapping is not part of the UMM foundation module. It is a candidate for a specialization/extension module.
• Today, the UML is the most commonly supported modeling language by modeling tools. In order to use the broad range of tools, a UMM business collaboration model must be a special kind of UML model. Thus, the UMM foundation module is based on the UML meta model. In fact, it provides a UML Profile consisting of stereotypes, tagged definitions and constraints.

• In order to support a broad adoption of the UMM-modeling approach the formal descriptions of the UMM shall be supplemented by a set of examples that show UMM compliant artifacts.

3.4.3 Caveats and Assumptions

This specification makes the following assumptions:

• This UML profile is based on the UML meta-model version 1.4.2. This version is the current ISO version. Using another UML meta-model as a basis for the development of a UMM compliant business collaboration model will not deliver correct results.

• The basic concepts of the UMM and the way they relate to each other shall be described and explained by means of a meta model (to be found in the non-normative “conceptual description” sections of this document)

• Most modeling tools do not evaluate OCL constraints against model data. Accordingly, validation of UMM semantics as defined by the OCL constraints in this specification will normally only be possible using either an external validation service or a custom plug-in.

• Different specialization and extension modules might extend the foundation module in order to define additional semantics to the minimum semantics required to create a UMM compliant business collaboration model.

3.5 Structure of the UMM Foundation Module

Section 5 defines the UML profile of the foundation module of the UMM meta model. The figure below shows the package structure of the foundation module of the UMM meta model. The number depicted in
the folders of this figure refers to the subsection which defines the stereotypes, tag definitions and constraints of the corresponding package. The first level packages of the foundation module conform to the three views of the current UMM version: Business Domain View (5.1), Business Requirements View (5.2), and Business Transaction View (5.3). Since the Business Domain View (5.1) does not include different types of artefacts, it is not split into sub-packages. The Business Requirements View (5.2) covers three different types of artefacts: activity graphs of business processes, business entity life cycles and collaboration requirements defined in use cases. Accordingly, it consists of the sub-packages Business Process View (5.2.1), Business Entity View (5.2.2), and Partnership Requirements View (5.2.3). Similarly, the Business Transaction View (5.3) is built by three different types of artefacts: choreography of a business collaboration, choreography of business interactions (currently i.e. business transactions) leading to synchronized states, and business information exchanged in the interactions. Consequently, it includes the sub-packages Business Choreography View (5.3.1), Business Interaction View (5.3.2), and Business Information View (5.3.3).

Each section describing a package is structured in the same way. The first subsection is informative. It describes the conceptual model of the artefact that is addressed by the package. The second subsection is normative and defines all the stereotypes and associated tag definitions that are defined in the package. The third subsection is normative and includes all the constraints both in plain text and in OCL that apply to the respective package. The fourth subsection is informative and depicts an example instance of the artefact type addressed by the package.
The UMM foundation module 1.0 is built on top of the UMM base module 1.0. This means that all stereotypes and tag definitions defined in the UMM base module 1.0 are imported into the UMM foundation module 1.0. The figure below shows the stereotypes defined in the UMM base module also used in the foundation module. Note, the stereotypes of the base module are depicted in grey background in all figures of this specification. The formal definition of the stereotypes RegistryObject and BusinessLibraryPackage is given in the UMM base module 1.0 specification. In the foundation module, packages - that are containers of stereotypes realizing main UMM artefacts - are defined as specializations of the base stereotype BusinessLibraryPackage. This means that such packages and their contents are candidates for registration in a registry. In the UMM foundation module 1.0 we do not define any stereotype that directly inherits from RegistryObject. As a consequence, only packages are candidates for registration.
5  UMM Foundation Module

5.0  Foundation Module Management

5.0.1  Conceptual Description (informative)

A project that follows the UMM approach leads to a business collaboration model. A business collaboration model that is UMM compliant is stereotyped as BusinessCollaborationModel. As described above the UMM is built by three views. The business domain view focuses on understanding the business domain under consideration. Although this view is considered as important, the results may be captured in non-UML compliant artefacts and/or may not be included in the model and referenced instead. Since the business domain view is optional, the BusinessCollaborationModel is composed of zero or one BusinessDomainView. The business requirements view and the business transaction view are mandatory parts of a business collaboration model. Thus a BusinessCollaborationModel is composed of exactly one BusinessRequirementsView. Similarly, a BusinessCollaborationModel is composed of exactly one BusinessTransactionView.
5.0.2 Stereotypes and Tag Definitions (normative)

Figure 6 UMM Foundation Module Management - Abstract Syntax

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessCollaborationModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Model</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td>Description</td>
<td>A business collaboration model is a model that is compliant to the UMM meta model. It MUST be compliant to the base and foundation module, and it MAY be compliant to one or more specialisation and/or extension modules.</td>
</tr>
</tbody>
</table>

Tag Definition

<table>
<thead>
<tr>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

Inherited tagged values:
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessDomainView</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Package</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A business domain is a framework for identification and understanding of business processes as well as categorizing them according to a classification schema. The business domain view is a container capturing the categorization scheme and categorized business processes.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td><strong>Inherited tagged values:</strong></td>
</tr>
<tr>
<td></td>
<td>- baseURN</td>
</tr>
<tr>
<td></td>
<td>- owner</td>
</tr>
<tr>
<td></td>
<td>- copyright</td>
</tr>
<tr>
<td></td>
<td>- reference</td>
</tr>
<tr>
<td></td>
<td>- version</td>
</tr>
<tr>
<td></td>
<td>- status</td>
</tr>
<tr>
<td></td>
<td>- businessTerm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessRequirementsView</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Package</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The business requirements view is a container for all elements needed to identify and describe the requirements on a collaboration between business partner types playing certain authorized roles.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td><strong>Inherited tagged values:</strong></td>
</tr>
<tr>
<td></td>
<td>- baseURN</td>
</tr>
<tr>
<td></td>
<td>- owner</td>
</tr>
<tr>
<td></td>
<td>- copyright</td>
</tr>
<tr>
<td></td>
<td>- reference</td>
</tr>
<tr>
<td></td>
<td>- version</td>
</tr>
<tr>
<td></td>
<td>- status</td>
</tr>
<tr>
<td></td>
<td>- businessTerm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessTransactionView</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Package</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The business transaction view is a container for all elements needed to describe the choreography of a business collaboration at various levels and the information exchanged in each step of the choreography.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td><strong>Inherited tagged values:</strong></td>
</tr>
<tr>
<td></td>
<td>- baseURN</td>
</tr>
<tr>
<td></td>
<td>- owner</td>
</tr>
<tr>
<td></td>
<td>- copyright</td>
</tr>
<tr>
<td></td>
<td>- reference</td>
</tr>
<tr>
<td></td>
<td>- version</td>
</tr>
<tr>
<td></td>
<td>- status</td>
</tr>
<tr>
<td></td>
<td>- businessTerm</td>
</tr>
</tbody>
</table>
5.0.3 Constraints (normative)

A BusinessCollaborationModel MUST NOT contain more than one BusinessDomainView package (but it MAY contain no BusinessDomainView package at all)

package Model_Management
context Model

inv zeroToOneBusinessDomainView:
self.isBusinessCollaborationModel() implies
self.ownedElement->select(isBusinessDomainView())->size()<=1

A BusinessCollaborationModel MUST contain exactly one BusinessRequirementsView package.

package Model_Management
context Model

inv oneBusinessRequirementsView:
self.isBusinessCollaborationModel() implies
self.ownedElement->one(isBusinessRequirementsView())

A BusinessCollaborationModel MUST contain exactly one BusinessTransactionView package

package Model_Management
context Model

inv oneBusinessTransactionView:
self.isBusinessCollaborationModel() implies
self.ownedElement->one(isBusinessTransactionView())

A BusinessDomainView, the BusinessRequirementsView, and the BusinessTransactionView MUST be directly located under the root of the BusinessCollaborationModel.

package Model_Management
context Package

inv rootLevelPackages
(self.isBusinessDomainView() or self.isBusinessRequirementsView() or self.isBusinessTransactionView()) implies
self.namespace.isBusinessCollaborationModel())
OCL-Methods

package Foundation::Core
context ModelElement

--Predefined method which evaluates, if the given ModelElement
--has a stereotype equal to the passed name

def:
let hasStereotype (st : String) : Boolean =
  self.stereotype->select(cst | cst.name = st)->notEmpty()

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessCollaborationModel'

def:
let isBusinessCollaborationModel() : Boolean =
  self.oclIsKindOf(Model) and
  self.hasStereotype('BusinessCollaborationModel')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessDomainView'

def:
let isBusinessDomainView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessDomainView')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessRequirementsView'

def:
let isBusinessRequirementsView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessRequirementsView')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessTransactionView'

def:
let isBusinessTransactionView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessTransactionView')
5.1 Business Domain View

5.1.1 Conceptual Description (informative)

The business domain view is used to discover business processes that are of relevance in a project. A business process is executed by at least one (but possibly more) business partner types. A business partner type might execute multiple business processes. Thus, the participates association between BusinessPartnerType and BusinessProcess is a (1..n) to (0..n) association. A business partner type has a vested interest in the business process. This is the characteristic of a stakeholder. Thus, a BusinessPartnerType is a specialization of a Stakeholder. In general, a stakeholder does not need to participate in a business process. A stakeholder might have interest in multiple business processes and a business process might be of interest to multiple stakeholders. The relationship between a BusinessProcess and a Stakeholder is described by the isOfInterestTo dependency in UMM. A business process can be decomposed into sub-processes using the «include» and «extends» association stereotypes. This is denoted by the unary (0..1) to (0..*) composition of BusinessProcess.

To enable users to readily identify business processes, these business processes should be classified into business categories. Thus, the BusinessDomainView is composed of one or many (1..n) BusinessCategories. A business category might be recursively composed of other business categories. This means business categories might build a hierarchy. Hence, a unary (0..1) to (0..n) composition is defined for BusinessCategory. A business process is assigned to exactly one business category. A business category on the lowest level of a business category hierarchy includes one or more processes, whereas a business category on a higher level does not include any business process. Accordingly, the composition between BusinessCategory and BusinessProcess is 1 to (0..n).

UN/CEFACT suggests - but does not mandate - the use of specializations of the stereotype of BusinessCategory. These specializations are BusinessArea and ProcessArea. A business area corresponds to a division of an organization and a process area corresponds to a set of common operations within the business area. Similarly to business category, business area as well as process area may form a hierarchy. Thus, BusinessArea and ProcessArea inherit the unary composition from BusinessCategory. However, it
is important that business areas include only business areas except the lowest level of a business area hierarchy which is composed of one or more process areas. Therefore, we have a (0..1) to (0..n) composition between BusinessArea and ProcessArea. Business areas must not include business processes. The lowest level of a process area hierarchy includes one or more business processes. Whereas process areas in a higher level of the hierarchy do not include any business process. Accordingly, the composition between ProcessArea and BusinessProcess is 1 to (0..n).

The stereotype BusinessCategory and the combination of the stereotypes BusinessArea and ProcessArea are considered as alternatives. A UMM model must not use both alternatives.

5.1.2 Stereotypes and Tag Definitions (normative)

![BusinessDomainView Abstract Syntax](image)

Figure 8 BusinessDomainView Abstract Syntax
Business categories are used to classify the business processes in the Business Domain View. The prime purpose of classifying the business processes is to enable potential users to readily identify processes that have been defined in the business category under consideration.

Consequently a business category is used to group either other business categories or business processes that belong to the respective business category. The Business Domain View is structured either by this stereotype BusinessCategory or by its specializations BusinessArea and Process Area (see below for these stereotype definitions).

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>The purpose to be achieved by the business process within the business category under consideration.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Defines the boundaries of the business category under consideration.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>businessOpportunity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>The strategic interest from a business perspective in order to address the business category under consideration.</td>
</tr>
</tbody>
</table>

Inherited tagged values:
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm
### Stereotype: **BusinessArea**

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>BusinessCategory</td>
</tr>
</tbody>
</table>

**Description**

A business area usually corresponds to a division of an enterprise. Business areas might be structured recursively. A business area (in case of a recursive structure only a business area on the lowest level) is a category of decomposable business process areas. This means a business area collates either other business areas or process areas.

The UMM does not mandate a specific classification schema. A classification schema that might be used is the Porter Value Chain. Based on the Porter Value Chain the UN/CEFACT Common Business Process Catalog recommends a list of eight flat (i.e. non-recursive) categories: Procurement/Sales, Design, Manufacture, Logistics, Recruitment/Training, Financial Services, Regulation, Health Care. This list of business areas is considered as non exhaustive.

**Tag Definition**

Inherited tagged values:
- objective
- scope
- businessOpportunity
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm

---

### Stereotype: **ProcessArea**

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
<td>BusinessCategory</td>
</tr>
</tbody>
</table>

**Description**

A process area corresponds to a set of common operations within a business area. Process areas might be structured recursively. A process area (in case of a recursive structure only a process area on the lowest level) is a category of common business processes. This means a process area collates either other process areas or business processes.

The UMM does not mandate a specific classification schema. The UN/CEFACT Common Business Process Catalog recommends a list of five flat (i.e. non-recursive) categories that correspond to the five successive phases of business collaborations as defined by the ISO Open-edi model: Planning, Identification, Negotiation, Actualization, Post-Actualization.

**Tag Definition**

Inherited tagged values:
- objective
- scope
- businessOpportunity
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Stakeholder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Actor</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>A stakeholder is a person or representative of an organization who has a stake – a vested interest – in a certain business category or in the outcome of a business process. A stakeholder does not necessarily participate in the execution of a business process.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the vested interest of the stakeholder in the business category it is defined within.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessPartnerType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Actor</td>
</tr>
<tr>
<td>Parent</td>
<td>Stakeholder</td>
</tr>
<tr>
<td>Description</td>
<td>A business partner type is an organization type, an organizational unit type or a person type that participates in a business process. Business partner types typically provide input to and/or receive output from a business process. Due to the fact that a business partner type participates in a business process she or he has by default a vested interest in the business process. It follows that a business partner type is a special kind of stakeholder.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>Inherited tagged values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- interest</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>UseCase</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>A business process is a set of related activities that together create value for a business partner. A business process might be performed by a single business partner type or by multiple business partner types crossing organizational boundaries. In case organizations collaborate in a business process, the business process should create value for all its participants. A business process can be decomposed into sub-processes using the «include» and «extends» association stereotypes defined in UML.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>definition</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Gives a definition of the business process. This definition must describe the customer value to be created by the business process. In case of a business process executed by multiple parties it describes the value to be created to all participants.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>beginsWhen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Specifies a business event that triggers the initiation of the business process.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>preCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Specifies conditions that have to be fulfilled in order to execute a business process. This condition SHOULD refer to states in a business entity life cycle. A pre-condition statement MAY use Boolean operators specifying a combination of multiple business entity states.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>endsWhen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Specifies a business event that leads to the termination of the business process.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>postCondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Specifies a condition that will be reached after executing the business process. Usually, this condition SHOULD refer to states in a business entity life cycle. A post-condition statement MAY use Boolean operators specifying a combination of multiple business entity states.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1..*</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Identifies situations leading to a deviation of the regular execution of the business process.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1..*</td>
</tr>
<tr>
<td>Description</td>
<td><strong>Lists the tasks that together make up a business process. In case of a business process executed by multiple parties a special emphasis on interface tasks is needed. An interface task is a business process step that requires communication with another business partner type.</strong></td>
</tr>
</tbody>
</table>
### Base Class

**Association**

**Parent**

N/A

**Description**

Describes the association between a business partner type and a business process. This stereotype defines that the business partner type provides input to and/or output from the associated business process.

### Tag Definition

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interest</td>
<td>String</td>
<td>1</td>
<td>Describes the vested interest of the business partner type in the business process associated by this participates-association.</td>
</tr>
</tbody>
</table>

### Stereotype

**isOfInterestTo**

**Base Class**

Dependency

**Parent**

N/A

**Description**

Describes a dependency from a business process to a stakeholder. This stereotype defines that a business process depends on the interest of the connected stakeholder.

### Tag Definition

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interest</td>
<td>String</td>
<td>1</td>
<td>Describes the vested interest of the stakeholder in the business process linked by this participates-dependency.</td>
</tr>
</tbody>
</table>

### 5.1.3 Constraints (normative)

The BusinessDomainView package MUST include at least one BusinessCategory package or at least one BusinessArea package. Furthermore the BusinessDomainView may contain Stakeholders and BusinessPartnerTypes. The BusinessDomainView MUST NOT include a combination of BusinessCategory and BusinessArea packages.

```plaintext
package Model_Management  
context Package  

inv isBusinessDomainViewPackage:  
    self.isBusinessDomainView() implies  
    self.contents->notEmpty() and (  
    self.contents->forall(isJustBusinessCategory() or  
    isStakeholderOrBusinessPartnerType()) or  
    self.contents->forall(isBusinessArea() or  
    isStakeholderOrBusinessPartnerType())  
```
A BusinessArea package MUST include one or more BusinessArea packages or one or more ProcessArea packages. It MUST NOT include combinations of BusinessArea and ProcessArea packages. It MAY contain BusinessPartnerTypes and Stakeholders.

package Model_Management
context Package

inv contentsOfBusinessArea:
  self.isBusinessArea() implies
  self.contents->notEmpty() and (
  self.contents->forAll(isProcessArea())
  or isStakeholderOrBusinessPartnerType())
  or self.contents->forAll(isBusinessArea() or
  isStakeholderOrBusinessPartnerType())

Either a ProcessArea contains one or more other ProcessAreas and zero or more BusinessPartnerTypes and Stakeholders or it MUST contain at least one BusinessProcess and MAY include BusinessPartnerTypes, Stakeholders and well as stereotyped associations participates and stereotyped dependencies isOfInterestTo.

package Model_Management
context Package

inv contentsOfProcessArea:
  self.isProcessArea() implies
  self.contents->notEmpty() and
  (self.contents->forAll(isProcessArea() or
  isStakeholderOrBusinessPartnerType()) or
  (self.contents->forAll(isBusinessProcess()) or isBusinessPartnerType() or
  isStakeholder() or isParticipates() or isIsOfInterestTo())
  and self.contents->select(isBusinessProcess())->size()>= 1)

Either a BusinessCategory contains one or more BusinessCategories and zero or more BusinessPartnerTypes and Stakeholders or it MUST contain at least one BusinessProcess and MAY include BusinessPartnerTypes, Stakeholders as well as stereotyped associations participates and stereotyped dependencies isOfInterestTo.

package Model_Management
context Package

inv contentsOfBusinessCategory:
  self.isBusinessCategory() implies
  self.contents->notEmpty() and
  (self.contents->forAll(isBusinessCategory() or
  isStakeholderOrBusinessPartnerType()) or
  (self.contents->forAll(isBusinessProcess())
  or isBusinessPartnerType() or
  isStakeholder() or isParticipates() or isIsOfInterestTo())
  and self.contents->select(isBusinessProcess())->size()>= 1)
A participates association that is part of a BusinessCategory (or its specialization ProcessArea) MUST always connect a BusinessPartnerType and a BusinessProcess.

```plaintext
class Association

inv isParticipatesConnector:
  (self.isParticipates() and self.namespace.isBusinessCategory()) implies
  self.allConnections->size() = 2 and
  self.allConnections->one(isBusinessProcess()) and
  self.allConnections->one(isBusinessPartnerType())
```

An isOfInterestTo dependency MUST always be established from a BusinessProcess to a Stakeholder.

```plaintext
class Dependency

inv isIsOfInterestTo:
  self.isIsOfInterestTo() implies
  self.client->one(isBusinessProcess()) and
  self.supplier->one(isStakeholder()) and
  self.client->size() = 1 and
  self.supplier->size() = 1
```

### 5.1.4 Example (informative)

![Diagram showing BusinessDomainView Example: Negotiation (Order from Quote)](image_url)

Figure 9 BusinessDomainView Example: Negotiation (Order from Quote)
5.1.5 OCL methods used in all packages of the BDV (normative)

## OCL-Methods

```ocl
package Foundation::Core
context ModelElement

-- checks if a model element has a certain stereotype
def:
let hasStereotype (st : String) : Boolean =
    self.stereotype->select(self.name = st)->notEmpty()

-- checks if a Package is stereotyped as
-- BusinessDomainView
def:
let isBusinessDomainView() : Boolean =
    self.oclIsKindOf(Package) and
    self.hasStereotype('BusinessDomainView')

-- checks if a Package is a BusinessCategory. This includes
-- also BusinessAreas and ProcessAreas due to the inheritance hierachy
-- in the metamodel
def :
let isBusinessCategory() : Boolean =
    self.oclIsKindOf(Package) and (self.hasStereotype('BusinessCategory') or
    isBusinessArea() or
    isProcessArea() )

-- checks if an Association is stereotyped as participates
def:
let isParticipates() : Boolean =
    self.oclIsKindOf(Association) and
    self.hasStereotype('participates')

-- checks if an Association is stereotyped as isOfInterestOf
def:
let isIsOfInterestTo() : Boolean =
    self.oclIsKindOf(Dependency) and
    self.hasStereotype('isOfInterestTo')

-- checks if a package is a ProcessArea
def:
let isProcessArea() : Boolean =
    self.oclIsKindOf(Package) and
    self.hasStereotype('ProcessArea')

-- checks if a package is a BusinessArea
def:
let isBusinessArea() : Boolean =
    self.oclIsKindOf(Package) and
    self.hasStereotype('BusinessArea')
```

UN/CEFACT – UMM Foundation Module Version 1.0 – Technical Specification 24
-- checks if an Actor is a BusinessPartnerType
def :
let isBusinessPartnerType() : Boolean =
    self.oclIsTypeOf(Actor) and
    self.hasStereotype('BusinessPartnerType')

-- checks if an Actor is a Stakeholder
def :
let isStakeholder() : Boolean =
    self.oclIsTypeOf(Actor) and (self.hasStereotype['Stakeholder'] or
    isBusinessPartnerType())

-- checks if an Actor is a BusinessPartnerType or a Stakeholder
def :
let isStakeholderOrBusinessPartnerType() : Boolean =
    self.isStakeholder() or self.isBusinessPartnerType()

-- checks if a UseCase is stereotyped as BusinessProcess
def :
let isBusinessProcess() : Boolean =
    self.oclIsTypeOf(UseCase) and
    self.hasStereotype('BusinessProcess')
5.2 Business Requirements View

5.2.0 Sub-Views in the Requirements View

5.2.0.1 Conceptual Description (informative)

Figure 10 BusinessRequirementsView Conceptual Overview

The business requirements view is the second out of the 3 views of a UMM compliant business collaboration model. The goal of the BRV is to identify collaborative business processes between different business partner types and to describe the requirements regarding these collaborative business processes. The BusinessRequirementsView packages serves a container for three different artifacts that help to capture the requirements of a collaborative business process:

A business process view describes the flow of activities and states of business processes discovered before in the business domain view. A business process view is not mandatory, but a business requirements view may consist of multiple business process views. Thus, the BusinessRequirementsView is composed of zero to many BusinessProcessViews. A business entity view describes the life cycles of business entities that are manipulated in a collaborative business process. The business entity view is also an optional part that may be repeated. Thus, the BusinessRequirementsView is composed of zero to many BusinessEntityViews.

Finally, the business requirements view covers the partnership requirements view describing the requirements on a partnership between business partner types. A partnership on the lowest level of granularity is a business transaction (see further below). Business collaborations are partnerships that are built by business transactions and/or other business collaborations. Accordingly, a transaction...
requirement view describes the requirements of a business transaction and a collaboration requirements view describes the requirements of a business collaboration. The same business collaboration may be executed between multiple different sets of business partner types. A collaboration realization view describes the requirements of a realization of a business collaboration use case for a specific set of business partner types. A PartnershipRequirementsView is an abstract concept that is either realized by a TransactionRequirementsView, a CollaborationRequirementsView, or a CollaborationRealizationView. The goal of a project (for which a model is developed) is to describe at least one business collaboration and a business collaboration consists of at least one business transaction. At least one of the business collaborations must be executed by a set of business partner types. It follows that the BusinessRequirementsView is composed of one to many CollaborationRequirementsViews, of one or many TransactionRequirementsViews, and of one to many CollaborationRealizationViews.

5.2.0.2 Stereotypes and Tag Definitions (normative)

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessProcessView</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td>Description</td>
<td>The business process view is a container for elements describing the behavior of an internal business process of a business partner type or the behavior of a business process that connects the internal processes of business partner types.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Inherited tagged values:</td>
</tr>
<tr>
<td></td>
<td>- baseURN</td>
</tr>
<tr>
<td></td>
<td>- owner</td>
</tr>
<tr>
<td></td>
<td>- copyright</td>
</tr>
<tr>
<td></td>
<td>- reference</td>
</tr>
<tr>
<td></td>
<td>- version</td>
</tr>
<tr>
<td></td>
<td>- status</td>
</tr>
<tr>
<td></td>
<td>- businessTerm.</td>
</tr>
</tbody>
</table>

Figure 11 BusinessRequirementsView Abstract Syntax
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessEntityView</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td>Description</td>
<td>The business entity view is a container to describe a business entity having business significance in the modelled domain including its business entity lifecycle and business entity states.</td>
</tr>
</tbody>
</table>

**Tag Definition**

**Inherited tagged values:**
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PartnershipRequirementsView (abstract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessLibraryPackage (from Base Module)</td>
</tr>
<tr>
<td>Description</td>
<td>The partnership requirements view is a container for all elements describing the requirements on a partnership between business partner types. These requirements do either apply to a business collaboration, a business transaction or the realization of a business collaboration. Due to this fact the partnership requirements view is split into three specializations the collaboration requirements view, the transaction requirements view, and the collaboration realization view. Since the partnership requirements view is an abstract stereotype one of its specializations must be used.</td>
</tr>
</tbody>
</table>

**Tag Definition**

**Inherited tagged values:**
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CollaborationRequirementsView</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package</td>
</tr>
<tr>
<td>Parent</td>
<td>PartnershipRequirementsView</td>
</tr>
<tr>
<td>Description</td>
<td>The collaboration requirements view is a container for all elements describing the requirements on a business collaboration between authorized roles.</td>
</tr>
</tbody>
</table>

**Tag Definition**

**Inherited tagged values:**
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm.
### 5.2.0.3 Constraints (normative)

A *BusinessRequirementsView* MUST contain at least one *CollaborationRequirementsView* package. It MUST contain at least one *TransactionRequirementsView* package. It MUST contain at least one *CollaborationRealizationView*. It MAY contain *BusinessProcessView* packages and *BusinessEntityView* packages. It MUST NOT contain any other elements.

```plaintext
package Model_Management
context Package

inv packagesAllowedInBRV:
    self.isBusinessRequirementsView() implies
    self.contents->forAll(isBusinessProcessView() or
    isBusinessEntityView() or
    isCollaborationRequirementsView() or
    isTransactionRequirementsView() or
    isCollaborationRealizationView()) and
    self.contents->exists(isCollaborationRequirementsView) and
    self.contents->exists(isTransactionRequirementsView) and
    self.contents->exists(isCollaborationRealizationView)
```
5.2.1 Business Process View

5.2.1.1 Conceptual Description (informative)

The business process view gives an overview about the business processes, their activities and the business partner types that execute these activities. A business process view package includes one or more business processes. If more than one business process is included, the business processes should relate to each other. Accordingly, the BusinessProcessView is composed of one to many BusinessProcesses. Business Processes might include or extend other business processes. This is denoted by the unary composition assigned to BusinessProcess.

The business process activity model represents the dynamic behavior of a business process. It depends on the relevance of a business process whether its flow is described by a business process activity model or not. Thus, a BusinessProcess is composed of 0 or 1 BusinessProcessActivityModel. A business process activity model describes a flow of activities performed by one participant or even by more participants. If two or more business partner types collaborate, a business process activity model is divided into partitions – one for each business partner type. In case of an internal business process, which is executed by one partner only, a single partition for that partner is optional. Consequently, a BusinessProcessActivityModel...
A business process activity model is described as a flow of business process activities. In case that no partition is used, the business process activities are directly included in the business process activity model. In case of partitions, a business process activity is assigned to the partition of the business partner type executing the activity. The need for a collaborative business process is identified whenever a transition connecting two business process activities crosses between partitions. It follows, that either a BusinessProcessActivityModel is composed of one or more BusinessProcessActivities or a Partition (which is part of a business process activity model) is composed of one or more BusinessProcessActivities. A business process activity might be refined by another business process activity model. Thus a BusinessProcessActivity is composed of zero or one BusinessProcessActivityModels which in turn is a composite of zero or one BusinessProcessActivity.

A business process activity model may also denote important states of business entities that are manipulated during the execution of a business process. A business entity state is the output from one business activity and input to another business activity. There is a transition from a business process activity to a business entity state signaling an output as well as a transition from a business entity state to a business process activity signaling an input. Some business entity states are meaningful to one business partner type only. These are internal business entity states. Business entity states that must be communicated to a business partner type are shared business entity states. A business process activity model may include both internal and shared business entity states. Hence, a BusinessProcessActivity model is composed of zero to many InternalBusinessEntityStates and of zero to many SharedBusinessEntityStates. If a business process activity model uses partitions, the two business process activities creating and consuming an internal business entity state are in the same partition. In contrast, the two business process activities creating and consuming a shared business entity state are in different partitions. A shared business entity state signals the need for a collaborative business process.

5.2.1.2 Stereotypes and Tag Definitions (normative)
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessProcessActivityModel</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>ActivityGraph</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The BusinessProcessActivityModel describes the behavior of the business processes of the involved BusinessPartnerTypes. It is a tool to identify requirements to collaborate between two or more BusinessPartnerTypes. A BusinessProcessActivityModel is linked to a BusinessProcess identified in the BusinessDomainView and describes the dynamic behavior of that BusinessProcess.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessProcessActivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>State</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A business process activity corresponds to a step in the execution of a business process activity model. A business activity might be refined by another business process activity model. Thus, the UML base class of business process activity is not an atomic action state, but a state – which is a generalization of action state and composite state.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>InternalBusinessEntityState</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>ObjectFlowState</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The InternalBusinessEntityState represents a state of a BusinessEntity that is internal to the business process of a certain BusinessPartnerType.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>SharedBusinessEntityState</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>ObjectFlowState</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The SharedBusinessEntityState represents a state of a BusinessEntity that is shared between the business processes of two involved BusinessPartnerTypes.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values.</td>
</tr>
</tbody>
</table>
5.2.1.3 Constraints (normative)

The BusinessProcessView MUST contain nothing else, but BusinessProcessActivityModels, BusinessPartnerTypes and BusinessProcesses and it must be empty.

package Model_Management
context Package

inv AllowedElementsInBusinessProcessView:
  self.isBusinessProcessView() implies
  self.contents->forall(isBusinessProcessActivityModel() or
  isBusinessPartnerType() or
  isBusinessProcess()) and
  self.contents->notEmpty()

A BusinessProcessActivityModel, which has no partitions, MUST contain one or more BusinessProcessActivities and MAY contain InternalBusinessEntityStates, SharedBusinessEntityStates, pseudo states, final states and transitions.

package Behavioral_Elements::State_Machines
context CompositeState

inv AllowedElementsInBusinessProcessActivityModelWithoutPartition:
  (self.stateMachine.isBusinessProcessActivityModel() and
  self.stateMachine.oclAsType(ActivityGraph).partition->isEmpty()) implies
  self.subvertex->notEmpty() and
  self.subvertex->exists(isBusinessProcessActivity()) and
  self.subvertex->forall(isBusinessProcessActivity() or
  isInternalBusinessEntityState() or
  isSharedBusinessEntityState() or
  isPseudoStateOrFinalStateOrTransition())

A partition in a BusinessProcessActivityModel MUST contain one or more BusinessProcessActivities and MAY contain InternalBusinessEntityStates, PseudoStates, FinalStates and Transitions.

package Behavioral_Elements::Activity_Graphs
context Partition

inv AllowedModelElementsInBusinessProcessActivityModelPartition:
  self.isPartition() implies
  self.contents->forall(isBusinessProcessActivity() or
  isInternalBusinessEntityState() or
  isPseudoStateOrFinalStateOrTransition() ) and
  self.contents->exists(isBusinessProcessActivity())
5.2.1.4 Example (informative)

Figure 14 BusinessProcessView (BusinessRequirementsView) Example: Purchase Product – Collaborative View

BusinessProcessActivityModel (ActivityGraph)
5.2.2 Business Entity View

5.2.2.1 Conceptual Description (informative)

A business entity is a real-world thing having business significance that is shared among two or more business partner types in a collaborative business process (e.g. “order”, “account”, etc.). Within the business domain view at least one, but possibly more business entities are described. Thus, the BusinessEntityView is composed of one to many BusinessEntities. It depends on the importance of the business entity lifecycle, whether its life cycle is included or not. Hence, a BusinessEntity is composed of zero to one BusinessEntityLifecycles. A business entity lifecycle represents the different business entity states a business entity can exist in. A business entity lifecycle consist of at least one business entity state. Inasmuch, the BusinessEntityLifecycle is composed of one or more BusinessEntityStates. Like any other UML state machine the business entity life cycle includes events and transitions including optional guards that lead from one business entity state to another one.

5.2.2.2 Stereotypes and Tag Definitions (normative)
A business entity is a real-world thing having business significance that is shared among two or more business partner types in a collaborative business process (e.g. order, account, etc.).

Description

No tagged values.

---

A business entity lifecycle represents the different business entity states a business entity can exist in and the events and transitions that lead from one business entity state to another business entity state of the same business entity.

Description

No tagged values.

---

A business entity state represents a certain state a business entity can exist in during its lifecycle (an “order” can exist in the states “issued”, “rejected”, “confirmed”, etc.)

Description

No tagged values.

---

### 5.2.2.3 Constraints (normative)

The *BusinessEntityView* MUST contain nothing else than *BusinessEntities*

```plaintext
package Model_Management
context Package

inv AllowedElementsInBusinessEntityView:
    self.isBusinessEntityView() implies
    self.contents->notEmpty() and
    self.contents->forAll(isBusinessEntity())
```

A *BusinessEntity* has zero or one *BusinessEntityLifecycle* that expresses its behavior

```plaintext
package Foundation::Core
context Class

inv LifecyclesOfBusinessEntity:
    self.isBusinessEntity() implies
    self.behavior->select(isBusinessEntityLifecycle())->size()<=1
```
A BusinessEntityLifecycle MUST only contain BusinessEntityStates, PseudoStates, FinalStates or Transitions

package Behavioral_Elements::State_Machines
context CompositeState

inv ContainsOnlyBusinessEntityStates:
self.stateMachine.isBusinessEntityLifecycle() implies
self.subvertex->forAll(isBusinessEntityState() or
isPseudoStateOrFinalStateOrTransition())
and self.subvertex->exists(isBusinessEntityState())
Figure 19 BusinessEntityView (BusinessRequirementsView) Example: Order BusinessEntityLifecycle (StateMachine)
5.2.3 Partnership Requirements View

5.2.3.1 Conceptual Description (informative)

The previous views helped to identify the need for a collaboration. The business partnership view describes the requirements of an identified collaboration between business partner types by the means of use cases. In this use case analysis we distinguish between business collaboration use cases, business transaction use cases, and business collaboration realizations. A business transaction use case describes the requirements of a transaction that is a special interaction between two authorized roles that is limited to an initiating information exchange and an optional response. A business collaboration use case describes the requirements of a business collaboration that is executed between two or more authorized roles, and that is composed of one or more business transactions or nested business collaborations. A business collaboration use case must be executed by a set of business partner types. Different sets of business partner types may realize the same business collaboration use case. A business collaboration realization is a realization of a business collaboration by a specific set of business partner types.

A partnership requirements view is an abstract concept. It is either a collaboration requirements view capturing the requirements of a business collaboration, a transaction requirements view capturing the
requirements of a business transaction, or a collaboration realization view capturing the requirements of a
business collaboration realization. Thus, the CollaborationRequirementsView, the
TransactionRequirementsView, and the CollaborationRealizationView are specializations of the abstract
PartnershipRequirementsView.

Each business collaboration use case is defined in its own collaboration requirements view. Accordingly,
the CollaborationRequirementsView is composed of exactly one BusinessCollaborationUseCase. Two or
more authorized roles participate in a business collaboration use case. These authorized roles (e.g. seller,
payee) must be defined in the same collaboration requirements view package as the corresponding
business collaboration use case. Accordingly, a CollaborationRequirementsView is composed of two or
more AuthorizedRoles. This means, if a certain role (e.g. seller, payee) participates in multiple business
collaborations, it requires a different authorized role for each business collaboration use case. Each
authorized role of the same role is in a different namespace of a corresponding collaboration requirements
view. Therefore, an authorized role participates in only one business collaboration use case– it is the one
in the same collaboration requirements view. Accordingly, BusinessCollaborationUseCase and
AuthorizedRole are related by an 1 to (2..n) association. It is important, that the same authorized role must
not be associated twice or more times to the same business collaboration use case.

Each business transaction use case is defined in its own transaction requirements view. Accordingly, the
TransactionRequirementsView is composed of exactly one BusinessTransactionUseCase. Two authorized
roles participate in a business transaction use case. These authorized roles (e.g. seller, payee) must be
defined in the same transaction requirements view package as the corresponding business transaction use
case. Accordingly, a TransactionRequirementsView is composed of exactly two AuthorizedRoles. This
means, if a certain role (e.g. seller, payee) participates in multiple business transactions, it requires a
different authorized role for each business collaboration use case. Each authorized role of the same role is
in a different namespace of a corresponding transaction requirements view. Therefore, an authorized role
participates in only one business transaction use case– it is the one in the same transaction requirements
view. Accordingly, BusinessTransactionUseCase and AuthorizedRole are related by an 1 to 2 association.
It is important to note, that the same authorized role is not associated twice to the same business
transaction use case.

A business collaboration use case may include nested business collaboration use cases. A business
collaboration use case may be optionally nested in multiple parent business collaboration use cases.
Hence, BusinessCollaborationUseCase has a unary (0..n) to (0..n) include-composition. A business
collaboration use case may include multiple business transaction use cases. A business transaction use
case must be included in at least one business collaboration use case. Consequently, an (1..n) to (0..n)
aggregation between BusinessCollaborationUseCase and BusinessTransactionUseCase exists. It is
important that a business collaboration use case includes at minimum one use case – no matter whether
this is a nested business collaboration use case or a business transaction use case. A hierarchy of business
collaboration use cases built by include-compositions must not include any cycles. A business transaction
uses case cannot be further decomposed by an include-association. UMM does not use any extend-
associations between business collaboration/transaction use cases.

For each include-relationship either between a business collaboration use case and a business transaction
use case or between two collaboration use cases, a mapping of the authorized role of the source use case
to the authorized roles of the target use case is necessary. Accordingly, the AuthorizedRole has a unary
mapsTo-relationship of (1..n) to (1..n). It is required that each authorized role of the target use case is the
target of a mapping from an authorized role of the source use case. Each authorized role of the source use
case may be mapped maximal once to an authorized role of the same target use case, but it may be
mapped to different authorized roles of different target use cases.
Business partner types identified in the previous UMM steps must not directly be associated with the business collaboration use cases and business transaction use cases. In order to specify that a specific set of business partner types collaborate, we use the concept of a business collaboration realization. Each business collaboration realization is defined in its own collaboration realization view. Accordingly, the CollaborationRealizationView is composed of exactly one BusinessCollaborationRealization. A business collaboration realization realizes exactly one business collaboration use case. Each business collaboration use case may be realized by multiple business collaboration realizations. Not each business collaboration use case (e.g. one that is nested within another one) needs to have a corresponding business collaboration realization. As a consequence, the realize-association between a BusinessCollaborationUseCase and BusinessCollaborationRealization is a 1 to (0..n).

Two or more authorized roles participate in a business collaboration realization. These authorized roles (e.g. seller, payee) must be defined in the same collaboration realization view package as the corresponding business collaboration realization. Accordingly, a CollaborationRealizationView is composed of two or more AuthorizedRoles. Usually, the names of the authorized roles participating in the business collaboration use case (e.g. payer and payee) will be the names of the authorized roles in the business collaboration realization (e.g. payer and payee) realizing it. However, the authorized roles participating in the business collaboration use case and in the business collaboration realization will be defined in different namespaces – each in the package of the corresponding view. Similar to the BusinessCollaborationUseCase, the BusinessCollaborationRealization and AuthorizedRole are related by an 1 to (2..n) association. Furthermore, the number of actors participating in a business collaborations use case must be the same as the number of actors participating in the business collaboration realization realizing it.

In order to bind a business collaboration realization to the business partner types executing it, business partner types are mapped to the authorized roles participating in the business collaboration realization. It is required that each authorized role of a business collaboration realization (but not an authorized role in general) is target of exactly one mapsTo-association from a business partner type. A business partner type may play multiple authorized roles of a business collaboration realization. Consequently, there is a (0..1) to (0..n) mapsTo-association between BusinessPartnerType and AuthorizedRole.
A business collaboration use case describes in detail the requirements on a collaboration between two or more involved partners. Business partner types take part in a business collaboration use case by playing an authorized role in it. A business collaboration use case can be broken down into further business collaboration use cases and business transaction use cases.

**Tag Definition**
- definition
- beginsWhen
- preCondition
- endsWhen
- postCondition
- exceptions
- actions
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessTransactionUseCase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>UseCase</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>BusinessProcess</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A business transaction use case describes in detail the requirements on a collaboration between exactly two involved partners. A business transaction use case can not be further refined and describes the requirements on a one-way or two-way information exchange. Business partner types take part in a business transaction use case by playing an authorized role in it.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>Inherited tagged values:</td>
</tr>
<tr>
<td></td>
<td>- definition</td>
</tr>
<tr>
<td></td>
<td>- beginsWhen</td>
</tr>
<tr>
<td></td>
<td>- preCondition</td>
</tr>
<tr>
<td></td>
<td>- endsWhen</td>
</tr>
<tr>
<td></td>
<td>- postCondition</td>
</tr>
<tr>
<td></td>
<td>- exceptions</td>
</tr>
<tr>
<td></td>
<td>- actions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessCollaborationRealization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Collaboration</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A business collaboration realization realizes a business collaboration use case between a specific set of business partner types. The requirements of the business collaboration realization are the ones defined in the tags of the corresponding business collaboration use case. Thus, the business collaboration realization does not include any tag definitions for capturing requirements.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>AuthorizedRole</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Actor</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>An authorized role (e.g. a “buyer”) is a concept which is more generic than a business partner type (e.g. a “broker”) and allows the reuse of collaborations by mapping an AuthorizedRole to a business partner type within a given scenario. Since business collaboration use case and business transaction use case are defined as occurring between authorized roles, they might be reused by different business partner types (a “broker” or a “custodian”) in different scenarios of the same domain or even in different domains.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>mapsTo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Dependency</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A maps to dependency represents (1) the fact that a business partner type plays a certain authorized role in a business collaboration realization and (2) the fact that an authorized role of a source business collaboration use case takes on a certain authorized role in a target business transaction use case or business collaboration use case.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No tagged values</td>
</tr>
</tbody>
</table>
5.2.3.3 Constraints (normative)

<table>
<thead>
<tr>
<th>The <strong>CollaborationRequirementsView</strong> MUST contain exactly one <strong>BusinessCollaborationUseCase</strong>, at least two <strong>AuthorizedRoles</strong>, and at least two <strong>participates</strong> associations.</th>
</tr>
</thead>
</table>
| package Model_Management
context Package

inv AllowedElementsInCollaborationRequirementsView:
  self.isCollaborationRequirementsView() implies
  self.contents->notEmpty() and
  self.contents->select(isAuthorizedRole())->size()>=2 and
  self.contents->one(isBusinessCollaborationUseCase()) and
  self.contents->select(isParticipates())->size()>=2 and
  self.contents->forAll(isAuthorizedRole() or
  isBusinessCollaborationUseCase() or isParticipates()) |

<table>
<thead>
<tr>
<th>The <strong>TransactionRequirementsView</strong> MUST contain exactly one <strong>BusinessTransactionUseCase</strong>, exactly two <strong>AuthorizedRoles</strong>, and exactly two <strong>participates</strong> associations</th>
</tr>
</thead>
</table>
| package Model_Management
context Package

inv AllowedElementsInTransactionRequirementsView:
  self.isTransactionRequirementsView() implies
  self.contents->notEmpty() and
  self.contents->select(isAuthorizedRole())->size()==2 and
  self.contents->one(isBusinessTransactionUseCase()) and
  self.contents->select(isParticipates())->size()==2 and
  self.contents->forAll(isAuthorizedRole() or
  isBusinessTransactionUseCase() or isParticipates()) |

<table>
<thead>
<tr>
<th>The <strong>CollaborationRealizationView</strong> MUST contain exactly one <strong>BusinessCollaborationRealization</strong>, at least two <strong>Authorized Roles</strong>, and at least two <strong>participates</strong> associations</th>
</tr>
</thead>
</table>
| package Model_Management
context Package

inv AllowedElementsInRealizationView:
  self.isCollaborationRealizationView() implies
  self.contents->notEmpty() and
  self.contents->select(isAuthorizedRole())->size()>=2 and
  self.contents->one(isBusinessCollaborationRealization()) and
  self.contents->select(isParticipates())->size()>=2 and
  self.contents->forAll(isBusinessCollaborationRealization() or
  isParticipates() or isAuthorizedRole()) |
A **BusinessCollaborationUseCase** MUST be associated with two or more *AuthorizedRoles* via stereotyped binary *participate* associations

```java
package Behavioral_Elements::Use_Cases
context UseCase
inv BusinessCollaborationUCAssociatedWith2AuthorizedRoles:
    self.isBusinessCollaborationUseCase() implies
    self.associations->size() >= 2 and
    self.associations->forall(a | a.isParticipates() and
    a.allConnections->exists(isAuthorizedRole())
    and a.connection->size=2)
```

A **BusinessTransactionUseCase** MUST be associated with exactly two *AuthorizedRoles* via stereotyped binary *participate* associations

```java
package Behavioral_Elements::Use_Cases
context UseCase
inv BusinessTransactionUCAssociatedWith2AuthorizedRoles:
    self.isBusinessTransactionUseCase() implies
    self.associations->size() = 2 and
    self.associations->forall(a | a.isParticipates() and
    a.allConnections->exists(isAuthorizedRole())
    and a.connection->size=2)
```

A **BusinessCollaborationRealization** MUST be associated with two or more *AuthorizedRoles* via stereotyped binary *participate* associations

```java
package Behavioral_Elements::Use_Cases
context UseCase
inv BusinessCollaborationRealizationAssociatedWith2AuthorizedRoles:
    self.isBusinessCollaborationRealization() implies
    self.associations->size() >= 2 and
    self.associations->forall(a | a.isParticipates() and
    a.allConnections->exists(isAuthorizedRole())
    and a.connection->size=2)
```
A BusinessCollaborationRealization MUST be the client of exactly one realization dependency to a BusinessCollaborationUseCase.

```plaintext
package Behavioral_Elements::Use_Cases
context UseCase

inv BusinessCollaborationRealizationRealizesOneBusinessCollaborationUseCase:
  self.isBusinessCollaborationRealization() implies
  self.clientDependency->size()=1 and
  self.clientDependency->forall(d | d.isRealization() and
  d.supplier->size()=1 and
  d.supplier->forall(isBusinessCollaborationUseCase()))
```

A BusinessCollaborationUseCase MUST include one or more other BusinessCollaborationUseCases or one or more BusinessTransactionUseCases, but at least one of them.

```plaintext
package Behavioral_Elements::Use_Cases
context UseCase

inv AllowedIncludesOfBCUC:
  self.isBusinessCollaborationUseCase() implies
  self.include->notEmpty() and
  self.include->forall(i | i.addition.isBusinessCollaborationUseCase() or
  i.addition.isBusinessTransactionUseCase())
```

A BusinessTransactionUseCase MUST not include further UseCases.

```plaintext
package Behavioral_Elements::Use_Cases
context UseCase

inv NoIncludesOfBTUC:
  self.isBusinessTransactionUseCase() implies
  self.include->collect(addition)->isEmpty()
```

A BusinessTransactionUseCase MUST be included in at least one BusinessCollaborationUseCase.

```plaintext
package Behavioral_Elements::Use_Cases
context UseCase

inv BTUCIncludedAtLeastOnce:
  self.isBusinessTransactionUseCase() implies
  self.include->forall(base.isBusinessCollaborationUseCase()) and
  self.include->collect(base)->notEmpty()
```
**A BusinessCollaborationUseCase and a BusinessTransactionUseCase MUST not be source or target of an extend association**

```plaintext
definition package Behavioral_Elements::Use_Cases
class UseCase
    definition inv BTUC_BCUC_IsNoExtendTarget:
        (self.isBusinessTransactionUseCase() or
         self.isBusinessCollaborationUseCase()) implies
        self.extend->isEmpty()
```

**A BusinessCollaborationRealization MUST not be source or target of an include or extends association**

```plaintext
definition package Behavioral_Elements::Use_Cases
class UseCase
    definition inv BusinessCollaborationRealizationNoIncludesAndExtends:
        self.isBusinessCollaborationRealization() implies
        self.extend->isEmpty() and
        self.include->isEmpty()
```

**All dependencies from/to an AuthorizedRole must be mapsTo dependencies.**

```plaintext
definition package Behavioral_Elements::Use_Cases
class Actor
    definition inv AllDependenciesToAndFromAuthorizedRoleMustBeMapsTo:
        self.isAuthorizedRole() implies
        self.clientDependency->forAll(d | d.isMapsToDependency()) and
        self.supplierDependency->forAll(s | s.isMapsToDependency())
```
An AuthorizedRole, which participates in a BusinessCollaborationRealization, must be the supplier of exactly one mapsTo dependency to a BusinessPartnerType. Furthermore the AuthorizedRole, which participates in the BusinessCollaborationRealization must be the client of exactly one mapsTo dependency to an AuthorizedRole participating in a BusinessCollaborationUseCase.

package Behavioral_Elements::Use_Cases
context Actor

inv BCRAuthorizedRoleIsMappedByOnlyOneBusinessPartnerType:
{self.isAuthorizedRole() and
 self.namespace.isCollaborationRealizationView()} implies
self.supplierDependency->size()=1 and (self.supplierDependency->forAll(c | c.client->size()=1 and self.supplierDependency->forAll(c.client->forAll(isBusinessPartnerType()))))
and self.clientDependency->size()=1 and (self.clientDependency->forAll(s | s.supplier->size()=1 and self.clientDependency->forAll(s | s.supplier->forAll(isAuthorizedRole() and s.namespace.isCollaborationRequirementsView())))}

A source BusinessCollaborationUseCase includes target BusinessTransactionUseCases and/or BusinessCollaborationUseCases. Each authorized role of the source use case must be mapped maximal once to an authorized role of the same target use case (but it may be mapped to different AuthorizedRoles of different target use cases). Each authorized role of the target use case is the supplier of a mapsTo dependency from an authorized role of the source use case.

package Behavioral_Elements::Use_Cases
context UseCase

inv AuthorizedRoleofBTUCisSupplierOfOnlyOneAuthorizedRoleofBCUC:
{self.isBusinessTransactionUseCase() or self.isBusinessCollaborationUseCase()} implies
self.include->select(a | a.base <> self)->collect(base)->collect(x | x.associations)->collect(y | y.allConnections)->select(isAuthorizedRole)->forAll(x | self.associations->collect(allConnections)->select(isAuthorizedRole)->collect(supplierDependency)->collect(client)->isUnique(x))
A *BusinessCollaborationUseCase* MUST have the same count of participating *AuthorizedRoles*, as each *BusinessCollaborationRealization*, realizing it.

```plaintext
package Behavioral_Elements::Use_Cases
context UseCase

inv AuthorizedRoleCountSameForBCUCandRealizingBCR:
  self.isBusinessCollaborationRealization() implies
  self.associations->collect(allConnections)->select(isAuthorizedRole)
  ->size() =
  (self.clientDependency->collect(supplier)->collect(associations)
  ->collect(allConnections)->
  select(isAuthorizedRole)->size())
```

*AuthorizedRoles* in a *TransactionRequirementsView*, *CollaborationRequirementsView* or *CollaborationRealizationView* must have a unique name within the scope of the package, they are located in.

```plaintext
package Model_Management
context Package

inv AuthorizedRolesMustHaveUniqueName:
  self.isTransactionRequirementsView() or
  self.isCollaborationRequirementsView() or
  self.isCollaborationRealizationView() implies
  self.contents->select(isAuthorizedRole())
  ->isUnique(element | element.name)
```
5.2.3.4 Example (informative)

![UseCase Diagram: Order From Quote](image)

**Figure 22 CollaborationRequirementsView (BusinessRequirementsView) Example: OrderFromQuote (UseCase Diagram)**

![UseCase Diagram: Place Order](image)

**Figure 23 TransactionRequirementsView (BusinessRequirementsView) Example: PlaceOrder Transaction**
5.2.4 OCL methods used in all packages of the BRV (normative)

<table>
<thead>
<tr>
<th>OCL-Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>package Foundation::Core</td>
</tr>
<tr>
<td>context ModelElement</td>
</tr>
<tr>
<td>--Predefined method which evaluates, if the given ModelElement --has a stereotype equal to the passed name def: let hasStereotype (st : String) : Boolean = self.stereotype-&gt;select(cst</td>
</tr>
<tr>
<td>--Predefined method which evaluates, if the given element --has the stereotype 'InternalBusinessEntityState' def: let isInternalBusinessEntityState() : Boolean = self.oclIsKindOf(ObjectFlowState) and self.hasStereotype('InternalBusinessEntityState')</td>
</tr>
</tbody>
</table>
--Predefined method which evaluates, if the given element
--has the stereotype 'ShardedBusinessEntityState'
def:
let isSharedBusinessEntityState() : Boolean =
    self.oclIsKindOf(ObjectFlowState) and 
    self.hasStereotype('SharedBusinessEntityState')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessProcessActivity'
def:
let isBusinessProcessActivity() : Boolean =
    self.oclIsKindOf(ObjectFlowState) and 
    self.hasStereotype('BusinessProcessActivity')

-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is initial
def:
let isInitialState() : Boolean =
    self.oclAsType(Pseudostate).kind = PseudostateKind::initial and 
    self.oclIsKindOf(Pseudostate)

-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is choice
def:
let isChoice() : Boolean =
    self.oclAsType(Pseudostate).kind = PseudostateKind::choice and 
    self.oclIsKindOf(Pseudostate)

-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is fork
def:
let isFork() : Boolean =
    self.oclAsType(Pseudostate).kind = PseudostateKind::fork and 
    self.oclIsKindOf(Pseudostate)

-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is join
def:
let isJoin() : Boolean =
    self.oclAsType(Pseudostate).kind = PseudostateKind::join and 
    self.oclIsKindOf(Pseudostate)

-- Returns true if the type of the element or is 'FinalState'
def:
let isFinalState() : Boolean =
    self.oclIsKindOf(FinalState)

-- Returns true if the type of the element 'Transition'
def:
let isTransition() : Boolean =
self.oclIsKindOf(Transition)
--Returns true if the element is a standard-element of an ActivityGraph
def:
let isPseudoStateOrFinalStateOrTransition() : Boolean =
  isInitialState() or isChoice() or isFork() or isJoin() or isTransition()
  or isFinalState()

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessProcessView'
def:
let isBusinessProcessView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessProcessView')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntityView'
def:
let isBusinessEntityView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessEntityView')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessRequirementsView'
def:
let isBusinessRequirementsView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessRequirementsView')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessProcessActivityModel'
def:
let isBusinessProcessActivityModel() : Boolean =
  self.oclIsKindOf(ActivityGraph) and
  self.hasStereotype('BusinessProcessActivityModel')

--return true if the given element is a partition

def:
let isPartition() : Boolean =
  self.oclIsKindOf(Partition)

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntity'
def:
let isBusinessEntity() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('BusinessEntity')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntityState'
def:
let isBusinessEntityState() : Boolean =
  self.oclIsKindOf(State) and
self.hasStereotype('BusinessEntityState')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntityLifecycle'
def:
let isBusinessEntityLifecycle() : Boolean =
    self.oclIsKindOf(StateMachine) and
    self.hasStereotype('BusinessEntityLifecycle')

--return true if the given element is a package
def:
let isPackage() : Boolean =
    self.oclIsKindOf(Package)

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessCollaborationUseCase'
def:
let isBusinessCollaborationUseCase() : Boolean =
    self.oclIsKindOf(UseCase) and
    self.hasStereotype('BusinessCollaborationUseCase')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessTransactionUseCase'
def:
let isBusinessTransactionUseCase() : Boolean =
    self.oclIsKindOf(UseCase) and
    self.hasStereotype('BusinessTransactionUseCase')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessCollaborationRealization'
def:
let isBusinessCollaborationRealization() : Boolean =
    self.oclIsKindOf(Collaboration) and
    self.hasStereotype('BusinessCollaborationRealization')

--Predefined method which evaluates, if the given element
--has the stereotype 'AuthorizedRole'
def:
let isAuthorizedRole() : Boolean =
    self.oclIsKindOf(Actor) and
    self.hasStereotype('AuthorizedRole')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessPartnerType'
def:
let isBusinessPartnerType() : Boolean =
    self.oclIsKindOf(Actor) and
    self.hasStereotype('BusinessPartnerType')

--Predefined method which evaluates, if the given element
--has the stereotype 'mapsTo'
def:
let isMapsToDependency() : Boolean =
    self.oclIsKindOf(Dependency) and
self.hasStereotype('mapsTo')

--Predefined method which evaluates, if the given element
--is a Realization dependency
def:
let isRealization() : Boolean =
  self.oclIsKindOf(Abstraction) and
  self.hasStereotype('realize')

-- checks if an Association is stereotyped as participates
def:
let isParticipates() : Boolean =
  self.oclIsKindOf(Association) and
  self.hasStereotype('participates')

--Predefined method which evaluates, if the given element
--is an Association
def:
let isAssociation() : Boolean =
  self.oclIsKindOf(Association)

--Predefined method which evaluates, if the given element
--has the stereotype 'CollaborationRequirementsView'
def:
let isCollaborationRequirementsView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('CollaborationRequirementsView')

--Predefined method which evaluates, if the given element
--has the stereotype 'TransactionRequirementsView'
def:
let isTransactionRequirementsView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('TransactionRequirementsView')

--Predefined method which evaluates, if the given element
--has the stereotype 'CollaborationRealizationView'
def:
let isCollaborationRealizationView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('CollaborationRealizationView')

-- checks if a UseCase is stereotyped a BusinessProcess
def:
let isBusinessProcess() : Boolean =
  self.oclIsTypeOf(UseCase) and
  self.hasStereotype('BusinessProcess')
5.3 Business Transaction View

5.3.0 Views in the Transaction View

5.3.0.1 Conceptual Description (informative)

The Business Transaction View (BTV) is an elaboration on the business requirements view by the business analyst and is how the business analyst sees the process to be modeled. According to these requirements the BTV defines a choreography of information exchanges. The business transaction view package is a container for three different artifacts that together describe the overall choreography of information exchanges. The business choreography view is a container for artifacts describing the flow of a complex business collaboration between business partner types that may involve many steps. In fact, a business choreography view captures artifacts that define a flow in accordance to the requirements of a corresponding collaboration requirements view of the BRV. A business interaction view is a container for artifacts that define a choreography leading to synchronized states of business entities at both sides of the interaction. In fact, a business interaction view captures artifacts that define a flow in accordance to the requirements of a corresponding transaction requirements view of the BRV. A business information view is a container of artifacts that describe the information exchanged in an interaction. Accordingly, the business choreography view and the business interaction view deal with artifacts describing the dynamic aspects of a collaboration and the business information view deals with artifacts describing the structural aspects of a collaboration. Each of the three views must occur at least once in the business transaction view. Thus the BusinessTransactionView is composed of one to many BusinessChoreographyViews, of one to many BusinessInteractionViews, and of one to many BusinessInformationViews.
5.3.0.2 Stereotypes and Tag Definitions (normative)

### Figure 26 BusinessTransactionView Abstract Syntax

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessChoreographyView</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessLibraryPackage (from BaseModule)</td>
</tr>
<tr>
<td>Description</td>
<td>The business choreography view is a container for artifacts describing the flow of a complex business collaboration between business partner types that may involve many steps.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Inherited tagged values:</td>
</tr>
<tr>
<td></td>
<td>- baseURN</td>
</tr>
<tr>
<td></td>
<td>- owner</td>
</tr>
<tr>
<td></td>
<td>- copyright</td>
</tr>
<tr>
<td></td>
<td>- reference</td>
</tr>
<tr>
<td></td>
<td>- version</td>
</tr>
<tr>
<td></td>
<td>- status</td>
</tr>
<tr>
<td></td>
<td>- businessTerm.</td>
</tr>
</tbody>
</table>
A business interaction view is a container for artifacts that define a choreography leading to synchronized states of business entities at both sides of the interaction.

Inherited tagged values:
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm.

A business information view is a container of artifacts that describe the information exchanged in an interaction.

Inherited tagged values:
- baseURN
- owner
- copyright
- reference
- version
- status
- businessTerm.

5.3.0.3 Constraints (normative)

A BusinessTransactionView MUST contain at least one BusinessChoreographyView package, at least one BusinessInteractionView package, and at least one BusinessInformationView package.

```plaintext
package Model_Management
ccontext Package
inv packagesAllowedInBTV:
  self.isBusinessTransactionView() implies
  self.contents->exists(isBusinessChoreographyView()) and
  self.contents->exists(isBusinessInteractionView()) and
  self.contents->exists(isBusinessInformationView())
```
5.3.1 Business Choreography View

5.3.1.1 Conceptual Description (informative)

![Diagram of Business Choreography View](image)

A business choreography view is used to define the business choreography of exactly one business collaboration. Therefore, the BusinessChoreographyView is composed of exactly one BusinessChoreography. A business choreography is a persistent representation of the execution of a business collaboration. The execution order of a business collaboration, i.e. the choreography of the business collaboration, is defined by the business choreography behavior. Each BusinessChoreography is composed of exactly one BusinessChoreographyBehavior. The business choreography behavior follows exactly the requirements defined in a corresponding business collaboration use case of the BRV. Each business collaboration use case of the BRV is mapped to exactly one business choreography behavior. Hence, a BusinessCollaborationUseCase and the BusinessChoreographyBehavior have a 1 to 1 mapsTo relationship.

Business choreography behavior is an abstract concept. In a future version there might exist different approaches to describe the choreography of a business collaboration. In this version, the only valid specialization of a BusinessChoreographyBehavior is the BusinessCollaborationProtocol. Thus, a business choreography is currently always defined by a business collaboration protocol. The activities of a business collaboration protocol are business collaboration activities and/or business transaction activities. Hence, a BusinessCollaborationProtocol is composed of zero to many BusinessCollaborationActivities and of zero to many BusinessTransactionActivities. However, at least one business collaboration activity or a business transaction activity must be present in a business collaboration.
collaboration protocol. Transitions defining the flow among the business collaboration activities and/or business transaction activities may be guarded by the states of business entities.

A business collaboration activity is characterized by the fact that it is refined by another business collaboration protocol. Not each business collaboration is a refined business collaboration activity – only the nested business collaboration protocols. A business collaboration protocol may be nested in different business collaboration activities. Thus, the aggregation relationship between BusinessCollaborationActivity and BusinessCollaborationProtocol is (0..n) to 1.

A business transaction activity is characterized by the fact that it is refined by a business transaction. Since the business transaction is a concept of the business interaction view it is described in more detail further below. Each business transaction must be at least once used to refine a business transaction activity. A business transaction may be nested in different business transaction activities. Hence, the aggregation relationship between BusinessTransactionActivity and BusinessTransaction is (1..n) to 1.

5.3.1.2 Stereotypes and Tag Definitions (normative)

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessChoreography</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Class</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A business choreography is a persistent representation of the execution of a business collaboration.</td>
</tr>
<tr>
<td><strong>Tag Definition</strong></td>
<td>No Tagged Values</td>
</tr>
</tbody>
</table>

Figure 28 BusinessChoreographyView (BusinessTransactionView) Abstract Syntax
<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessChoreographyBehavior (abstract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>ActivityGraph</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The business choreography behavior defines the dynamic behavior of a business collaboration, i.e. the choreography of a business collaboration.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>No Tagged Values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessCollaborationProtocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>ActivityGraph</td>
</tr>
<tr>
<td>Parent</td>
<td>BusinessChoreographyBehavior</td>
</tr>
<tr>
<td>Description</td>
<td>A business collaboration protocol is a specialization of a business choreography behaviour. It choreographs business transaction activities and/or business collaboration activities. At least one activity of either one must be present. A business collaboration protocol is a long running transaction that does not meet the atomic principle of transactions. It should be used in cases where transaction rollback is inappropriate.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>No Tagged Values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessTransactionActivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>ActionState</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>A business transaction activity is an activity within a business collaboration protocol. It is an action state which is refined by a nested business transaction. The business transaction activity executes the nested business transaction. The business transaction activity can be executed more than once if the “isConcurrent” property is true.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>timeToPerform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>TimeExpression</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>A business transaction activity has to be executed within a specific duration. The initiating partner must send a failure notification to a responding partner on timeout. A responding partner simple terminates its activity. The time to perform is the maximum duration between the moment the requesting authorized role initiates the business transaction activity, i.e. sending the requesting business information, and the moment the requesting authorized role receives a substantive response. The substantive response is the responding business information if there is any. In case not, it is the acknowledgement of processing, if any. If not it is the acknowledgement of receipt, if any.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>isConcurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>If the business transaction activity is concurrent then more than one business transaction activity of the same underlying business transaction can be open at one time in executing the same business collaboration with the same business partner type. If the business transaction activity is not concurrent then only one business transaction activity of the same underlying business transaction can be open at one time.</td>
</tr>
</tbody>
</table>
### 5.3.1.3 Constraints (normative)

#### A BusinessChoreographyBehavior MUST be the client of exactly one mapsTo dependency to a BusinessCollaborationUseCase

```plaintext
definition package Behavioral_Elements::Activity_Graphs context ActivityGraph

inv BCBmapsToBCUseCase:
  self.isBusinessChoreographyBehavior() implies 
  self.clientDependency->size() = 1 and 
  self.clientDependency->forAll(d | d.isMapsToDependency() and 
  d.supplier->forAll(isBusinessCollaborationUseCase()) and 
  d.supplier->size=1)
```

#### A BusinessChoreographyView package MUST contain exactly one BusinessChoreography and no other elements.

```plaintext
definition package Model_Management context Package

inv BCVcontainsExactlyOneBC:
  self.isBusinessChoreographyView() implies 
  self.contents->one(isBusinessChoreography()) and 
  self.contents->size() = 1
```

#### The behavior of a BusinessChoreography MUST be described by exactly one BusinessChoreographyBehaviour

```plaintext
definition package Foundation::Core context Class

inv BCdescribedByOneBusinessChoreographyBehavior:
  self.isBusinessChoreography() implies 
  self.behavior->one(isBusinessChoreographyBehavior()) and 
  self.behavior->size() = 1
```
A BusinessCollaborationProtocol MUST contain at least one BusinessTransactionActivity or BusinessCollaborationActivity and MAY contain PseudoStates, FinalStates and Transitions.

```plaintext
class CompositeState

inv AllowedModelElementsInBCP:
  self.stateMachine.isBusinessCollaborationProtocol() implies
  self.subvertex->forAll(isBusinessTransactionActivity() or isBusinessCollaborationActivity() or isPseudoStateOrFinalStateOrTransition() or isTransition())
  and (self.subvertex->exists(isBusinessTransactionActivity()) or self.subvertex->exists(isBusinessCollaborationActivity()))
```

A BusinessCollaborationActivity MUST be refined by exactly one BusinessCollaborationProtocol via a dependency with the stereotype mapsTo.

```plaintext
class ActionState

inv BCArefinedByExactlyOneBCP:
  self.isBusinessCollaborationActivity() implies
  self.clientDependency->size() = 1 and
  self.clientDependency->forAll(d | d.isMapsToDependency() and d.supplier->forAll(isBusinessCollaborationProtocol()) and d.supplier->size=1)
```

A BusinessTransactionActivity MUST be refined by exactly one BusinessTransaction via a dependency with the stereotype mapsTo.

```plaintext
class ActionState

inv BTAreefinedByExactlyOneBT:
  self.isBusinessTransactionActivity() implies
  self.clientDependency->size() = 1 and
  self.clientDependency->forAll(d | d.isMapsToDependency() and d.supplier->forAll(isBusinessTransaction()) and d.supplier->size=1)
```
5.3.1.4 Example (informative)

Figure 29 BusinessChoreographyView (BusinessTransactionView) Example: OrderFromQuote BusinessCollaborationProtocol (ActivityGraph)
5.3.2 Business Interaction View

5.3.2.1 Conceptual Description (informative)

A business interaction view is used to define exactly one business interaction that leads to a synchronized business state between the two authorized roles executing it. Thus, the BusinessInteractionView is composed of exactly one BusinessInteraction. A business interaction is a persistent representation of a synchronization of business states between authorized roles. The choreography of this synchronization and the required information exchanges are defined by the business interaction behavior. Each BusinessInteraction is composed of exactly one BusinessInteractionBehavior. The business interaction behavior follows exactly the requirements defined in a corresponding business transaction use case of the BRV. Each BusinessTransactionUseCase of the BRV is mapped to exactly one BusinessInteractionBehavior, and each BusinessInteractionBehavior is mapped from exactly one BusinessTransactionUseCase.

BusinessInteractionBehavior is an abstract concept. In a future version there may exist different approaches to describe the choreography and information exchanges in a business interaction. In this version, the only valid specialization of a BusinessInteractionBehavior is the BusinessTransaction. A business transaction is an atomic business process between two authorized roles, which involves sending business information from one authorized role to the other and an optional reply. The business transaction is built by two partitions - one for each authorized role. Hence, a BusinessTransaction is composed of exactly two BusinessTransactionSwimlanes. Each BusinessTransactionSwimlane relates to one AuthorizedRole. An Authorized Role is assigned to exactly one BusinessTransactionSwimlane. It follows, that the two swimlanes of a business transaction must be assigned to different authorized roles.
Within a business transaction each authorized role performs exactly one business action – the requesting authorized role performs a requesting business activity and the responding authorized role performs a responding business activity. Each business action – no matter whether requesting or responding business activity – is assigned to a swimlane, and each swimlane comprises exactly one business action. It follows that a BusinessTransaction is composed of exactly one RequestingBusinessActivity and exactly one RespondingBusinessActivity. Both RequestingBusinessActivity and RespondingBusinessActivity are specializations of BusinessAction. A BusinessAction is assigned to one BusinessTransactionSwimlane, and a BusinessTransactionSwimlane comprises one BusinessAction. Since a swimlane is dedicated to exactly one authorized role, it follows that the business action is executed by this authorized role. Furthermore an authorized role executes just one business action, because only one business action sits within a swimlane.

The requesting business activity outputs the requesting information envelope that is input to the responding business activity. Business information created by the responding business activity and returned to the requesting business activity is optional. It follows, that a BusinessTransaction is composed of exactly one RequestingInformationEnvelope and zero or one RespondingInformationEnvelope. Both RequestingInformationEnvelope and RespondingInformationEnvelope are instances of the type InformationEnvelope. A RequestingBusinessActivity outputs exactly one RequestingInformationEnvelope and a RequestingInformationEnvelope is created by exactly one RequestingBusinessActivity. A RequestingBusinessActivity receives zero or one RespondingInformationEnvelope as input and a RespondingInformationEnvelope is input to exactly one RequestingBusinessActivity.

A RespondingBusinessActivity outputs zero or one RespondingInformationEnvelope and a RespondingInformationEnvelope is created by exactly one RespondingBusinessActivity. A RespondingBusinessActivity receives exactly one RequestingInformationEnvelope as input and a RequestingInformationEnvelope is input to exactly one RespondingBusinessActivity.

Note, that a RequestingInformationEnvelope (or a RespondingInformationEnvelope) is a stereotype of the base class ObjectFlowState. The type of the ObjectFlowState is defined by the InformationEnvelope that is a stereotype of base class Class. According to UML, multiple ObjectFlowStates might be instances of the same Class. It follows that different requesting or responding information envelopes might be instances of the same information envelope. In other words, an information envelope might be reused in different business transactions.
### 5.3.2.2 Stereotypes and Tag Definitions (normative)

**Stereotype** | **BusinessInteraction**
---|---
**Base Class** | Class
**Parent** | N/A
**Description** | A business interaction is a persistent representation of a synchronization of business states between authorized roles. It is a unit of work that allows roll-back.
**Tag Definition** | No Tagged Values

**Stereotype** | **BusinessInteractionBehavior (abstract)**
---|---
**Base Class** | ActivityGraph
**Parent** | N/A
**Description** | A business interaction behavior defines the choreography of actions as well as involved business information and business signal exchanges that lead to synchronized business states between two authorized roles executing it.
**Tag Definition** | No Tagged Values
A business transaction is the basic building block to define choreography between authorized roles. If an authorized role recognizes an event that changes the state of a business object, it initiates a business transaction to synchronize with the collaborating authorized role. It follows that a business transaction is an atomic unit that leads to a synchronized state in both information systems. We distinguish one-way and two-way business transaction: In the former case, the initiating authorized role reports an already effective and irreversible state change that the reacting authorized role has to accept. Examples are the notification of shipment or the update of a product in a catalog. It is a one-way business transaction, because business information (not including business signals for acknowledgments) flows only from the initiating to the reacting authorized role. In the other case, the initiating partner sets the business object(s) into an interim state and the final state is decided by the reacting authorized role. Examples include request for registration, search for products, etc. It is a two-way transaction, because business information flows from the initiator to the responder to set the interim state and back to set the final and irreversible state change. In a business context irreversible means that returning to an original state requires another business transaction. E.g., once a purchase order is agreed upon in a business transaction a rollback is not allowed anymore, but requires the execution of a cancel order business transaction. We distinguish 2 one-way business transactions and four two-way business transactions. The type of transaction is indicated in the tagged value of business transaction type. The other tagged values provide quality of service parameters.

A business transaction follows always the same pattern: A business transaction is performed between two authorized roles that are assigned to exactly one swimlane each. Each authorized role performs exactly one activity. An object flow between the requesting and the responding business activity is mandatory. An object flow in the reverse direction is optional. According to the business transaction semantics, the requesting business activity does not end after sending the envelope - it is still alive. The responding business activity may output the response which is returned to the still living requesting business activity.

### Tag Definition

<table>
<thead>
<tr>
<th><strong>Tag</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
</table>
| `businessTransactionType` | Type: String  
| **Multiplicity**     | 1                                                                         |

**Description:**
The business transaction type determines a corresponding business transaction pattern. A business transaction pattern provides a language and grammar for constructing business transactions. The business transaction type follows one of the following six property-value conventions:

1. **Commercial Transaction** - used to model the “offer and acceptance” business transaction process that results in a residual obligation between both parties to fulfill the terms of the contract

2. **Query/Response** – used to query for information that a responding partner already has e.g. against a fixed data set that resides in a database

3. **Request/Response** - used for business contracts when an initiating partner requests information that a responding partner already has and when the request for business information requires a complex interdependent set of results

4. **Request/Confirm** - used if an initiating partner asks for information that requires only confirmation with respect to previously established contracts or with respect to a responding partner’s business rules

5. **Information Distribution** - used to model an informal information exchange business transaction that therefore has no non-repudiation requirements

6. **Notification** - used to model a formal information exchange business transaction that therefore has non-repudiation requirements

---

**UN/CEFACT – UMM Foundation Module Version 1.0 – Technical Specification**
<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Both partners must agree to exchange business information using a secure transport channel. The following security controls ensure that business document content is protected against unauthorized disclosure or modification and that business services are protected against unauthorized access. This is a point-to-point security requirement. Note that this requirement does not protect business information once it is off the network and inside an enterprise. The following are requirements for secure transport channels. Authenticate sender identity – Verify the identity of the sender (employee or organization) that is initiating the interaction (authenticate). For example, a driver’s license or passport document with a picture is used to verify an individual’s identity by comparing the individual against the picture. Authenticate receiver identity – Verify the identity of the receiver (employee or organization) that is receiving the interaction. Verify content integrity – Verify the integrity of the content exchanged during the interaction i.e. check that the content has not been altered by a 3rd party. Maintain content confidentiality – Confidentiality ensures that only the intended receiver can read the content of the interaction. Information exchanged during the interaction must be encrypted when sent and decrypted when received. For example, you seal envelopes so that only the recipient can read the content.</td>
</tr>
</tbody>
</table>

### BusinessTransactionSwimlane

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BusinessTransactionSwimlane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Partition</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>A business transaction swimlane is used to define an area of responsibility. An authorized role is appointed to the partition of a business transaction swimlane. This authorized role takes on the responsibility for the business action that is allocated within that area of responsibility.</td>
</tr>
<tr>
<td>Tag Definition</td>
<td>No Tagged Values</td>
</tr>
<tr>
<td>Stereotype</td>
<td>BusinessAction (abstract)</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Base Class</td>
<td>ActionState</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The business action is executed by an authorized role during a business transaction. Business action is an abstract stereotype. This means a business action is either a requesting business activity or a responding business activity.</td>
</tr>
</tbody>
</table>

### Tag Definition

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsAuthorizationRequired</td>
<td>Boolean</td>
<td>1</td>
<td>If an authorized role needs authorization to request a business action or to respond to a business action then the sender must sign the business document exchanged and the receiver must validate this business control and approve the authorizer. A receiver must signal an authorization exception if the sender is not authorized to perform the business activity. A sender must send notification of failed authorization if a receiver is not authorized to perform the responding business activity.</td>
</tr>
<tr>
<td>isNonRepudiationRequired</td>
<td>Boolean</td>
<td>1</td>
<td>The isNonRepudiationRequired tag is used to indicate that an involved party must not be able to repudiate the execution of the business action that input/outputs business information.</td>
</tr>
<tr>
<td>isNonRepudiationReceiptRequired</td>
<td>Boolean</td>
<td>1</td>
<td>The isNonRepudiationOfReceiptRequired tag requires the receiver of an information envelope to send a signed receipt. The isNonRepudiationOfReceiptRequired tag indicates that an involved party must not be able to repudiate the execution of sending the signed receipt.</td>
</tr>
<tr>
<td>timeToAcknowledgeReceipt</td>
<td>TimeExpression</td>
<td>1</td>
<td>Both partners may agree to mutually verify receipt of business information within a specific time duration. Acknowledgements of receipt may be sent for both the requesting business information and the responding business information. This means the sender of the business information may be the requesting authorized role as well as the responding authorized role – it depends on whether a requesting or a responding business information is acknowledged. Similarly, the affirmant may be the requesting authorized role as well as the responding authorized role – again depending of which business information is acknowledged. Inasmuch we use the terms sender and affirmant in the explanation of acknowledgement of receipt semantics. An affirmant must exit the transaction if they are not able to verify the proper receipt of a business information within the agree timeout period. A sender must retry a business transaction if necessary or must send notification of failed business control (possibly revoking a contractual offer) if an affirmant does not verify properly receipt of a business information within the agreed time period. The time to acknowledge receipt is the maximum duration from the time a business information...</td>
</tr>
</tbody>
</table>
is sent by a sender until the time a verification of receipt is “properly received” by
the sender (of the business information). This verification of receipt is an audit-able
business signal and is instrumental in contractual obligation transfer during a
contract formation process (e.g. offer/accept).

<table>
<thead>
<tr>
<th>Type</th>
<th>TimeExpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Description        | Similarly to the timeToAcknowledgeReceipt, the sender of a business information
                  | might be the requesting authorized role as well as the responding authorized role –
                  | depending whether a requesting or a responding business information is
                  | acknowledged. Also the affirmand may be one of the two authorized roles. Thus, we
                  | use again the terms sender and affirmand in the explanation of the acknowledgment
                  | of processing semantics.
                  | Both partners may agree to the need for an acknowledgment of processing to be
                  | returned by a responding partner after the requesting business information passes a
                  | set of business rules and is handed over to the application for processing. The time
to acknowledge processing of a business information is the duration from the time a
sender sends a business information until the time an acknowledgment of
processing is “properly received” by the sender (of the business information). An
affirmand must exit the transaction if they are not able to acknowledge processing of
business information within the maximum timeout period. A sender must retry a
business transaction if necessary or must send notification of failed business control
(possibly revoking a contractual offer) if an affirmand does not acknowledge
processing of business information within the agreed time period. |

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
</tbody>
</table>
| Description        | In order to define the isIntelligibleCheckRequired semantics, we use again the terms
                  | sender and affirmand as introduced for the last two tag definitions.
                  | Both partners may agree that an affirmand must check that business information is
                  | not garbled (unreadable, unintelligible) before verification of proper receipt is
                  | returned to the sender (of the business information). Verification of receipt must be
                  | returned when a document is “accessible” but it is preferable to also check for
garbled transmissions at the same time in a point-to-point synchronous business
network where partners interact without going through an asynchronous service
provider. |
A requesting business activity is a business action that is performed by an authorized role requesting business service from another authorized role.

### timeToRespond

<table>
<thead>
<tr>
<th>Type</th>
<th>TimeExpression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>Both partners may agree in case of a two-way business transaction that the responding authorized role must return the responding information business information within a specific duration. A responding authorized role must exit the transaction if they are not able to return the responding business information within the agreed timeout period. A requesting authorized role must retry a business transaction if necessary or must send notification of failed business control (possibly revoking a contractual offer) if a responding authorized role does not deliver the responding business information within the agreed time period. The time to perform is the maximum duration from the time a requesting business information is sent by a requesting authorized role until the time a responding business information is “properly received” by the requesting authorized role in return.</td>
</tr>
</tbody>
</table>

### retryCount

<table>
<thead>
<tr>
<th>Type</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicity</td>
<td>1</td>
</tr>
<tr>
<td>Description</td>
<td>The requesting authorized role must re-initiate the business transaction so many times as specified by the retry count in case that a time-out-exception – by exceeding the time to acknowledge receipt, or the time to acknowledge processing, or the time to respond – is signaled. This parameter only applies to time-out signals and not document content exceptions or sequence validation exceptions.</td>
</tr>
</tbody>
</table>

Inherited tagged values:
- isAuthorizationRequired
- isNonRepudiationRequired
- isNonRepudiationReceiptRequired
- timeToAcknowledgeReceipt
- timeToAcknowledgeAcceptance
- isIntelligibleCheckRequired
### Stereotype: RespondingBusinessActivity

<table>
<thead>
<tr>
<th><strong>Base Class</strong></th>
<th>ActionState</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent</strong></td>
<td>Business Action</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>A responding business activity is a business action that is performed by an authorized role responding to another authorized role’s request for business service.</td>
</tr>
</tbody>
</table>

**Tag Definition**

*Inherited tagged values:*

- isAuthorizationRequired
- isNonRepudiationRequired
- isNonRepudiationReceiptRequired
- timeToAcknowledgeReceipt
- timeToAcknowledgeAcceptance
- isIntelligibleCheckRequired

### Stereotype: RequestingInformationEnvelope

<table>
<thead>
<tr>
<th><strong>Base Class</strong></th>
<th>ObjectFlowState</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The requesting information envelope is a container of business information that is sent from the requesting authorized role to the responding authorized role to indicate a state change in one or more business entities. This business state change might be irreversible in the case of a one-way business transaction or an interim state of a two-way business transaction. It is important to note that the term requesting information envelope does not mean that the business information refers to a request in a business sense. The term requesting information envelope indicates that the execution of a transaction is requested from the requesting authorized role to the responding authorized role – no matter whether this is an information distribution, a notification, a request, or the offer in a commercial transaction.</td>
</tr>
</tbody>
</table>

**Tag Definition**

*No Tagged Values*

### Stereotype: RespondingInformationEnvelope

<table>
<thead>
<tr>
<th><strong>Base Class</strong></th>
<th>ObjectFlowState</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The responding information envelope is a container of business information that is sent in case of a two-way business transaction from the responding authorized role to the requesting authorized role in order to set one or more business entities in a final state (which were in an interim state before).</td>
</tr>
</tbody>
</table>

**Tag Definition**

*No Tagged Values*

### 5.3.2.3 Constraints (normative)

A `BusinessInteractionView` package MUST contain exactly one `BusinessInteraction` and no other elements

```plaintext
package Model_Management
context Package

inv BIVcontainsExactlyOneBI:
    self.isBusinessInteractionView() implies
    self.contents->one(isBusinessInteraction())
and self.contents->size()=1
```
A BusinessInteractionBehavior MUST be connected with exactly one BusinessTransactionUseCase via a dependency with the stereotype mapsTo.

```plaintext
class ActivityGraph

context ActivityGraph

inv BIBmapsToExactlyOneBusinessTransactionUseCase:
  self.isBusinessInteractionBehavior() implies
  self.clientDependency->size() = 1 and
  self.clientDependency->forAll(d | d.isMapsToDependency() and
  d.supplier->forAll(isBusinessTransactionUseCase()) and
  d.supplier->size=1)
```

The behaviour of a BusinessInteraction must be described by exactly one BusinessInteractionBehavior.

```plaintext
class Core

context Class

inv BehaviorOfBIdescribedByExactlyOneBusinessInteractionBehavior:
  self.isBusinessInteraction() implies
  self.behavior->one(isBusinessInteractionBehavior()) and
  self.behavior->size()=1
```

A BusinessTransaction MUST have exactly two partitions, which MUST be stereotyped as BusinessTransactionSwimlanes. One partition MUST contain the RequestingBusinessActivity and one MUST contain the RespondingBusinessActivity.

```plaintext
class ActivityGraph

context ActivityGraph

inv BusinessTransactionHasExactlyTwoBTSwimlanes:
  self.isBusinessTransaction() implies
  self.oclAsType(ActivityGraph).partition->size() = 2 and
  self.oclAsType(ActivityGraph).partition->forAll(part | part.isUMMTransactionSwimlane() and
  (part.contents->one(isRequestingBusinessActivity()) xor part.contents
  ->one(isRespondingBusinessActivity()))) and
  self.oclAsType(ActivityGraph).partition->collect(part | part.contents)->one(isRequestingBusinessActivity()) and
  self.oclAsType(ActivityGraph).partition->collect(part | part.contents)->one(isRespondingBusinessActivity())
```
A BusinessTransactionSwimlane MUST have a classifier, which MUST be one of the associated AuthorizedRoles of the corresponding BusinessTransactionUseCase.

package Behavioral_Elements::Activity_Graphs
context Partition

inv BusinessTransactionSwimlaneClassifier:
  self.isUMMTransactionSwimlane() implies
  self.classifierRole.base->size()=1 and
  self.activityGraph.clientDependency->
  collect(s | s.supplier)->collect(a | a.oclAsType(UseCase).associations)->
  collect(allConnections)
  ->select(isAuthorizedRole())->one(x | x = (self.classifierRole.base->
  asSequence->first()))

The partition of the requesting authorized role must contain exactly one RequestingBusinessActivity, one RequestingInformationEnvelope and one InitialState. Furthermore there MUST be at least two FinalStates in this BusinessTransactionSwimlane.

package Behavioral_Elements::Activity_Graphs
context Partition

inv ContentsOfRequestingPartition:
  self.isUMMTransactionSwimlane() implies
  self.contents->one(isRequestingBusinessActivity()) implies
  self.contents->forall(isRequestingBusinessActivity() or
  isRequestingInformationEnvelope() or
  isInitialState() or
  isFinalState() or
  isTransition())
  and
  self.contents->one(isRequestingInformationEnvelope()) and
  self.contents->select(isFinalState())->size()>1 and
  self.contents->one(isInitialState())
The partition of the responding authorized role MUST exactly contain one RespondingBusinessActivity. Furthermore if the transaction is a two way business transaction, then the partition must contain a RespondingInformationEnvelope as well. If the transaction is a one way business transaction, then the responder partition must not contain a RespondingInformationEnvelope.

```plaintext
package Behavioral_Elements::Activity_Graphs
context Partition

inv ContentsOfResponderPartition :
  self.isUMMTransactionSwimlane() implies
  self.contents->one(isRespondingBusinessActivity()) implies
  self.contents->forAll(isRespondingBusinessActivity() or isRespondingInformationEnvelope() or isTransition())
  and if
  self.activityGraph.isTwoWayTransaction()
  then
    self.contents->one(isRespondingInformationEnvelope())
  else
    not self.contents->exists(isRespondingInformationEnvelope())
  endif

Exactly one Transition MUST lead from the InitialState to the RequestingBusinessActivity

```plaintext
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrInitialState2RequestingBusinessActivity:
  self.isUMMTransactionSwimlane() implies
  self.contents->one(isRequestingBusinessActivity()) implies
  self.contents->select(isInitialState())->forAll(oclAsType(Pseudostate).outgoing->size()=1 and oclAsType(Pseudostate).outgoing->asSequence()->first().target.isRequestingBusinessActivity())

Exactly one Transition MUST lead from a RequestingBusinessActivity to the RequestingInformationEnvelope

```plaintext
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrRequestingBusinessActivity2RequInfEnvelope:
  self.isUMMTransactionSwimlane() implies
  self.contents->one(isRequestingBusinessActivity()) implies
  self.contents->select(isRequestingBusinessActivity())->forAll(oclAsType(ActionState).outgoing->size()=1 and oclAsType(ActionState).outgoing->asSequence()->first().target.isRequestingInformationEnvelope())
```
Exactly one Transition MUST lead from the RequestingInformationEnvelope to the RespondingBusinessActivity

```plaintext
class Behavioral_Elements::Activity_Graphs
context Partition

inv TrRequestingInformationEnvelope2RespondingBusinessActivity:
  self.isUmmTransactionSwimlane() implies
  self.contents->one(isRequestingBusinessActivity()) implies
  self.contents->select(isRequestingInformationEnvelope())-
  forAll(oclAsType(ObjectFlowState).outgoing->size()=1 and
  oclAsType(ObjectFlowState).outgoing->asSequence
  ->first().target.isRespondingBusinessActivity())
```

Exactly one Transition MUST lead from the RespondingBusinessActivity to the RespondingInformationEnvelope (only two way business transactions)

```plaintext
class Behavioral_Elements::Activity_Graphs
context Partition

inv TrRespondingBusinessActivity2RespondingInformationEnvelope:
  self.activityGraph.isTwoWayTransaction() implies
  self.contents->one(isRespondingBusinessActivity()) implies
  self.contents->select(isRespondingBusinessActivity())-
  forAll(oclAsType(ActionState).outgoing->size()=1 and
  oclAsType(ActionState).outgoing->asSequence
  ->first().target.isRespondingInformationEnvelope())
```

Exactly one Transition MUST lead from the RespondingInformationEnvelope to the RequestingBusinessActivity (only two way business transactions)

```plaintext
class Behavioral_Elements::Activity_Graphs
context Partition

inv TrRespondingInformationEnvelope2RequestingBusinessActivity:
  self.activityGraph.isTwoWayTransaction() implies
  self.contents->one(isRespondingBusinessActivity()) implies
  self.contents->select(isRespondingInformationEnvelope())-
  forAll(oclAsType(ObjectFlowState).outgoing->size()=1 and
  oclAsType(ObjectFlowState).outgoing->asSequence
  ->first().target.isRequestingBusinessActivity())
```
There MAY be a Transition from RespondingBusinessActivity to RequestingBusinessActivity (only for one way business transactions)

```plaintext
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrPossibleRespondingInformationEnvelope2RequestingBusinessActivity:
  self.activityGraph.isOneWayTransaction() implies
  self.contents->one(isRespondingBusinessActivity()) implies
  self.contents->select(isRespondingBusinessActivity())->
  forAll(oclAsType(ActionState).outgoing->size()==1 and
          oclAsType(ActionState).outgoing->asSequence->first().target.isRequestingBusinessActivity() or
          oclAsType(ActionState).outgoing->isEmpty())
```

One Transition MUST lead from the RequestingBusinessActivity to each FinalState.

```plaintext
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrRequestingBusinessActivity2FinalState:
  self.isUMMTransactionSwimlane() implies
  self.contents->one(isRequestingBusinessActivity()) implies
  self.contents->select(isRequestingBusinessActivity())->
  forAll(oclAsType(ActionState).outgoing->size()==1 and
          oclAsType(ActionState).outgoing->asSequence->first().target.isFinalState())
```

Each RequestingInformationEnvelope and each RespondingInformationEnvelope MUST have a classifier, which MUST itself be a class and stereotyped as InformationEnvelope

```plaintext
package Behavioral_Elements::Activity_Graphs
context ObjectFlowState

inv ObjectFlowStateHasClassifier:
  (self.isRequestingInformationEnvelope() or
   self.isRespondingInformationEnvelope()) implies
  self.type.oclAsType(ClassifierInState).type.isInformationEnvelope()
```
5.3.2.4 Example (informative)

Figure 32 BusinessInteractionView (BusinessTransactionView) Example: PlaceOrder BusinessTransaction (ActivityGraph)
5.3.3 Business Information View

5.3.3.1 Conceptual Description (informative)

A business information view is a container of artifacts that describe the information exchanged in an interaction. We already mentioned before that RequestingInformationEnvelope and RespondingInformationEnvelope are of type InformationEnvelope. An information envelope serves as a cover for all the information exchanged between the requesting business activity and the responding business activity or vice versa, respectively. The information included in the envelope is structured by classes that are stereotyped as InformationEntity. Information entities might be recursively nested. Thus there is a unary composition hierarchy added to InformationEntity. An information envelope is built by zero or one header and one or more bodies. Both header and body are presented as information entities. It follows, that an InformationEnvelope is composed of exactly zero or one InformationEntity with the rolename header and of one or more InformationEntities with the rolename body. An InformationEnvelope is a specialization of an InformationEntity that fulfills all the rules mentioned for the information envelope as well.

The current UMM foundation module does not define any rules on how to build information entities. All methodologies and rules to build good quality class diagrams do also apply to model an information envelope and its contents. Modelers who want to use UN/CEFACT’s Core Components might do so as well - it is only important that all resulting classes no matter what type of Core Component are stereotyped as InformationEntity. However, there is a specialization module – the Core Component UML Profile – on the way in order to better support the modeling of business information by Core Components.

Figure 33 BusinessInformationView (BusinessTransactionView) Conceptual Overview
5.3.3.2 Stereotypes and Tag Definitions (normative)

![cd BusinessInformation - Abstract Syntax](image)

**Figure 34 BusinessInformationView (BusinessTransactionView) Abstract Syntax**

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>InformationEntity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base Class</strong></td>
<td>Class</td>
</tr>
<tr>
<td><strong>Parent</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>An information entity realizes structured business information that is exchanged between authorized roles performing activities in a business transaction. Information entities include or reference other information entities through associations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>isConfidential</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If the flag is set, the information entity is encrypted so that unauthorized parties cannot view the information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>isTamperProof</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If the flag is set, the information entity has an encrypted message digest that can be used to check if the message has been tampered with. This requires a digital signature (sender’s digital certificate and encrypted message digest) associated with the document entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definition</th>
<th>isAuthenticated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Boolean</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>If the flag is set, there is a digital certificate associated with the document entity. This provides proof of the signer’s identity.</td>
</tr>
</tbody>
</table>
### Stereotype InformationEnvelope

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>InformationEnvelope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class</td>
</tr>
<tr>
<td>Parent</td>
<td>InformationEntity</td>
</tr>
</tbody>
</table>

### Description

An information envelope is a container for information entities. The information envelope is a specialization of the information entity. It extends the concept of the information entity by the fact that it includes exactly one information entity that takes on the role of a header and at least one information entity that takes on the role of a body. Furthermore, the information exchanged in a business transaction, i.e., a requesting business information and a responding business information, is always of type information envelope.

### Tag Definition

#### Inherited tagged values:
- isConfidential
- isTamperProof
- isAuthenticated

### 5.3.3.3 Constraints (normative)

A `BusinessInformationView` package must contain only `InformationEntities` and `InformationEnvelopes` and no other elements.

```plaintext
package Foundation::Core
class Class

inv AllowedElementsInBusinessInformationView:
    self.isBusinessInformationView() implies
    self.content->forall(a | a.isInformationEntity() or a.isInformationEnvelope())
```

An `InformationEnvelope` **MUST** have zero or one association to an `InformationEntity` with role name header.

```plaintext
package Foundation::Core
class Class

inv InformationEnvelopeHasHeader:
    self.isInformationEnvelope() implies
    self.associations->size() < 1 and
    self.associations->forall(a | a.connection->size() = 2 and a.allConnections->one(participant.isInformationEntity() and AssociationEndRole.name = 'header'))
```
An **InformationEnvelope** MUST have at least one associated **InformationEntity** with role name body

```
package Foundation::Core
context Class

inv InformationEnvelopeHasBodies:
  self.isInformationEnvelope() implies
  self.associations->forAll(a | a.connection->size() = 2 and
  a.allConnections->exists(participant.isInformationEntity() and
  AssociationEndRole.name = 'body'))
```

An **InformationEntity** MAY be composed of other **InformationEntities**

```
package Foundation::Core
context Class

inv contentsOfInformationEntitiy:
  self.isInformationEntity() implies
  self.associations->
  forAll(a | a.allConnections->exists(isAggregate()) and
  a.allConnections->exists(participant.isInformationEntity()))
```

### 5.3.3.4 Example (informative)

![Figure 35 BusinessInformationView (BusinessTransactionView) Example: OrderEnvelope (ClassDiagram) - conceptual](image-url)
5.3.4 OCL methods used in all packages of the BTV (normative)

<table>
<thead>
<tr>
<th>OCL-Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>package Foundation::Core</td>
</tr>
<tr>
<td>context ModelElement</td>
</tr>
<tr>
<td>--Predefined method whichs evaluates, if the given ModelElement</td>
</tr>
<tr>
<td>--has a stereotype equal to the passed name</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let hasStereotype (st : String) : Boolean =</td>
</tr>
<tr>
<td>self.stereotype-&gt;select(self.name = st)-&gt;notEmpty()</td>
</tr>
<tr>
<td>--Predefined method whichs evaluates, if the given element</td>
</tr>
<tr>
<td>--has the stereotype 'BusinessTransaction'</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let isBusinessTransaction() : Boolean =</td>
</tr>
<tr>
<td>self.oclIsKindOf(ActivityGraph) and</td>
</tr>
<tr>
<td>self.hasStereotype('BusinessTransaction')</td>
</tr>
<tr>
<td>--Predefined method whichs evaluates, if the given element</td>
</tr>
<tr>
<td>--has the stereotype 'BusinessInteraction'</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let isBusinessInteraction() : Boolean =</td>
</tr>
<tr>
<td>self.oclIsKindOf(Class) and</td>
</tr>
<tr>
<td>self.hasStereotype('BusinessInteraction')</td>
</tr>
<tr>
<td>--Predefined method whichs evaluates, if the given element</td>
</tr>
<tr>
<td>--is a subtype of 'BusinessInteractionBehavior'</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let isBusinessInteractionBehavior() : Boolean =</td>
</tr>
<tr>
<td>self.oclIsKindOf(ActivityGraph) and</td>
</tr>
<tr>
<td>self.hasStereotype('BusinessInteraction')</td>
</tr>
<tr>
<td>--Predefined method whichs evaluates, if the given element</td>
</tr>
<tr>
<td>--is a 'BusinessChoreography'</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let isBusinessChoreography() : Boolean =</td>
</tr>
<tr>
<td>self.oclIsKindOf(Class) and</td>
</tr>
<tr>
<td>self.hasStereotype('BusinessChoreography')</td>
</tr>
<tr>
<td>--Predefined method which evaluates, if the</td>
</tr>
<tr>
<td>--ActivityGraph is a BusinessCollaborationProtocol</td>
</tr>
<tr>
<td>def :</td>
</tr>
<tr>
<td>let isBusinessCollaborationProtocol() : Boolean =</td>
</tr>
<tr>
<td>self.oclIsKindOf(ActivityGraph) and</td>
</tr>
<tr>
<td>self.hasStereotype('BusinessCollaborationProtocol')</td>
</tr>
<tr>
<td>--Predefined method which evaluates, if the</td>
</tr>
<tr>
<td>--ActivityGraph is a subtype of</td>
</tr>
<tr>
<td>--BusinessChoreographyBehavior</td>
</tr>
<tr>
<td>def :</td>
</tr>
</tbody>
</table>
let isBusinessChoreographyBehavior() : Boolean =
  self.oclIsKindOf(ActivityGraph) and
  self.hasStereotype('BusinessCollaborationProtocol')

-- Predefined method which evaluates, if the given element
-- has the stereotype 'RequestingBusinessActivity' and
-- if its type is ActionState
def:
let isRequestingBusinessActivity() : Boolean =
  self.oclIsKindOf(ActionState) and
  self.hasStereotype('RequestingBusinessActivity')

-- Predefined method which evaluates, if the given element
-- has the stereotype 'RespondingBusinessActivity' and
-- if its type is ActionState
def:
let isRespondingBusinessActivity() : Boolean =
  self.oclIsKindOf(ActionState) and
  self.hasStereotype('RespondingBusinessActivity')

-- Returns true if the element is located in a partition and
-- its stereotype is 'BusinessTransactionSwimlane'
def:
let isBusinessTransactionSwimlane() : Boolean =
  self.hasStereotype('BusinessTransactionSwimlane')
  and self.oclIsKindOf(Partition)

-- Returns true if the type of the element
-- is 'PseudoKindState' and its Pseudostatekind is pk_initial
def:
let isInitialState() : Boolean =
  self.oclIsKindOf(Pseudostate) and
  self.oclAsType(Pseudostate).kind = PseudostateKind::initial

-- Returns true if the type of the element is 'FinalState'
def:
let isFinalState() : Boolean =
  self.oclIsKindOf(FinalState)

-- Returns true if the type of the element
-- is 'PseudoKindState' and its Pseudostatekind is pk_choice
def:
let isChoice() : Boolean =
  self.oclIsKindOf(Pseudostate) and
  self.oclAsType(Pseudostate).kind = PseudostateKind::choice

-- Returns true if the type of the element
-- is 'PseudoState' and its Pseudostatekind
-- is pk_fork
def:
let isFork() : Boolean =
  self.oclIsKindOf(Pseudostate) and
  self.oclAsType(Pseudostate).kind = PseudostateKind::fork
--- Returns true if the type of the element is 'PseudokindState' and its Pseudostatekind is pk_choice
def:
let isJoin() : Boolean = self.oclIsKindOf(Pseudostate) and self.oclAsType(Pseudostate).kind = PseudostateKind::join

--- Returns true if the given element has a tagged value named 'tag' with a value 'value'
def:
let hasTaggedValue (tag : String, value : String) : Boolean = self.taggedValue->select(name = tag)->select(dataValue = value)->notEmpty()

--- Returns true if the element has a tagged value named 'BusinessTransaction' with a value 'NotificationActivity' or 'InformationDistributionActivity'
def:
let isOneWayTransaction() : Boolean = self.hasTaggedValue('BusinessTransactionType','NotificationActivity') or self.hasTaggedValue('BusinessTransactionType','InformationDistributionActivity')

--- Returns true if the element has a tagged value name 'BusinessTransaction' with a value 'QueryResponseActivity' or 'RequestResponseActivity' or 'CommercialTransactionActivity' or 'RequestConfirmActivity'
def:
let isTwoWayTransaction() : Boolean = self.hasTaggedValue('BusinessTransactionType','QueryResponseActivity') or self.hasTaggedValue('BusinessTransactionType','RequestResponseActivity') or self.hasTaggedValue('BusinessTransactionType','CommercialTransactionActivity') or self.hasTaggedValue('BusinessTransactionType','RequestConfirmActivity')

--- Returns true if the stereotype of the given element is 'BusinessCollaborationActivity' and if the type of the element is ActionState
def:
let isBusinessCollaborationActivity() : Boolean = self.hasStereotype('BusinessCollaborationActivity') and self.oclIsKindOf(ActionState)

--- Returns true if the stereotype of the given element is 'BusinessTransactionActivity' and if the type of the element is ActionState
def:
let isBusinessTransactionActivity() : Boolean = self.hasStereotype('BusinessTransactionActivity') and self.oclIsKindOf(ActionState)
-- Returns true if the type of the element is Transition
def:
let isTransition() : Boolean =
  self.oclIsKindOf(Transition)

-- Returns true if the given element is an element of an Activity Graph
-- (InitialState, Choice, Fork, Join, Transition or FinalState)
def:
let isPseudoStateOrFinalStateOrTransition() : Boolean =
  isInitialState() or
  isChoice() or
  isFork() or
  isJoin() or
  isFinalState()

-- Returns true if a package is stereotyped as BusinessTransactionView
def:
let isBusinessTransactionView() : Boolean =
  self.hasStereotype('BusinessTransactionView') and
  oclIsKindOf(Package)

-- Returns true if a package is stereotyped as BusinessChoreographyView
def:
let isBusinessChoreographyView() : Boolean =
  self.hasStereotype('BusinessChoreographyView') and
  oclIsKindOf(Package)

-- Returns true if the stereotype of the given element is
-- 'BusinessInformationView'
-- and if the type of the element is Package
def:
let isBusinessInformationView() : Boolean =
  self.hasStereotype('BusinessInformationView') and
  self.oclIsKindOf(Package)

-- Returns true if the stereotype of the given element is
-- 'BusinessInteractionView'
-- and if the type of the element is Package
def:
let isBusinessInteractionView() : Boolean =
  self.hasStereotype('BusinessInteractionView') and
  self.oclIsKindOf(Package)

-- Returns true if the stereotype of the given element is 'InformationEntity'
-- and if the type of the element is Class
def:
let isInformationEntity() : Boolean =
  self.hasStereotype('InformationEntity') and
  self.oclIsKindOf(Class)
-- Returns true if the association type of an association end is composite
def:
let isComposition() : Boolean =
    self.oclIsKindOf(AssociationEnd) and
    self.oclAsType(AssociationEnd).aggregation = AggregationKind::composite

-- Returns true if the association type of an association end is aggregation
def:
let isAggregate() : Boolean =
    self.oclIsKindOf(AssociationEnd) and
    self.oclAsType(AssociationEnd).aggregation = AggregationKind::aggregate

-- Returns true if the element is a partition
--and stereotyped as BusinessTransactionSwimlane
def:
let isUMMTransactionSwimlane() : Boolean =
    self.oclIsKindOf(Partition) and
    self.hasStereotype('BusinessTransactionSwimlane')

--Returns true if the stereotype of the element is
--'InformationEnvelope' and its type is Class
def:
let isInformationEnvelope() : Boolean =
    self.hasStereotype('InformationEnvelope') and
    oclIsKindOf(Class)

--Returns true if the stereotype of the element
-- is 'RequestingInformationEnvelope'
def:
let isRequestingInformationEnvelope() : Boolean =
    self.hasStereotype('RequestingInformationEnvelope') and
    oclIsKindOf(ObjectFlowState)

--Returns true if the stereotype of the element
-- is 'RespondingInformationEnvelope'
def:
let isRespondingInformationEnvelope() : Boolean =
    self.hasStereotype('RespondingInformationEnvelope') and
    oclIsKindOf(ObjectFlowState)

--Predefined method which evaluates, if the given element
--has the stereotype 'mapsTo'
def:
let isMapsToDependency() : Boolean =
    self.oclIsKindOf(Dependency) and
    self.hasStereotype('mapsTo')

--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessCollaborationUseCase'
def:
let isBusinessCollaborationUseCase() : Boolean =
    self.oclIsKindOf(UseCase) and
    self.hasStereotype('BusinessCollaborationUseCase')
-- Predefined method which evaluates, if the given element
-- has the stereotype 'BusinessTransactionUseCase'
def :
let isBusinessTransactionUseCase() : Boolean =
  self.oclIsKindOf(UseCase) and
  self.hasStereotype('BusinessTransactionUseCase')

-- Predefined method which evaluates, if the given element
-- has the stereotype 'AuthorizedRole'
def :
let isAuthorizedRole() : Boolean =
  self.oclIsKindOf(Actor) and
  self.hasStereotype('AuthorizedRole')
Copyright Statement

Copyright © UN/CEFACT 2006. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to UN/CEFACT except as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by UN/CEFACT or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and UN/CEFACT DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.