UN/CEFACT's Modeling Methodology (UMM)
UMM Meta Model – Foundation Module
Candidate for 2.0
Draft for IMPLEMENTATION VERIFICATION
2009-01-30

Table of Contents

1 About this Document

1.1 Status of this Document

This document is currently being developed, and is at Open Development Process Step 6: "Implementation Verification"

(i) Activities

The UNECE secretariat provides links on the UNECE website to the implementation. The FMG notifies Heads of Delegation and various e-mail distribution list subscribers that the Implementation Draft is available for implementation verification and provides them with details regarding the process for submitting comments. The project team processes comments and posts updated Implementation Drafts and comment logs to the PG website or UNECE website (through the secretariat). The comment/update/posting cycle continues until at least two independent implementations have been confirmed and the PG approves a project team recommendation to conclude ODP6. While the criteria, evaluation, and ultimate decision to conclude ODP6 is left to the PG, the PG must ensure that the project team has met all comment processing requirements

(ii) Artifacts

Typical artifacts produced by ODP6 include:

a. Final Draft
b. Two implementation verifications
c. Updated Comment Log

The first review cycle of ODP 6 is open until 17h April 2009. Please send your comments to the Project Lead Christian Huemer (huemer@big.tuwien.ac.at)

1.2 Revision History

Version	Release	Date	Comment
Candidate for 2.0	Internal Draft	2008-04-11	
Candidate for 2.0	Public Draft	2008-06-27	Changes are marked in green
Candidate for 2.0	Implementation Verification	2009-01-30	Changes from both previous drafts are marked in green

1.3 Document Context

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

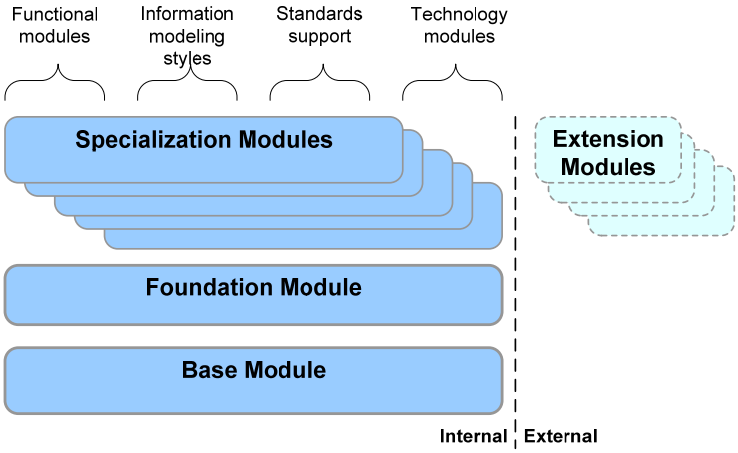


Figure 1 Module structure of the UMM meta model

Base: Covers the fundamental principles that are shared across all the other modules.

Foundation: Includes the core concepts of the UMM. In addition, it defines all the concepts that are used as part of the minimal methodology to produce a UMM compliant business collaboration model. Furthermore, it provides fundamental principles which are shared across all other modules.

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 specific 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 organization(s) which are external to UN/CEFACT.

This specification defines the foundation module of UMM 2.0.

1.4 Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119] as quoted here:

- MUST: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation that does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).

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 Vienna
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	Germany
	Birgit Hofreiter	Austria
	Christian Huemer	Austria
	Philipp Liegl	Austria
	Glenn Miller	Canada
	Harry Moyer	Australia
	Rainer Schuster	Austria
	Marco Zapletal	Austria
Contributors:	Steve Capell	Australia
	Sylvie Colas	France
	Barbara Flügge	Switzerland
	William McCarthy	USA
	Thomas Motal	Austria

Dennis Müller	Germany
Christian Senf	Germany
Nita Sharma	USA
Gunther Stuhec	Germany

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.

3 Introduction

3.1 Audience

A reader of the document MUST have a deep understanding of UML 2.1.2. She or he MUST be able to understand meta models denoted as UML class diagrams. She or he SHOULD be familiar with the UML 2.1.2. meta model, at least she or he MUST be able to check back the UML 2.1.2. 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 within an 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 (TRADE/R.650/Rev.4/Add.1/Rev.1 - 19 April 2007) http://www.unece.org/cefact/cf_plenary/plenary07/trd_R650_Rev4_A1E.pdf
  - UPCC: UML Profile for Core Components http://unstandards.org:8080/display/public/UPCC+-+UML+Profile+for+Core+Components
  - Core Component Technical Specification http://www.unece.org/cefact/ebxml/CCTS_V2-01_Final.pdf
- **International Organization for Standardization (ISO)**
  - Open-edi Reference Model. ISO/IEC 14662 http://standards.iso.org/ittf/PubliclyAvailableStandards/c037354_ISO_IEC_14662_2004(E).zip
- **Object Management Group (OMG)**
  - Unified Modeling Language Specification (UML), Version2.1.2 http://www.omg.org/docs/formal/07-02-05.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 need 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 processes and their information exchanges. At the FSV layer, this choreography must be implemented by a set of composite services. Therefore it follows, that UMM, which targets the BOV layer, defines what the business is about; and the 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 Requirements View (bRequirementsV)
- Business Choreography View (bChoreographyV)
- Business Interaction View (bInteractionV)

Business Requirements View (bRequirementsV): The Business Requirements View 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 requirements 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. In order to model the dynamics of each business process, one may use a Business Process Activity Model, or a Sequence Diagram, which would be placed beneath the Business Process Use Case. As a practical note, the Business Process Activity Model may depict a process or processes which involve one or more Business Partners. A Sequence Diagram will depict information exchanges between two or more Business Partners. The Business Partners are described within their own package (Business Partner View). A Business Process Activity Model may show state changes to Business Entities. Business Entities are "real-word" things having business significance and are shared among the business partners involved in the collaboration. The Business

Entities and their lifecycles of state changes are modeled in the Business Entity View. Furthermore, the Business Entity View also contains one or more packages which represent the conceptual data structures of the Business Entities.

Business Choreography View (bChoreographyV): The Business Choreography View is used to define and document the global choreography between collaborating business partners in an inter-organizational business process. Within the Business Choreography View, the Business Transaction View contains and documents the requirements of Business Transaction Use Cases, and their participating Authorized Roles. The dynamics of a Business Transaction Use Case are described by a Business Transaction. 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 requirements of any solution that must be implemented on each business partner's side in a service-oriented collaboration architecture. Within the Business Choreography View, the Business Collaboration View contains and documents the requirements of Business Collaboration Use Cases and their participating Authorized Roles. The dynamics of a Business Collaboration Use Case are described by a Business Collaboration Protocol. A Business Collaboration Protocol choreographs the flow among business transactions, and/or nested Business Collaboration Protocols. This flow depends on the states of business entities. When a Business Collaboration Use Case is identified, but different sets of parties may execute this collaboration, the different Realizations (executions) may be modeled within the Business Realization View, as a Business Realization Use Cases.

Business Information View (bInformationV): 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 Information View contains Business Information Artifacts. UMM does not mandate a specific Business Information Modeling approach. However, UMM strongly recommends that Business Information is modeled in accordance to UN/CEFACT's Core Components Technical Specification and Message Assembly Guidelines. In order to model Core Components by means of UML, UN/CEFACT provides the Profile for Core Components (UPCC).

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, which describe a public choreography of an inter-organizational system. Local choreographies and private processes of a businesses partner are out of scope.
- 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 an 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 UML 2.1.2. 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. 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 other future technologies. 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 is 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 2.1.2. This version is the current OMG version. Using another UML meta-model as a basis for the development of a UMM compliant business collaboration model may not deliver correct results.
- The basic concepts of the UMM and the way they relate to each other are described and explained by means of a meta model (to be found in the non-normative “conceptual description” sections of this document).
- 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 models.

3.5 Structure of the UMM Foundation Module

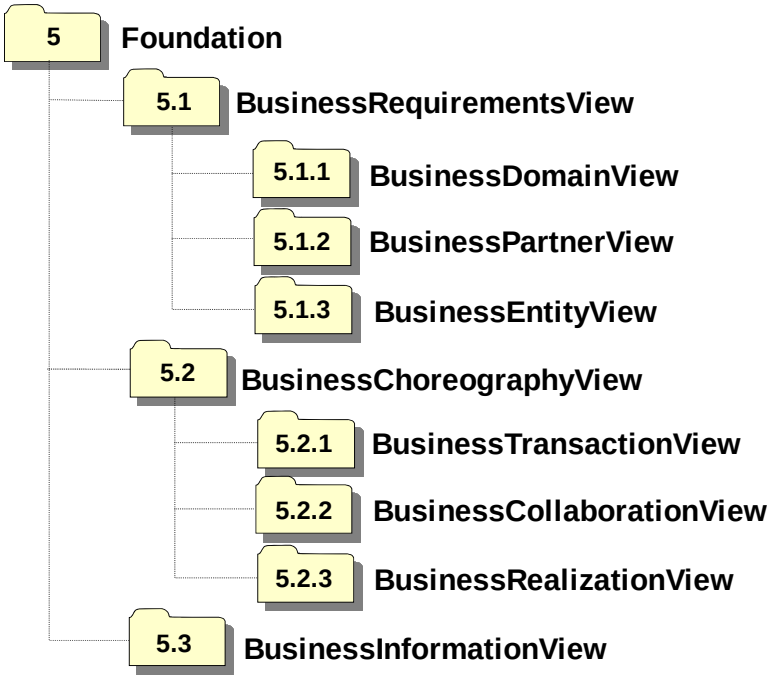


Figure 2 Package overview of UMM Foundation Module meta model

Section 5 defines the UML profile of the foundation module of the UMM meta model (Section 1, Figure 1). The figure above (Figure 2), shows the package structure of the foundation module of the UMM meta model. The numbers referring to the subsections are included in figure 2. Those numbers refer to the subsections containing 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 Requirements View (5.1), Business Choreography View (5.2), and Business Information View (5.3).

The Business Requirements View (5.1) comprises the Business Domain View (5.1.1), the Business Partner View (5.1.2), and the Business Entity View (5.1.3). The second top-level package, the Business Choreography View (5.2) consists of the Business Transaction View (5.2.1), the Business Collaboration View (5.2.2), and the Collaboration Realization View (5.2.3). The third top-level package is the Business Information View (5.3). It does not contain any sub packages.

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 in plain text that apply to the respective package. The constraints are also expressed using the Object Constraint Language (OCL) in section 6. The two remaining informative subsections cover on the one hand side worksheets used to gather information from business people in order to create the UMM models and on the other hand side examples to depict instances of the artefact type addressed by the package.

The

4 Dependencies on other UMM Modules (normative)

4.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bInformation	BusinessInformation
bLibrary	BusinessLibrary
InfEnvelope	InformationEnvelope

4.2 Dependency between Base Module and Foundation Module

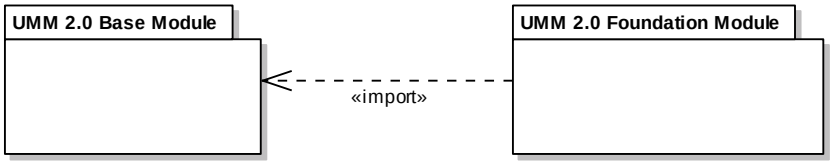


Figure 3 UMM Foundation Dependencies

The UMM foundation module 2.0 is built on top of the UMM base module 2.0. This means that all stereotypes and tag definitions defined in the UMM base module 2.0 are imported into the UMM foundation module 2.0. The figure below shows the stereotypes defined in the UMM base module also used in the foundation module. Note that the stereotypes of the base module are identified with notes in all figures of this specification. The formal definition of the stereotypes *bInformation, InfEnvelope* and *bLibrary* is given in the UMM base module 2.0 specification. In the foundation module, packages - that are containers of stereotypes realizing main UMM artefacts - are defined as specializations of the base stereotype *bLibrary*. This means that such packages and their contents are candidates for registration in a registry. In the UMM foundation module 2.0 we do not define any stereotype that directly inherits from *bLibrary*. As a consequence, only packages are candidates for registration.

The concepts of *bInformation* and *InfEnvelope* are used to define the business document information being exchanged between authorized roles in a UMM business transaction.

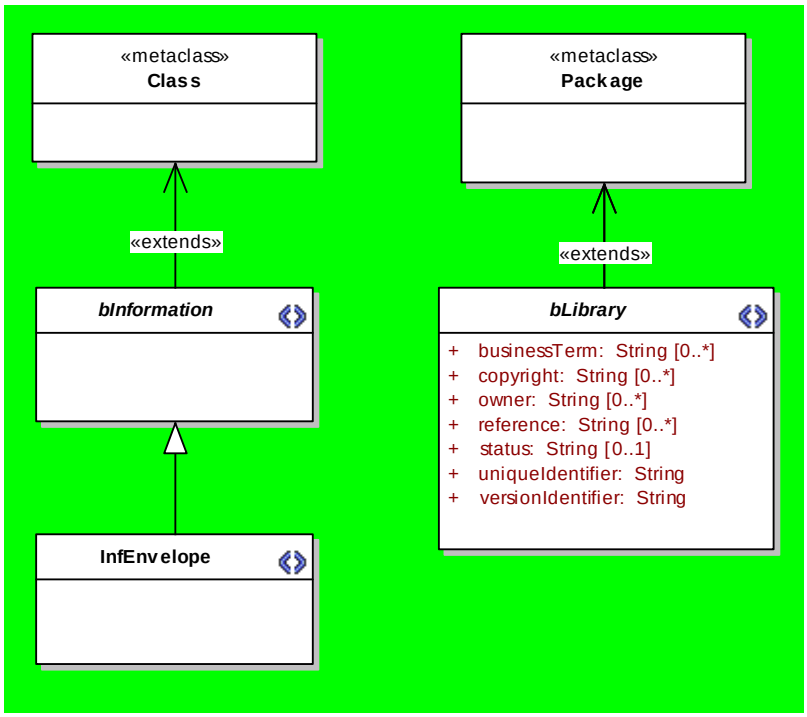


Figure 4 UMM Base Abstract Syntax

5 UMM Foundation Module

5.0 Foundation Module Management

5.0.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bCollModel	BusinessCollaborationModel
bRequirementsV	BusinessRequirementsView
bChoreographyV	BusinessChoreographyView
bInformationV	BusinessInformationView
bLibrary	BusinessLibrary

5.0.2 Conceptual Description (informative)

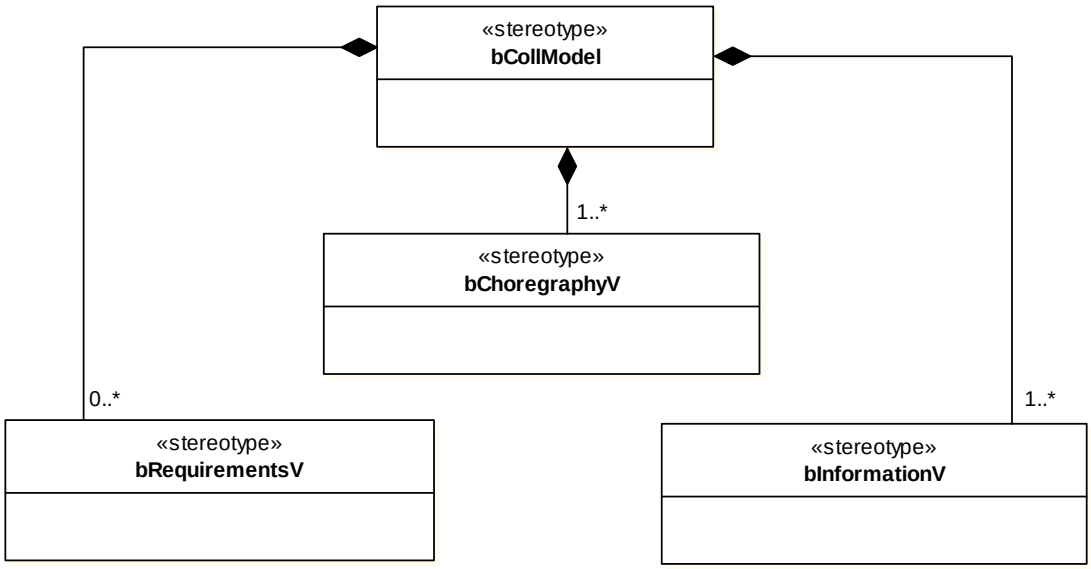


Figure 5 UMM Foundation Module Management - Conceptual Overview

A project that follows the UMM approach leads to a business collaboration model. A business collaboration model that is UMM compliant is stereotyped as *bCollModel*. As described above, the UMM is built by three views. The business requirements view stereotyped as *bRequirementsV* is optional and may occur multiple times in a business collaboration model. The business choreography view (stereotyped as *bChoreographyV*) and the business information view (stereotyped as bInformationV) are mandatory parts of a business collaboration model and may also occur multiple times.

Within the business requirements view the specific requirements of the business collaboration between two or more business partners are captured. The collected information from the business collaboration is then further elaborated within the business choreography view. The information exchanged during the process is modeled in the business information view. For further information on the specific sub-views of the UMM, please see the relevant sub-chapters of this specification.

5.0.3 Stereotypes and Tag Definitions (normative)

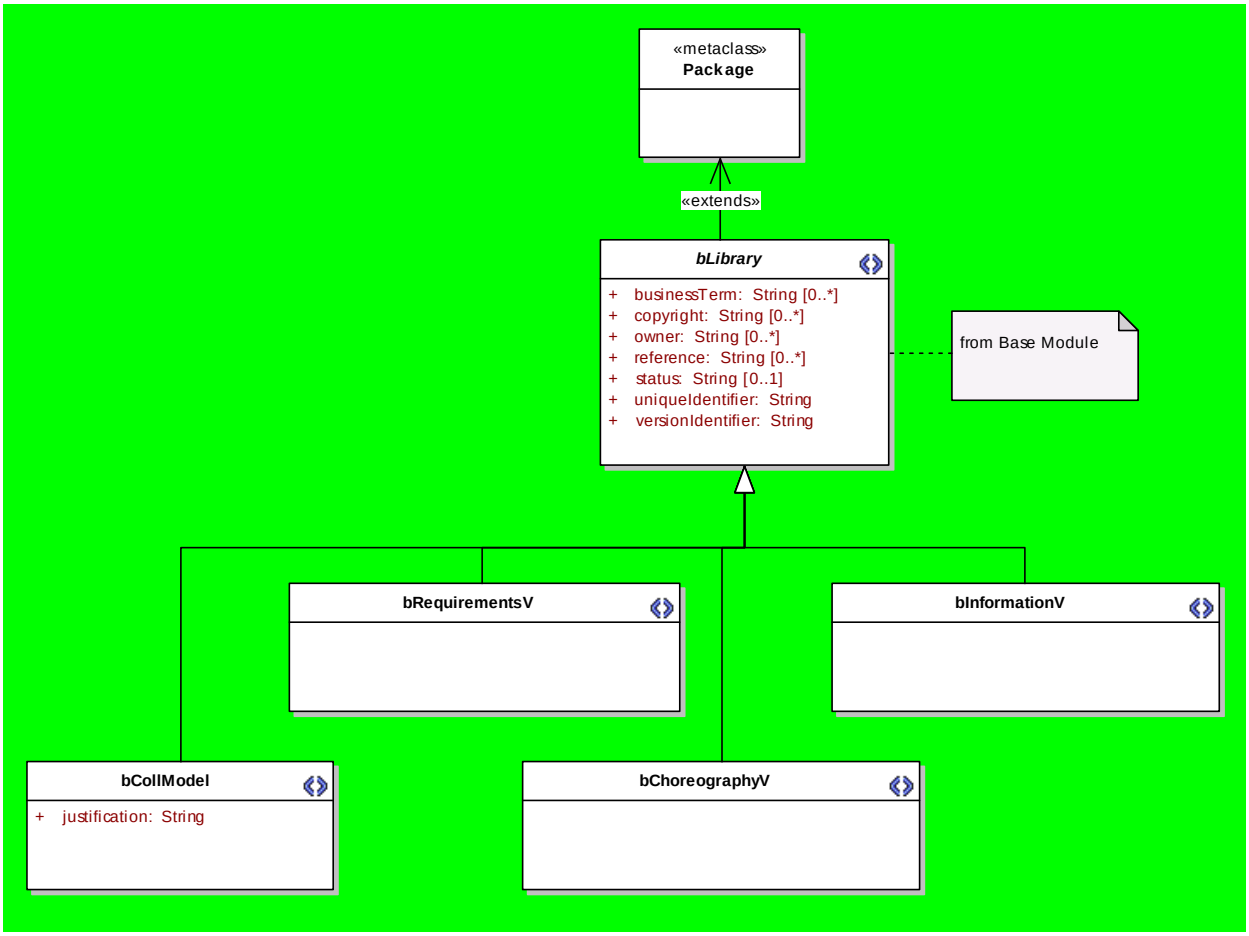


Figure 6 UMM Foundation Module Management - Abstract Syntax

Stereotype	**bCollModel (BusinessCollaborationModel)**	
Base Class	Package	
Parent	BusinessLibrary (from Base Module)	
Description	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 specialization and/or extension modules. Since a business collaboration model is of base class package, a UML model MAY contain one to many business collaboration models. Therefore, either the root element of a UML model is stereotyped as business collaboration model or any of the packages beneath the root element.	
Tag Definition	**justification**	
	Type	String
	Multiplicity	1
	Description	Explains the reason from a business perspective why the given business case is considered for possible business collaborations.

	Inherited tagged values: – businessTerm – copyright – owner – reference – status – uniqueIdentifier – versionIdentifier

369

Stereotype	**bRequirementsV (BusinessRequirementsView)**
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business requirements view is a container for all elements needed to identify and describe the requirements of collaborations between business partners. It captures the relevant packages which are used for discovering relevant business processes and their business partners executing/participating in them, as well as the lifecycle and state changes of business entities which are important within a business process.
Tag Definition	**Inherited tagged values:** – businessTerm – copyright – owner – reference – status – uniqueIdentifier – versionIdentifier

370

Stereotype	**bChoreographyV (BusinessChoreographyView)**
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business choreography view is a container for all elements needed to describe the choreography of business collaborations at various levels.
Tag Definition	**Inherited tagged values:** – businessTerm – copyright – owner – reference – status – uniqueIdentifier – versionIdentifier

371

Stereotype	**bInforationV (BusinessInformationView)**

Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business information view is a container for all elements representing the exchanged information in business collaborations.
Tag Definition	**Inherited tagged values:** – businessTerm – copyright – owner – reference – status – uniqueIdentifier – versionIdentifier

372

373 ### 5.0.4 Constraints (normative)

374

375 Constraints with respect to the *BusinessCollaborationModel* (bCollModel)

376 C.1. A BusinessCollaborationModel MUST contain one to many BusinessChoreographyViews
377 C.2. A BusinessCollaborationModel MUST contain one to many BusinessInformationViews
378 C.3. A BusinessCollaborationModel MAY contain zero to many BusinessRequirementsViews
379 C.4. A BusinessRequirementsView, a BusinessChoreographyView and a BusinessInformationView MUST
380 be directly located under a BusinessCollaborationModel

381 ## 5.1 Business Requirements View

382 ### 5.1.0 Sub-Views in the Business Requirements View

383 #### *5.1.0.1 Abbreviations and Stereotypes*

384

Stereotype Abbreviation	Full Stereotype Name
bRequirementsV	BusinessRequirementsView
bDomainV	BusinessDomainView
bPartnerV	BusinessPartnerView
bEntityV	BusinessEntityView

385

5.1.0.2 Conceptual Description (informative)

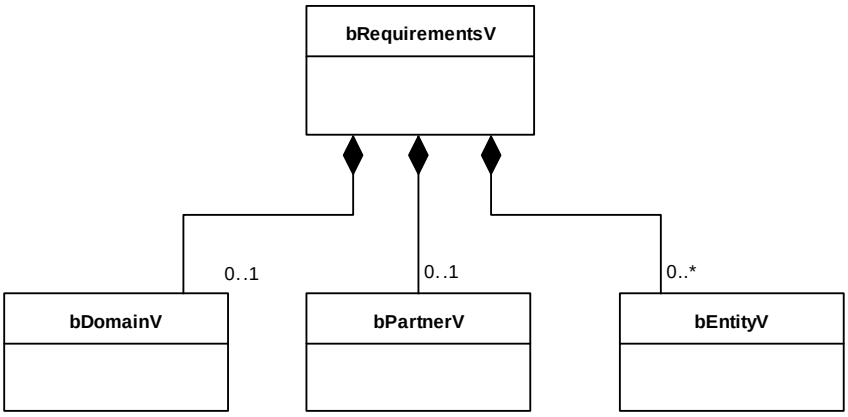


Figure 7 BusinessRequirementsView Conceptual Overview

The *BusinessRequirementsView* is composed by three significant sub-views which are all of optional use. Firstly, the *BusinessDomainView* captures all of the business processes which may be of interest for the domain under consideration. In order to enable users to readily identify business processes, these business processes are classified into business categories. This classification is done by creating business areas and process areas. However, UN/CEFACT recommends using this classification, but it is not mandatory. Secondly, the *BusinessPartnerView* specifies a list of business partners and stakeholders that are involved in the business processes defined in the *BusinessDomainView*. Furthermore the relationships between each others are defined in this package. Finally, the *BusinessEntityView* defines the business entities that are involved in a business process. A business entity is a real-world thing having business significance that is shared among two or more business partner in a collaborative business process (e.g. order, account, etc.). It is important to depict the possible state changes of such business entities within the *BusinessEntityView* in order to get an understanding of how a collaborative business process affects such real-world things during the execution of a business process.

5.1.0.3 Stereotypes and Tag Definitions (normative)

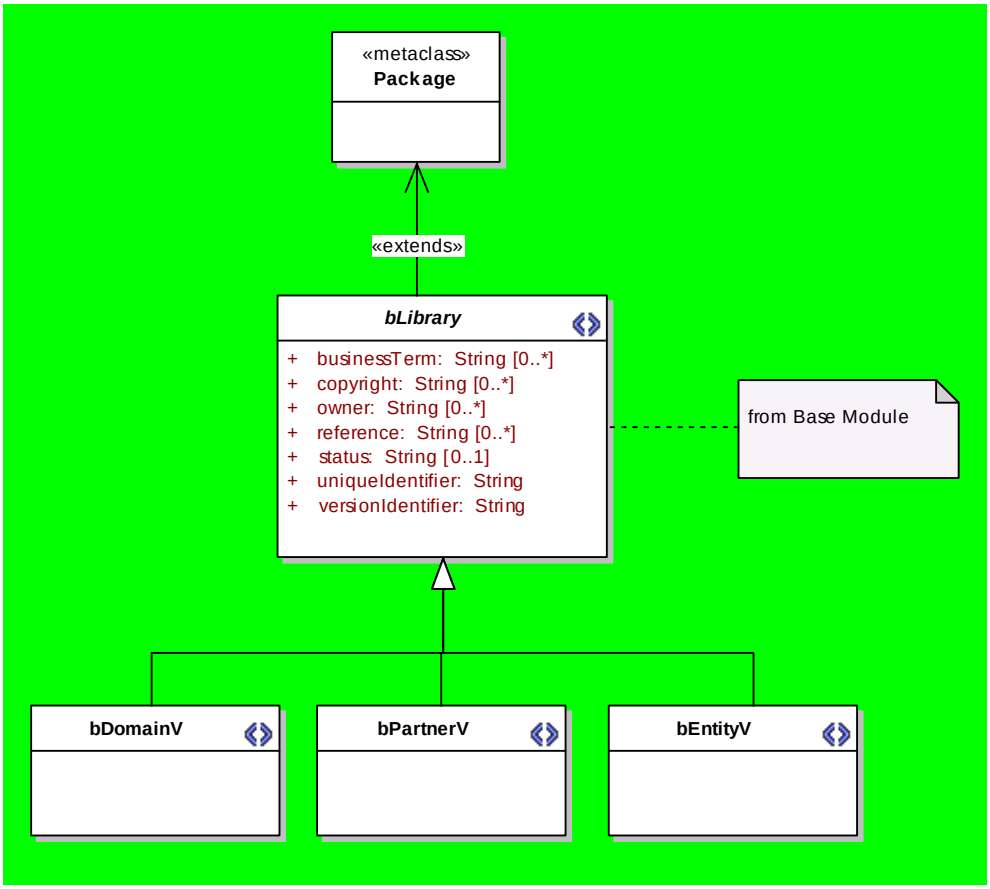


Figure 8 BusinessRequirementsView Abstract Syntax

Stereotype	bDomainV (BusinessDomainView)
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business domain view is used to discover business processes that are of relevance in a project. A business domain is a framework for identification and understanding of business processes as well as for categorizing them according to a classification schema. The business domain view is a container capturing the categorization scheme and categorized business processes.
Tag Definition	**Inherited tagged values:** – owner – copyright – reference – status – businessTerm – uniqueIdentifier – versionIdentifier

Stereotype	bPartnerV (BusinessPartnerView)
Base Class	Package

Parent	BusinessLibrary (from Base Module)
Description	The business partner view captures a list of business partners and stakeholders in the domain under consideration as well as the relationships between them.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – owner – copyright – reference – versionIdentifier – status – businessTerm

Stereotype	**bEntityV (BusinessEntityView)**
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business entity view is a container to describe the lifecycle of a business entity having business significance in the modelled domain including its' business entity states.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

5.1.0.4 Constraints (normative)

Constraints with respect to a *BusinessRequirementsView*:

C.5. A BusinessRequirementsView MAY contain zero or one BusinessDomainView.

C.6. A BusinessRequirementsView MAY contain zero or one BusinessPartnerView.

C.7. A BusinessRequirementsView MAY contain zero to many BusinessEntityViews.

C.8. A BusinessDomainView, a BusinessPartnerView, and a BusinessEntityView MUST be located directly under a BusinessRequirementsView.

5.1.1 Business Domain View

5.1.1.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bDomainV	BusinessDomainView
bCategory	BusinessCategory
bArea	BusinessArea

ProcessArea	ProcessArea
bProcessUC	BusinessProcessUseCase
bProcess	BusinessProcess
bProcessAction	BusinessProcessAction
bESharedState	SharedBusinessEntityState
bEInternalState	InternalBusinessEntityState
participates	participates
isOfInterestTo	isOfInterestTo

421

5.1.1.2 Conceptual Description (informative)

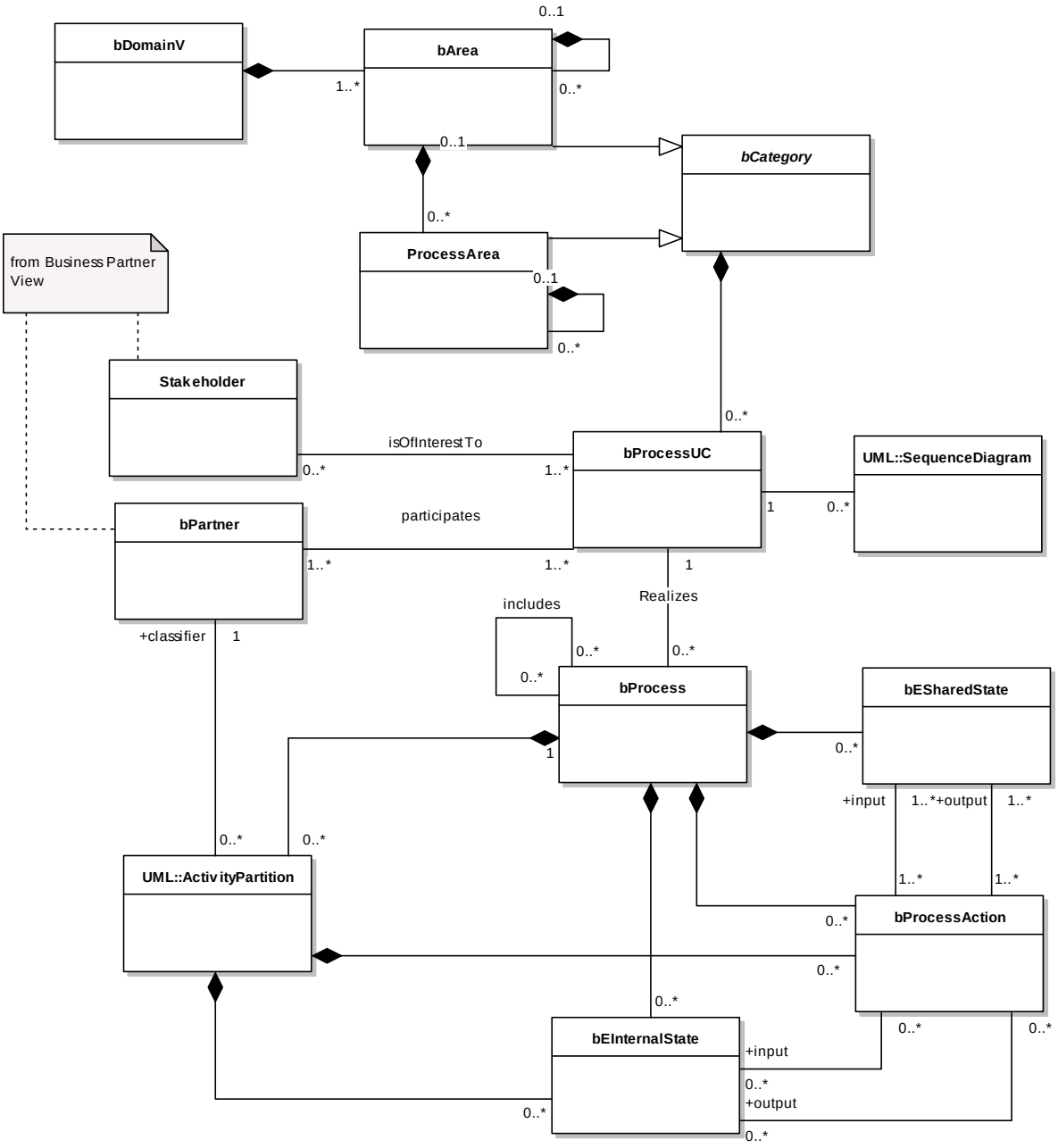


Figure 9 BusinessDomainView – Conceptual Overview

The business domain view is used to discover business processes use cases that are of relevance in a project. A business process use case is executed by at least one (but possibly more) business partners. A business partner might execute multiple business process use cases. Thus, the *participates* association between *BusinessPartner* and *BusinessProcessUseCase* is a (1..n) to (0..n) association. A stakeholder does not need to participate in a business process use case. A stakeholder might have interest in multiple business process use cases and a business process use case might be of interest to multiple stakeholders. The relationship between a *BusinessProcessUseCase* 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.

To enable users to readily identify business process use cases, they should be classified into business categories. A business category is an abstract concept, which has two concrete specializations – business area and process area. 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. A business area might be composed of other business areas. This means, a business area may form a hierarchy. Thus, a unary (0..1) to (0..n) composition is defined for a *BusinessArea*. The lowest level of a business area hierarchy includes process areas or business processes use cases. Therefore, we have a (0..1) to (0..n) composition between *BusinessArea* and *ProcessArea.* Furthermore, a *BusinessArea* may also include 0...n *BusinessProcessUseCases* if no further classification using process areas is required. Similar to a business area, a process area may form a hierarchy. This means, a unary (0..1) to (0..n) composition is defined for a *ProcessArea.* Similar to a business area, a *ProcessArea* may contain zero to many *BusinessProcessUseCases*. On the lowest level of a *ProcessArea* hierarchy, at least one *BusinessProcessUseCase* must be present.

The flow of a business process use case may be described by business processes. Thus, a *BusinessProcessUseCase* is realized by zero to many *BusinessProcesses*. A business process represents the dynamic behavior of a business process use case. A business process corresponds to a flow of actions performed by one participant or even by more participants. If two or more business partners collaborate, a business process is divided into partitions – one for each business partner. In case of an internal business process, which is executed by one partner only, a single partition for that partner is optional. Consequently, a *BusinessProcess* is composed of zero or more UML *ActivityPartitions*. An *ActivityPartition* is assigned to one *BusinessPartner*; a *BusinessPartner* is assigned to one *ActivityPartition*. However, a *BusinessPartner* may be assigned to multiple *ActivityPartitions* – each one in a different *BusinessProcess*. Hence, there is a 1 to (0..n) association between *BusinessPartner* and *ActivityPartition*.

A business process is described as a flow of business process actions. In the case where no activity partition is used, the business process actions are directly included in the Activity Diagram of the business process. In case of activity partitions, a business process action is assigned to the partition of the business partner executing the action. The need for a collaborative business process is identified whenever a transition connecting two business process actions crosses activity partitions. It follows, that either a *BusinessProcess* is composed of one or more *BusinessProcessAction* or an *ActivityPartition* (which is part of a business process) is composed of one or more *BusinessProcessActions*. A business process action might be refined by another business process. Thus a *BusinessProcessAction* is composed of zero or one *BusinessProcess* which in turn is a composite of zero or one *BusinessProcessActions*.

A business process 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 action and input to another business action. There is a transition from a business process action to a business entity state signaling an output as well as a transition from a business entity state to a business process action signaling an input. Some business entity states are meaningful to one business partner only. These are internal business entity states. Business entity states that must be communicated to a business partner are shared business entity states. A business process may include both internal and shared business entity states. Hence, a *BusinessProcess* is composed of zero to many *InternalBusinessEntityStates* and of zero to many *SharedBusinessEntityStates*. If a business process uses activity partitions, the two business process actions

creating and consuming an internal business entity state are in the same activity partition. In contrast, the two business process actions creating and consuming a shared business entity state are in different activity partitions. A shared business entity state signals the need for a collaborative business process.

5.1.1.3 Stereotypes and Tag Definitions (normative)

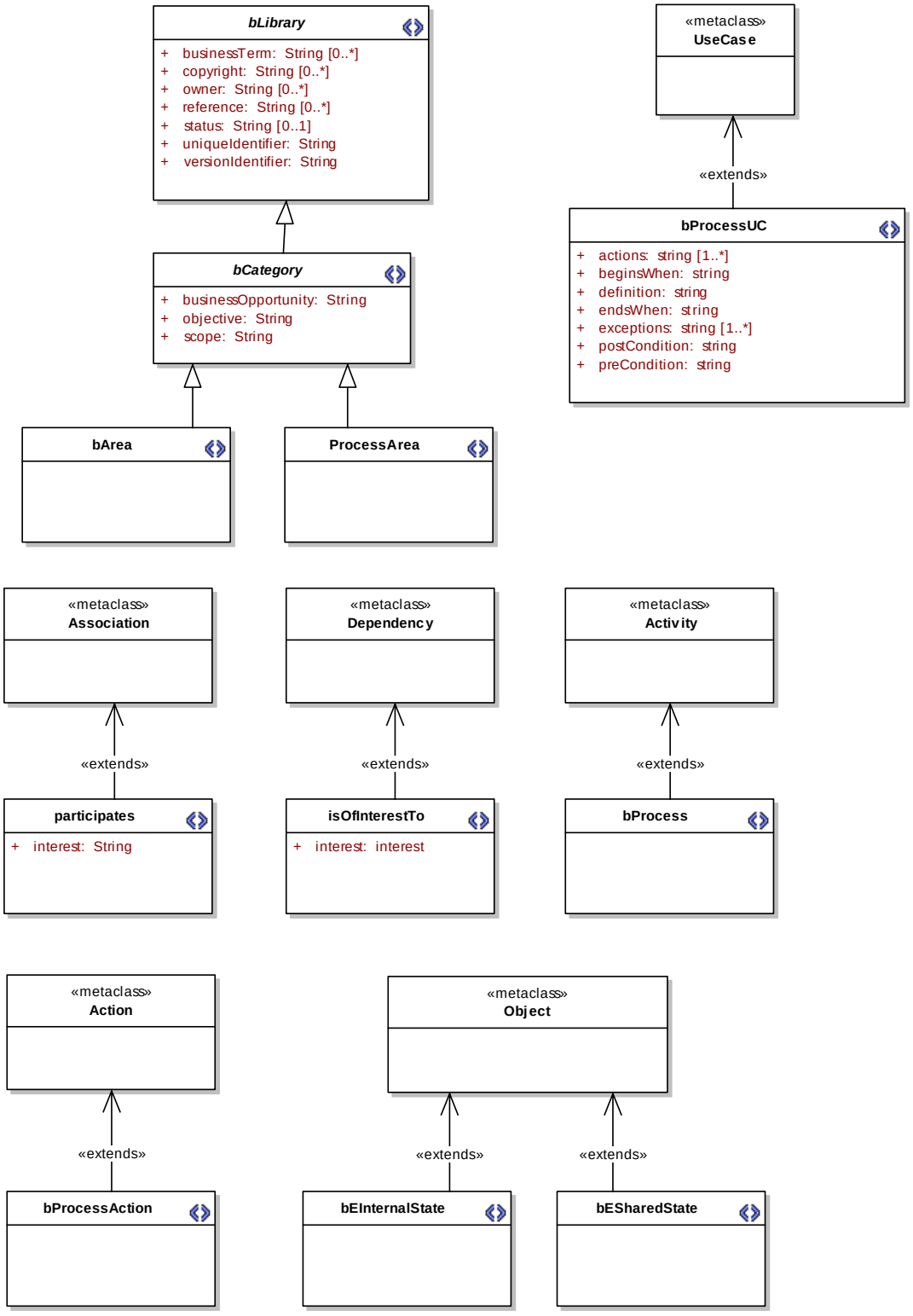


Figure 10 BusinessDomainView Abstract Syntax

Stereotype	bCategory (BusinessCategory, abstract)
Base Class	Package
Parent	BusinessLibraryPackage (from Base Module)
Description	A business category is an abstract concept. 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 by its specializations *BusinessArea* and *ProcessArea* (see below for these stereotype definitions).
Tag Definition	**objective** **Type**: String **Multiplicity**: 1 **Description**: The purpose to be achieved by the business process within the business category under consideration. **scope** **Type**: String **Multiplicity**: 1 **Description**: Defines the boundaries of the business category under consideration. **businessOpportunity** **Type**: String **Multiplicity**: 1 **Description**: The strategic interest from a business perspective in order to address the business category under consideration. **Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

481

Stereotype	bArea (BusinessArea)
Base Class	Package
Parent	BusinessCategory
Description	A business area usually corresponds to a division of an enterprise. Business areas might be structured recursively. A business area is a category of decomposable business areas or process areas (on the lowest

	level of business area hierarchy). This means that a business area collates either other business areas, process areas or business process use case. 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, and Health Care. This list of business areas is considered as non exhaustive.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – objective – scope – businessOpportunity – owner – copyright – reference – status – businessTerm

482

Stereotype	**ProcessArea (ProcessArea)**
Base Class	Package
Parent	BusinessCategory
Description	A process area corresponds to a set of common operations within a business area. Process areas might be structured recursively. A process area is a category of common business process use cases. This means a process area collates either other process areas or business process use cases. 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:** – uniqueIdentifier – versionIdentifier – objective – scope – businessOpportunity – owner – copyright – reference – status – businessTerm

483

Stereotype	**bProcessUC (BusinessProcessUseCase)**
Base Class	UseCase
Parent	N/A

Description	A business process use case is a set of related activities that together create value for a business partner. A business process use case might be performed by a single business partner type or by multiple business partner types crossing organizational boundaries. In the case where organizations collaborate in a business process, the business process should create value for all of its participants. A business process use case can be decomposed into sub-processes using the «include» and «extends» association stereotypes defined in UML.	
Tag Definition	**definition**	
	Type	String
	Multiplicity	1
	Description	Gives a definition of the business process use case. This definition must describe the customer value to be created by the business process use case. In the case of a business process use case executed by multiple parties, it describes the value to be created to all participants.
	beginsWhen	
	Type	String
	Multiplicity	1
	Description	Specifies a business event that triggers the initiation of the business process use case.
	preCondition	
	Type	String
	Multiplicity	1
	Description	Specifies a condition that has to be fulfilled in order to execute a business process use case. This condition SHOULD refer to states in the life cycle of a business entity. A pre-condition statement MAY use Boolean operators specifying a combination of multiple business entity states.
	endsWhen	
	Type	String
	Multiplicity	1
	Description	Specifies a business event that leads to the termination of the business process use case.
	postCondition	
	Type	String
	Multiplicity	1
	Description	Specifies a condition that will be reached after executing the business process use case. Usually, this condition SHOULD refer to states in the life cycle of a business entity. A post-condition statement MAY use Boolean operators specifying a combination of multiple business entity states.
	exceptions	

	Type	String
	Multiplicity	1..*
	Description	Identifies situations leading to a deviation of the regular execution of the business process use case.
	actions	
	Type	String
	Multiplicity	1..*
	Description	Lists the tasks that together make up a business process use case. In the case of a business process use case executed by multiple parties, a special emphasis on interface tasks is needed. An interface task is a step in the business process use case that requires communication with another business partner.

484

Stereotype	**participates (participates)**	
Base Class	Association	
Parent	N/A	
Description	Describes the association between a business partner and a business process use case. This stereotype defines that the business partner provides input to and/or output from the associated business process use case.	
Tag Definition	**interest**	
	Type	String
	Multiplicity	1
	Description	Describes the vested interest of the business partner type in the business process associated by this participates-association.

485

Stereotype	**isOfInterestTo (isOfInterestTo)**	
Base Class	Dependency	
Parent	N/A	
Description	Describes a dependency from a business process use case to a stakeholder. This stereotype defines that a business process use case depends on the interest of the connected stakeholder.	
Tag Definition	**interest**	
	Type	String
	Multiplicity	1
	Description	Describes the vested interest of the stakeholder in the business process use case linked by this is of interest to dependency.

486

Stereotype	bProcess (BusinessProcess)
Base Class	Activity
Parent	N/A
Description	The business process describes the behavior of a business process use case between the involved business partners. It is a tool to identify requirements to collaborate between two or more business partners. A business process refines a business process use case by describing its dynamic behaviour.
Tag Definition	No tagged values.

487

Stereotype	bProcessAction (BusinessProcessAction)
Base Class	Action
Parent	N/A
Description	A business process action corresponds to a step in the execution of a business process. A business process action might be refined by another business process. In this case, a UML call behavior action MUST be used as base class for the business process action
Tag Definition	No tagged values.

488

Stereotype	bEInternalState (InternalBusinessEntityState)
Base Class	ObjectNode
Parent	N/A
Description	The internal business entity state represents a state of a business entity that is internal to the business process of a business partner.
Tag Definition	No tagged values.

489

Stereotype	bESharedState (SharedBusinessEntityState)
Base Class	ObjectNode
Parent	N/A
Description	The shared business entity state represents a state of a business entity that is shared between the business processes between two involved business partners.
Tag Definition	No tagged values.

490

491 ### *5.1.1.4 Worksheets*

Form for Business Domain View	
General	
Name	
Description	
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Business Area(s)	
Business Area No 1	
Business Area No 2	
Business Area No 3	
Business Area No 4	
Business Area No 5	
Business Area No 6	(add columns as needed)

492

Form for Business Area	
General	
Name	
Description	
Details	

Objective	
Scope	
Business Opportunity	
Included in	(insert the parent Business Area or Business Domain View)
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Business Area(s)	(insert additional nested business areas if appropriate; otherwise fill process areas that apply)
Business Area No 1	
Business Area No 2	
Business Area No 3	
Business Area No 4	
Business Area No 5	(add columns as needed)
Process Area(s)	(you only fill process areas if you do not have completed a business area above)
Process Area No 1	
Process Area No 2	
Process Area No 3	
Process Area No 4	
Process Area No 5	(add columns as needed)

493

Form for Process Area	
General	
Name	
Description	
Details	
Objective	
Scope	
Business Opportunity	
Included in	(insert the parent Business Area or Process Area)
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Process Area(s)	**(if present)**
Process Area No 1	
Process Area No 2	
Process Area No 3	
Process Area No 4	
Process Area No 5	(add columns as needed)

494

495

Form for Business Process	
General	
Name	
Description	
Details	
Classified to Business Areas and Process Areas	
Participants and their interests	
Stakeholders and their interests	
Reference(s)	
Start/End Characteristics	
Pre-condition	
Post-condition	
Begins When	
Ends When	
Actions	
Exceptions	
Relationships	
Included Business Processes	
Affected Business Entities and their states	

5.1.1.5 Constraints (normative)

C.9. A *BusinessDomainView* MUST include one to many *BusinessAreas*.

C.10. A *BusinessArea* MUST include one to many *BusinessAreas* or one to many *ProcessAreas* or one to many *BusinessProcessUseCases*.

C.11. A *ProcessArea* MUST contain one to many other *ProcessAreas* or one to many *BusinessProcessUseCases*

C.12. *A BusinessProcessUseCase* MUST be associated with one to many *BusinessPartners* using the *participates* relationship

C.13. *A BusinessProcessUseCase* MAY be associated with zero to many *Stakeholders* using the *isOfInterestTo* relationship

C.14. A *BusinessProcessUseCase* SHOULD be refined by zero to many *BusinessProcesses.* These relationships MAY also be visualized by realize relationships from each of the owned *BusinessProcesses* to the owning *BusinessProcessUseCase*

C.15. A BusinessProcess MUST be modeled as a child of a BusinessProcessUseCase

C.16. A *BusinessProcessUseCase* MAY be refined by zero to many UML Sequence Diagrams

C.17. A BusinessProcess MAY contain zero to many ActivityPartitions

C.18. A BusinessProcess, which has no ActivityPartitions, MUST contain one or more BusinessProcessActions and MAY contain zero to many InternalBusinessEntityStates and zero to many SharedBusinessEntityStates.

C.19. An *ActivityPartition* being part of a *BusinessProcess* MUST contain one to many *BusinessProcessActions* and MAY contain zero to many *InternalBusinessEntityStates.*

C.20. A *SharedBusinessEntityState* MUST NOT be located in an *ActivityPartition.* (They must be contained within the *BusinessProcess* even if this BusinessProcess contains *ActivityPartitions*.)

5.1.1.6 Example (informative)

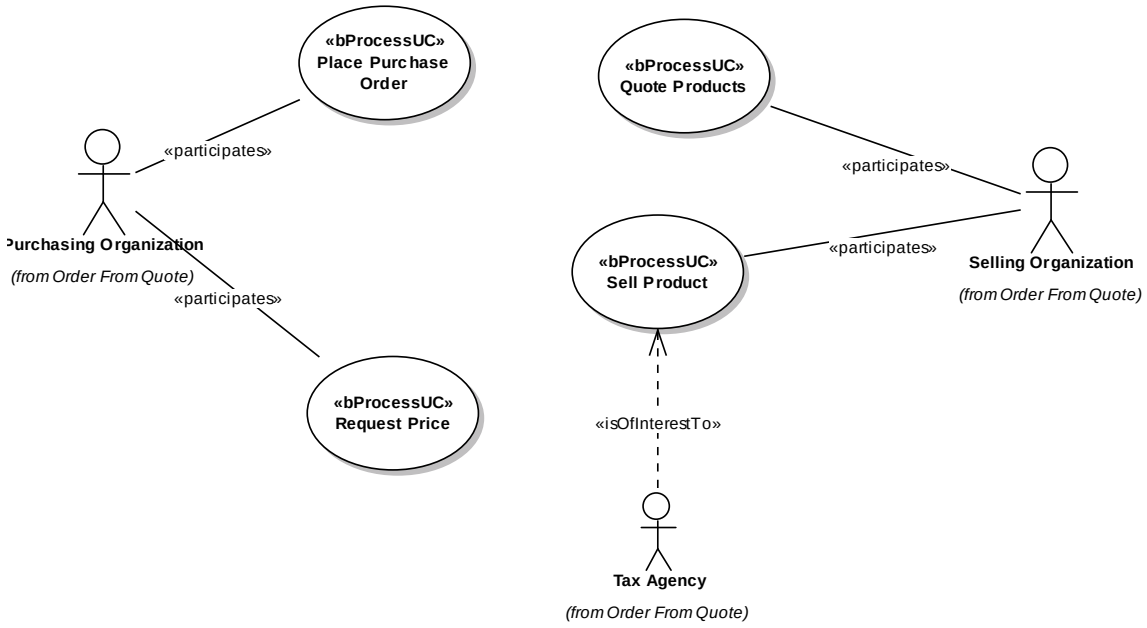
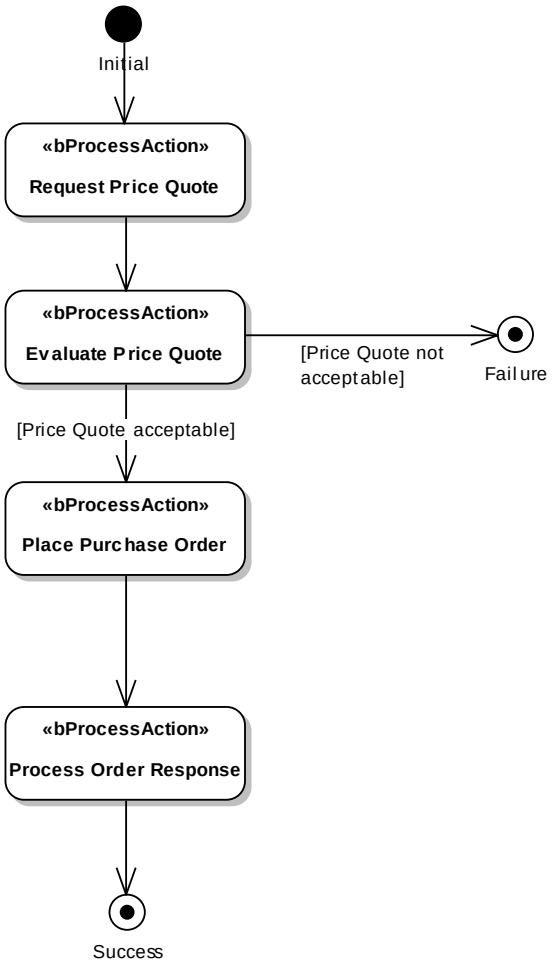
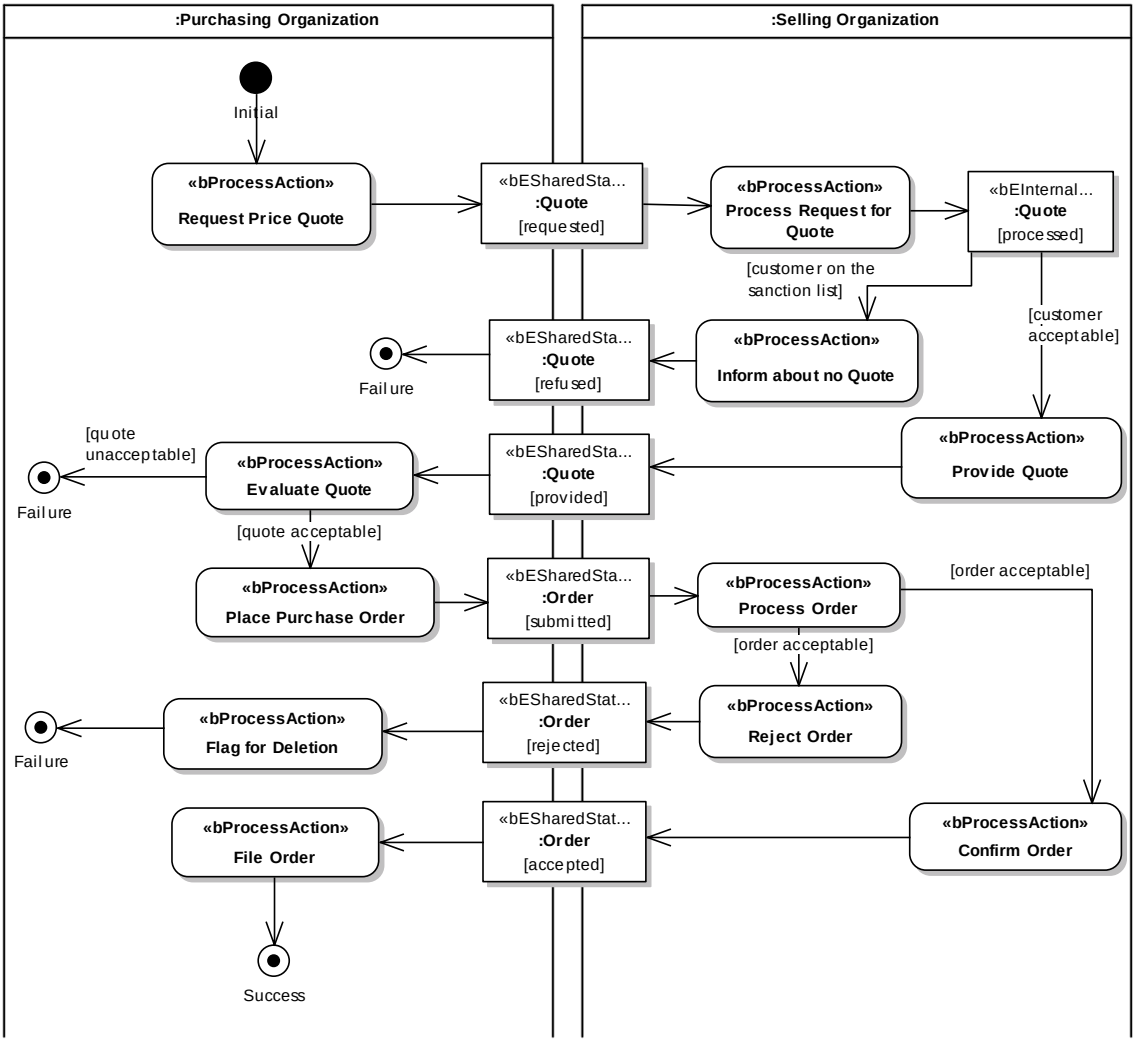


Figure 11 Business Domain View Example: Negotiation In the Order from Quote Example (Use Case Diagram showing Business Process Use Cases)



525

526 **Figure 12 Business Domain View Example: Business Process of the internal Place Order Business Process Use Case (Activity**
527 **Diagram)**



528

529 **Figure 13 Business Domain View Example: Business Process of the to-be-designed inter-organizational process called Purchase**
530 **Product (Activity Diagram)**

531

532 5.1.2 Business Partner View

533 *5.1.2.1 Abbreviations of Stereotypes*

Stereotype Abbreviation	Full Stereotype Name
bPartnerV	BusinessPartnerView
bPartner	BusinessPartner
Stakeholder	Stakeholder

5.1.2.2 Conceptual Description (informative)

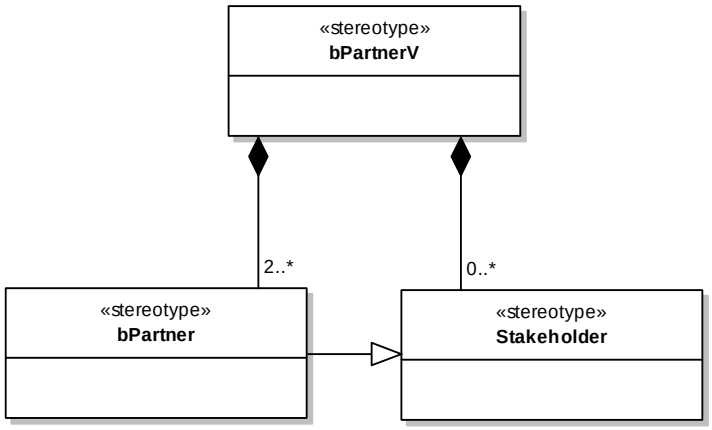


Figure 14 BusinessPartnerView – Conceptual Overview

A business partner is an organization type, an organizational unit type or a person type that participates in a business process. A *BusinessPartnerView* must contain at least two *BusinessPartners*. 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. By definition, a business partner always has a vested interest in the business processes which they are participating in. Therefore, a *BusinessPartner* is a special type of a *Stakeholder*. In UML, specific relationships between Actors MAY be defined. The business partner view does not restrict the definition of those relationships between business partners and/or stakeholders. For example, generalizations between business partners MAY be defined.

5.1.2.3 Stereotypes and Tag Definitions (normative)

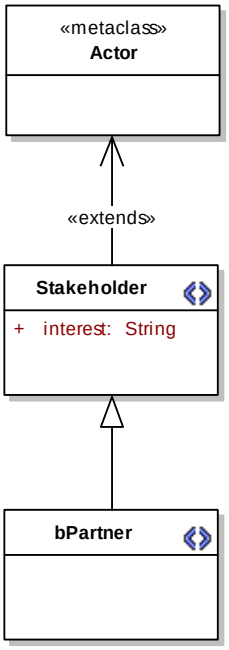


Figure 15 BusinessPartnerView – Abstract Syntax

Stereotype	Stakeholder (Stakeholder)	
Base Class	Actor	
Parent	N/A	
Description	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.	
Tag Definition	**interest**	
	Type	String
	Multiplicity	1
	Description	Describes the vested interest of the stakeholder in the business category it is defined within.

548

Stereotype	bPartner (BusinessPartner)
Base Class	Actor
Parent	Stakeholder
Description	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, they have, by default, a vested interest in the business process. Therefore, a business partner type is a special kind of stakeholder.
Tag Definition	**Inherited tagged values:** - interest

549 ### *5.1.2.4 Constraints (normative)*

550 C.21. A *BusinessPartnerView* MUST contain at least two to many *BusinessPartners.* If the
551 *BusinessPartnerView* is hierarchically decomposed into sub packages these *BusinessPartners* MAY be
552 contained in any of these sub packages.

553 C.22. A BusinessPartnerView MAY contain zero to many Stakeholders

554 ### *5.1.2.5 Example (informative)*

555

556 **Figure 16 Business Partner View Example**

5.1.3 Business Entity View

5.1.3.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bEntityV	BusinessEntityView
bEntity	BusinessEntity
bEState	BusinessEntityState
bDataV	BusinessDataView

5.1.3.2 Conceptual Description (informative)

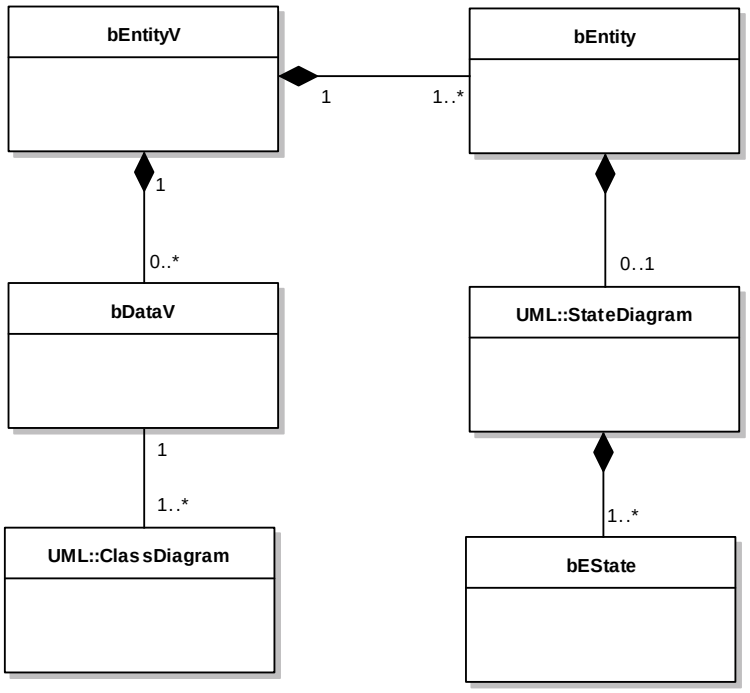


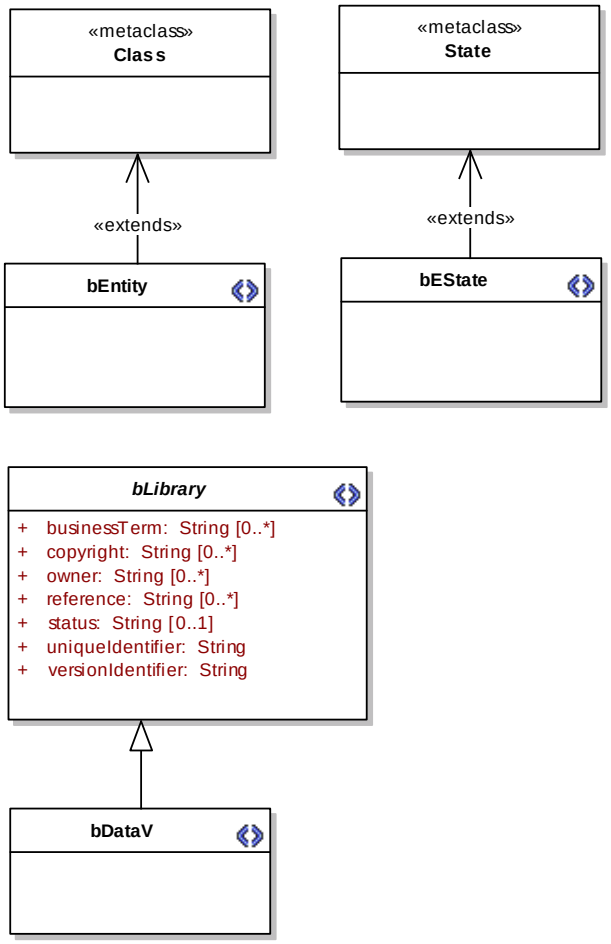
Figure 17 BusinessEntityView (BusinessRequirementsView) Conceptual Overview

A business entity is a real-world thing having business significance that is shared between two or more business partners in a collaborative business process (e.g. “order”, “account”, etc.). Within the business entity view at least one, but possibly more business entities are described. Thus, the *BusinessEntityView* is composed of one to many *BusinessEntities*. The lifecycle of a business entity MAY be described as a flow of business entity states. Depending on the importance of the business entity lifecycle, the lifecycle may or may not be included. A lifecycle is described using a UML State Diagram. Hence, a *BusinessEntity* is composed of zero to one UML State Diagram. The lifecycle represents the different business entity states a business entity can exist in. The lifecycle of a business entity consists of at least one business entity state. Therefore, the lifecycle of a business entity is composed of one or more *BusinessEntityStates*.

A business entity is a potential candidate for becoming a business document in later steps of the UMM. A business data view MAY be used to elaborate a first conceptual design of a business entity. Hence, a *BusinessEntity* is composed of zero to one *BusinessDataViews*. Within a business data view, A UML class

575 diagram is used to describe the assembly of a business entity. Thus, a *BusinessDataView* contains one to
576 many UML Class Diagrams.

577 ### *5.1.3.3 Stereotypes and Tag Definitions (normative)*



578

579 **Figure 18 BusinessEntityView (BusinessRequirementsView) Abstract Syntax**

580

Stereotype	bEntity (BusinessEntity)
Base Class	Class
Parent	N/A
Description	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.).
Tag Definition	No tagged values.

581

Stereotype	bEState (BusinessEntityState)
Base Class	State
Parent	N/A
Description	A business entity state represents a specific state a business entity can exists in during its lifecycle (an "order" can exist in the states "issued", "rejected", "confirmed", etc.)
Tag Definition	No tagged values.

Stereotype	bDataV (BusinessDataView)
Base Class	Package
Parent	BusinessLibrary
Description	The business data view is a container for all elements needed to describe the conceptual assembly of a business entity
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

5.1.3.4 Constraints (normative)

C.23. A *BusinessEntityView* MUST contain one to many *BusinessEntities*

C.24. A *BusinessEntity* SHOULD have one UML State Diagram that describe its lifecycle

C.25. A UML State Diagram describing the lifecycle of a *BusinessEntity* MUST contain one to many *BusinessEntityState*s. The parent of a *BusinessEntityState* MUST be a *BusinessEntity*.

C.26. A *BusinessEntityView* MAY contain zero to many *BusinessDataView* that describe its conceptual design

C.27. The parent of a BusinessDataView MUST be a BusinessEntityView

C.28. A BusinessDataView SHOULD use a UML Class Diagram to describe the conceptual design of a *BusinessEntity*

C.29. A *BusinessDataView* SHOULD contain one to many classes.

5.1.3.5 Worksheets

Form for Business Entity	
General	
Business Entity Name	
Description	

Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Lifecycle	
Pre-Condition	
Post-Condition	
Begins When	
Ends When	
Exceptions	
Lifecycle States (add more Business Entity States if needed)	
Business Entity State	
Name	
Description	
Preceding State(s) including events and transition conditions	
Valid Actions	
Business Entity State	
Name	
Description	
Preceding State(s) including events and transition conditions	

Valid Actions	
Business Entity State	
Name	
Description	
Preceding State(s) including events and transition conditions	
Valid Actions	
Business Entity State	
Name	
Description	
Preceding State(s) including events and transition conditions	
Valid Actions	
Business Entity State	
Name	
Description	
Preceding State(s) including events and transition conditions	
Valid Actions	

596

5.1.3.6 *Example (informative)*

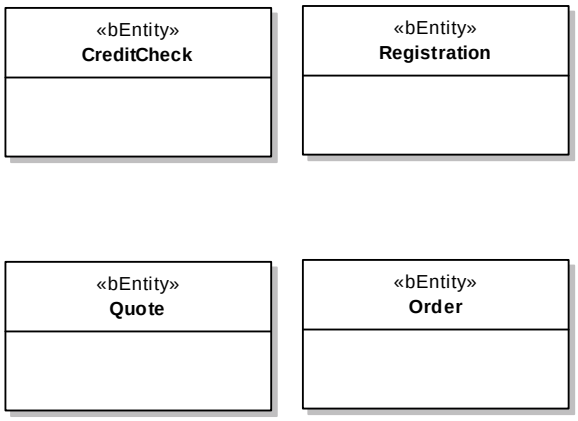


Figure 19 BusinessEntityView Example: BusinessEntities - CreditCheck, Registration, Quote and Order (Class Diagram)

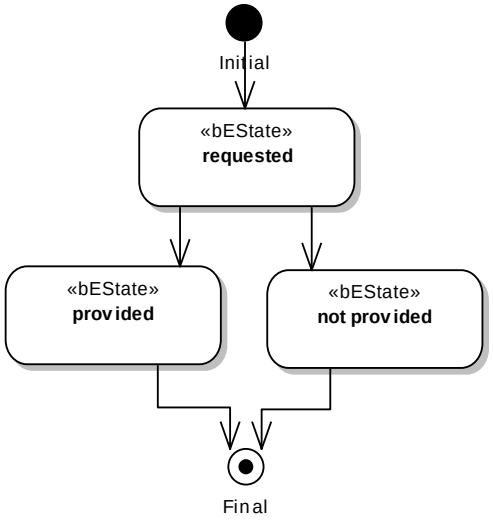
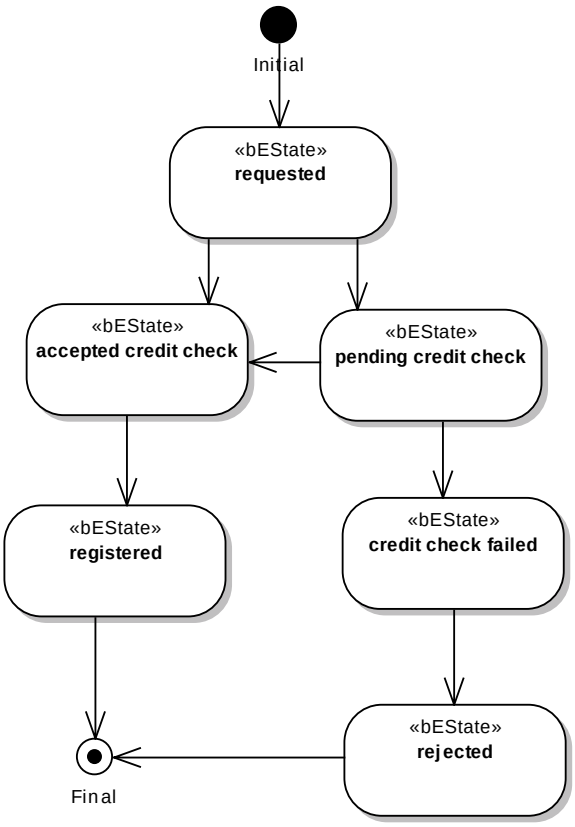
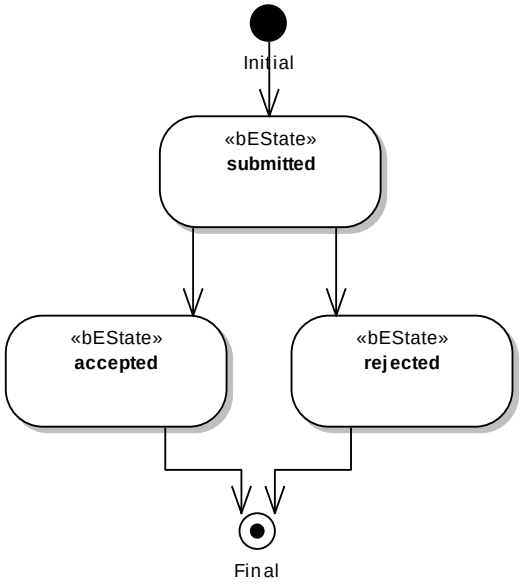


Figure 20 BusinessEntityView Example: CreditCeck Lifecycle (State Diagram)



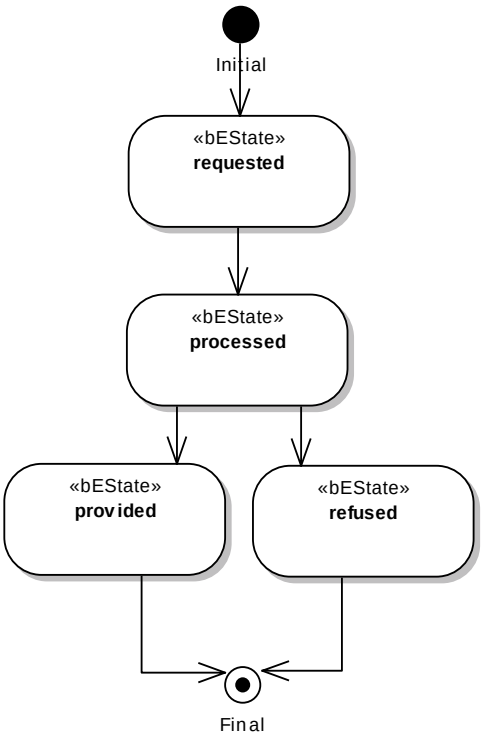
602

603 **Figure 21 BusinessEntityView Example: Registration Lifecycle (State Diagram)**



604

605 **Figure 22 BusinessEntityView (BusinessRequirementsView) Example: Order BusinessEntityLifecycle (State Diagram)**



606

607 **Figure 23 BusinessEntityView (BusinessRequirementsView) Example: Quote BusinessEntityLifecycle (State Diagram)**

608

609 5.2 Business Choreography View

610 5.2.1 Sub-Views in the Business Choreography View

611 *5.2.1.1 Abbreviations and Stereotypes*

612

Stereotype Abbreviation	Full Stereotype Name
bChoreographyV	BusinessChoreographyView
bCollaborationV	BusinessCollaborationView
bTransactionV	BusinessTransactionView
bRealizationV	BusinessRealizationView

613

5.2.1.2 Conceptual Description (informative)

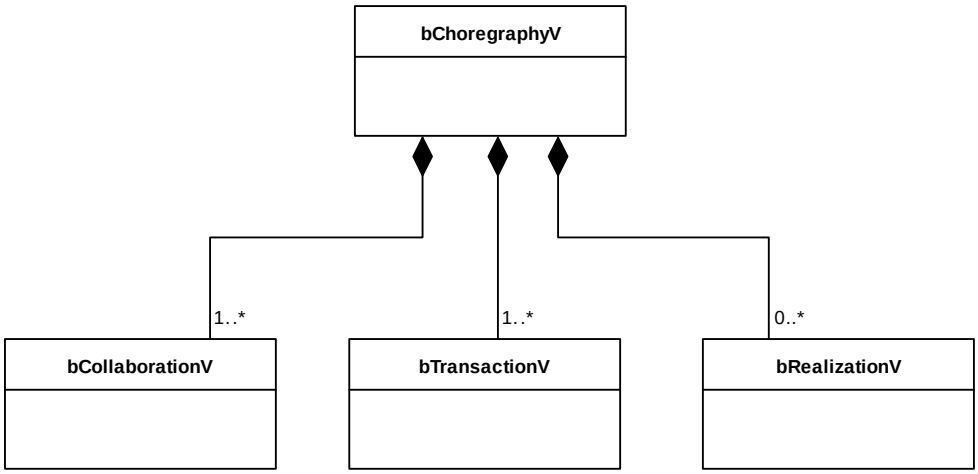
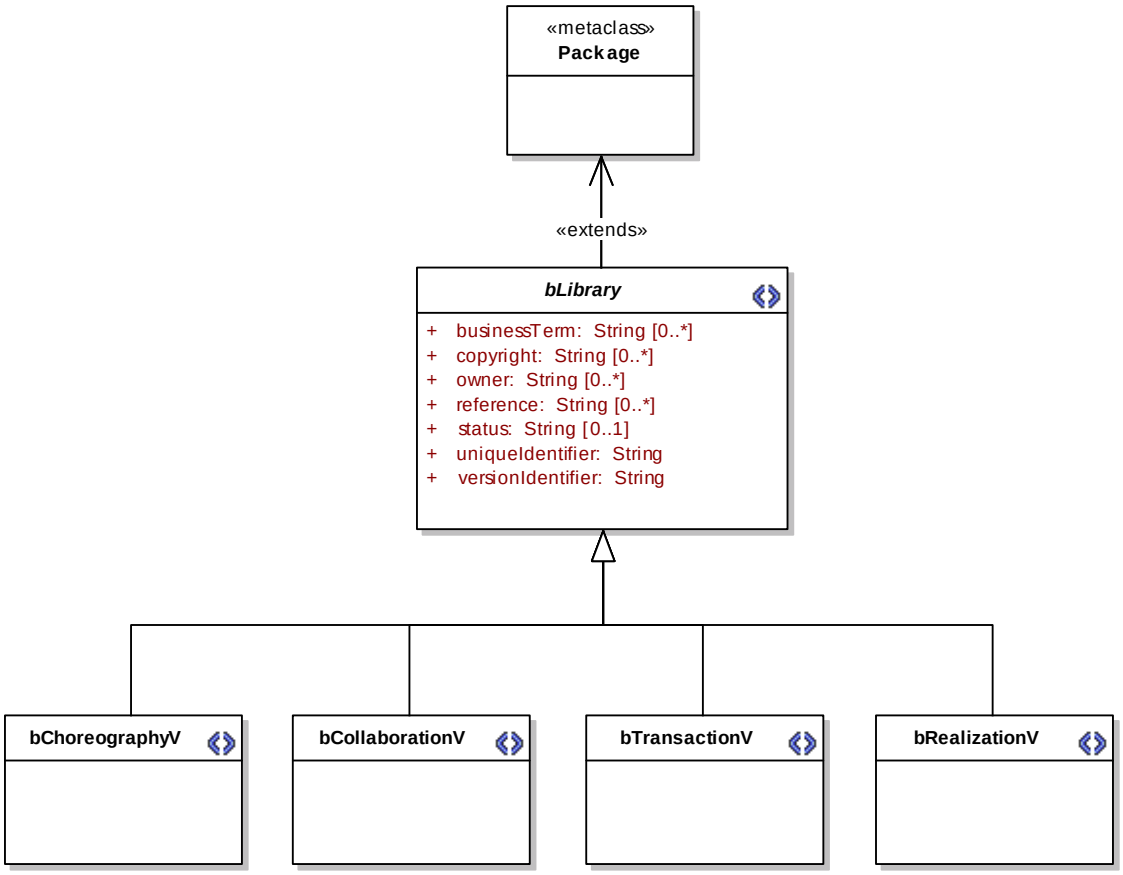


Figure 24 BusinessChoreographyView Conceptual Overview

The *BusinessChoreographyView* is the second out of the 3 views of a UMM compliant business collaboration model. The business choreography view describes the view how the business analyst sees the process to be modeled. The requirements captured in the business requirements view serve as a basis for the definition of a choreography of information exchanges. The business choreography view is a container for three different artifacts that together describe the overall choreography of information exchanges. A *BusinessTransactionView* 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 transaction view captures two different artifacts that define the business transaction. First, the business analyst defines concrete requirements specifying the business transaction on a more general level by using business transaction use cases. Second, he defines the flow of information exchanges in accordance to the requirements specified in this container. The business collaboration view is a container for artifacts describing the flow of a complex business collaboration between business partner types that may involve many steps. Similar to the business transaction view, the *BusinessCollaborationView* captures two different artifacts as well. Once the business analyst has specified the concrete requirements for a business collaboration by using business collaboration use cases, he is able to define the flow in accordance to the requirements defined in this container. Finally, the *CollaborationRealizationView* describes the realization of a business collaboration use case for a specific set of business partner types.

634 ### 5.2.1.3 Stereotypes and Tag Definitions (normative)



635
636 **Figure 25 BusinessChoreographyView Abstract Syntax**

637

Stereotype	bChoreographyV (BusinessChoreographyView)
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business choreography 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.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

638

Stereotype	bCollaborationV (BusinessCollaborationView)

Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business collaboration view is a container for artifacts describing the flow of a complex business collaboration between business partner types that may involve many steps.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

639

Stereotype	**bTransactionV (BusinessTransactionView)**
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The transaction requirements view is a container for artifacts that define a choreography leading to synchronized states of business entities at both sides of the business transaction.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

640

Stereotype	**bRealizationV (BusinessRealizationView)**
Base Class	Package
Parent	BusinessLibrary (from Base Module)
Description	The business realization view is a container for all elements describing the realization of a business collaboration use case by business partner types.
Tag Definition	**Inherited tagged values:** – uniqueIdentifier – versionIdentifier – owner – copyright – reference – status – businessTerm

641

642

5.2.1.4 Constraints (normative)

Constraints with respect to a *BusinessChoreographyView*:

C.30. A BusinessChoreographyView MUST contain one to many BusinessCollaborationViews

C.31. A BusinessChoreographyView MUST contain one to many BusinessTransactionViews.

C.32. A BusinessChoreographyView MAY contain zero to many BusinessRealizationViews.

C.33. A BusinessTransactionView, a BusinessCollaborationView, and a BusinessRealizationView MUST be directly located under a BusinessChoreographyView

5.2.2 Business Transaction View

5.2.2.1 Abbreviations of Stereotypes

Steretype Abbreviation	Full Stereotype Name
bTransactionView	BusinessTransactionView
AuthorizedRole	AuthorizedRole
bTransactionUC	BusinessTransactionUseCase
bTransaction	BusinessTransaction
bTPartition	BusinessTransactionPartition
bInformation	BusinessInformation
ReqInfPin	RequestingInformationPin
ResInfPin	RespondingInformationPin
InfPin	InformationPin
BusinessAction	BusinessAction
ReqAction	RequestingBusinessAction
ResAction	RespondingBusinessAction
bESharedState	SharedBusinessEntityState

5.2.2.2 Conceptual Description (informtive)

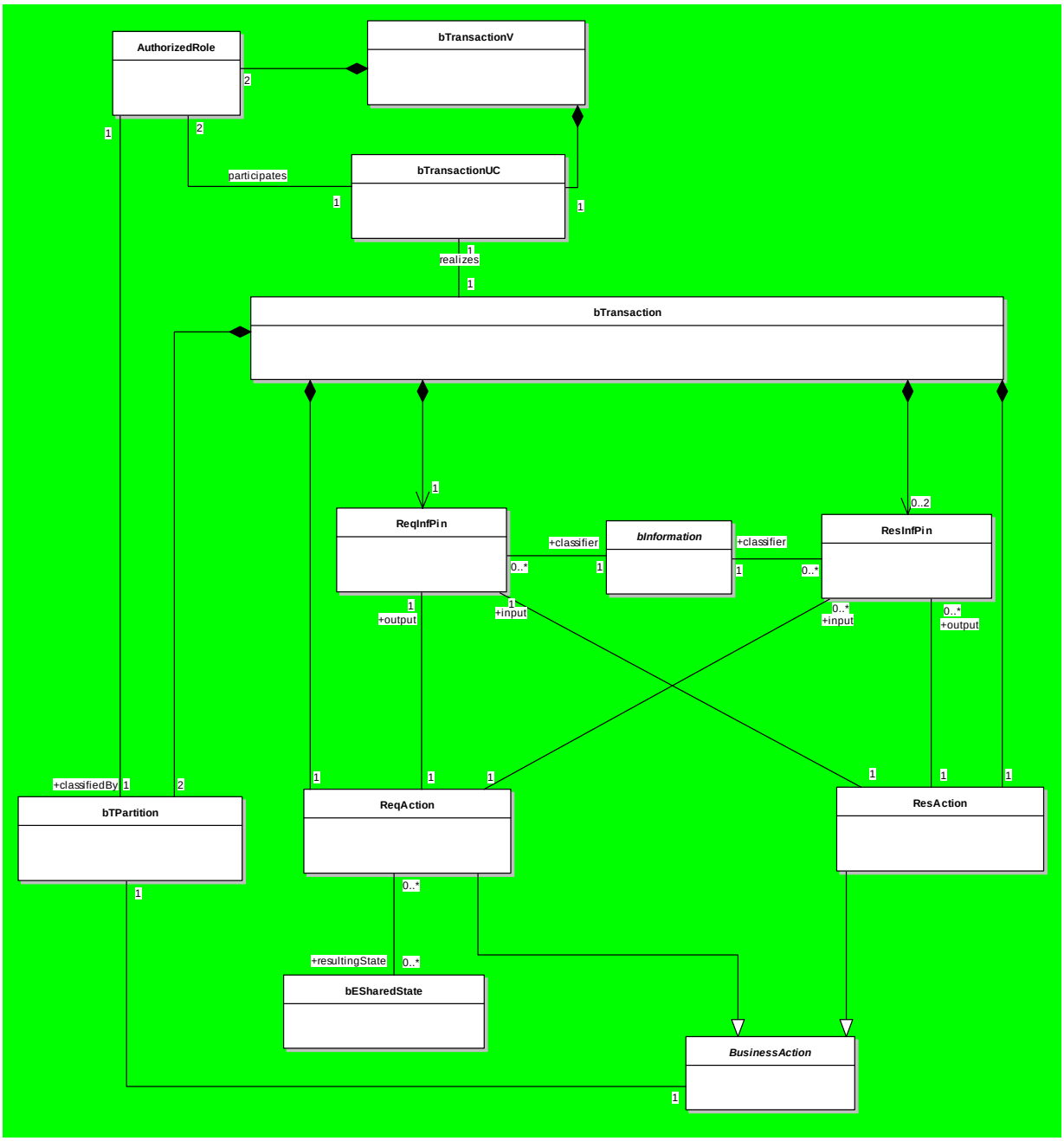


Figure 26 Business Transaction View – Conceptual Overview

Each *BusinessTransactionView* defines exactly one message exchange that leads to a synchronized business state between the two authorized roles executing it. The flow of messages is specified by the concept of a *BusinessTransaction*. The requirements of a *BusinessTransaction* are captured by a *BusinessTransactionUseCase*.

Each business transaction and its corresponding business transaction use case are defined in their own business transaction view package. Accordingly, the business transaction view is composed of exactly one *BusinessTransactionUseCase* and one *BusinessTransaction*.

5.2.2.2.1 Business Transaction Use Case Diagram

Two authorized roles participate in a business transaction use case. These authorized roles must be defined in the same business transaction view package as the corresponding business transaction use case. Accordingly, a *BusinessTransactionView* is composed of exactly two *AuthorizedRoles*. This means, if a certain role (e.g. buyer, seller, etc.) participates in multiple business transactions, it requires a different authorized role for each business transaction use case. Each authorized role of the same role (i.e., with the same name) is in a different namespace of a corresponding business transaction view. Therefore, an authorized role participates in only one business transaction use case – it is the one in the same business transaction view. Accordingly, *BusinessTransactionUseCase* and *AuthorizedRole* are related by a 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.

5.2.2.2.2 Business Transaction Diagram

A *BusinessTransaction* choreographs the synchronization of business states and the required information exchange between two authorized roles. The business transaction follows exactly the requirements defined in the corresponding business transaction use case. The business transaction that describes the business transaction use case is defined as a child beneath. Accordingly, each *BusinessTransactionUseCase* has exactly one *BusinessTransaction* beneath. A business transaction is a "composite" UML *Activity*. The graph of a business transaction is described by a flow of UML *Actions*.

A business transaction is an atomic step in a collaborative 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 *BusinessTransactionPartitions*. Each *BusinessTransactionPartition* relates to one *AuthorizedRole*. An *Authorized Role* is assigned to exactly one *BusinessTransactionPartition*. It follows, that the two partitions 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 action and the responding authorized role performs a responding business action. Each business action – no matter whether requesting or responding business action – is assigned to a swimlane, and each swimlane comprises exactly one business action. It follows that a *BusinessTransaction* is composed of exactly one *RequestingBusinessAction* and exactly one *RespondingBusinessAction*. Both, *RequestingBusinessAction* and *RespondingBusinessAction* are specializations of the abstract type *BusinessAction*. A *BusinessAction* is assigned to one *BusinessTransactionPartition*, and a *BusinessTransactionPartition* comprises one *BusinessAction*. Since a partition 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 partition.

The requesting business action outputs the requesting information through the requesting information pin that is input to the responding business action's requesting information pin. Business information created by the responding business action and returned to the requesting business action is optional. If business

information is returned by the responding business action zero to many responding information pins might be specified. Multiple responding information pins may be used to describe different business intentions (e.g., a positive and a negative response to a purchase order).

It follows, that a *BusinessTransaction* is composed of exactly two *RequestingInformationPins* and zero to many *RespondingInformationPins*. Both *RequestingInformationPin* and *RespondingInformationPin* are instances of the type *BusinessInformation*. A *RequestingBusinessAction* has exactly one *RequestingInformationPin and* zero to many *RespondingInformationPins.*

A *RespondingBusinessAction* has exactly one *RequestingInformationPin and* zero to many *RespondingInformationPins.*

RequestingInformationPin and *RespondingInformationPin* are stereotypes of the UML base class *Pin.* The type of the *Pin* is defined by the *BusinessInformation* that is a stereotype of the UML base class *Class*. According to UML, multiple *Pins* might be instances of the same *Class*. It follows that different requesting or responding information pins might be instances of the same business information. In other words, business information might be reused in different business transactions. Action pins that specify output information from a business action MUST be stereotyped and classified accordingly, whereas action pins that specify input information to a business action MAY not be stereotyped and classified.

If multiple responding information pins are defined, those must be in an XOR relationship with each other. In order to specify an XOR relationship between multiple incoming or outgoing responding information pins, each of them has to be enclosed by an UML *ParameterSet* (c.f. Figure 32). If only one responding information pin is defined within a business transaction, ParameterSets SHOULD not be used.

In order to determine the outcome of a business transaction (success or failure) the contents of the responding business document SHOULD be evaluated. OCL constraints SHOULD be used for assessing the document's content. An OCL constraint may either check the responding business information's type (e.g., positive or negative response to a quote) or directly investigate the document's content (e.g., if products were quoted or not). If the responding business information is checked, the constraints MUST be applied as condition guards to the transitions leading into the respective final states (e.g., success or response) of the business transaction. If the business transaction does not include a response, OCL constraints MAY not be used.

A business transaction synchronizes the states between the two authorized roles executing it. Thus, the execution of a business transaction results in a certain business entity shared state (the concept of business entity shared states have already been introduced in section 5.1.1.2). In order to point out the state change, setting the resulting shared state of a business entity might be visualized on the diagram of a business transaction. A *SharedBusinessEntityState* MAY be included as a predecessor of a final state to indicate the resulting synchronized state. The example in Figure 33 illustrates this concept.

5.2.2.3 Stereotypes and Tag Definitions (normative)

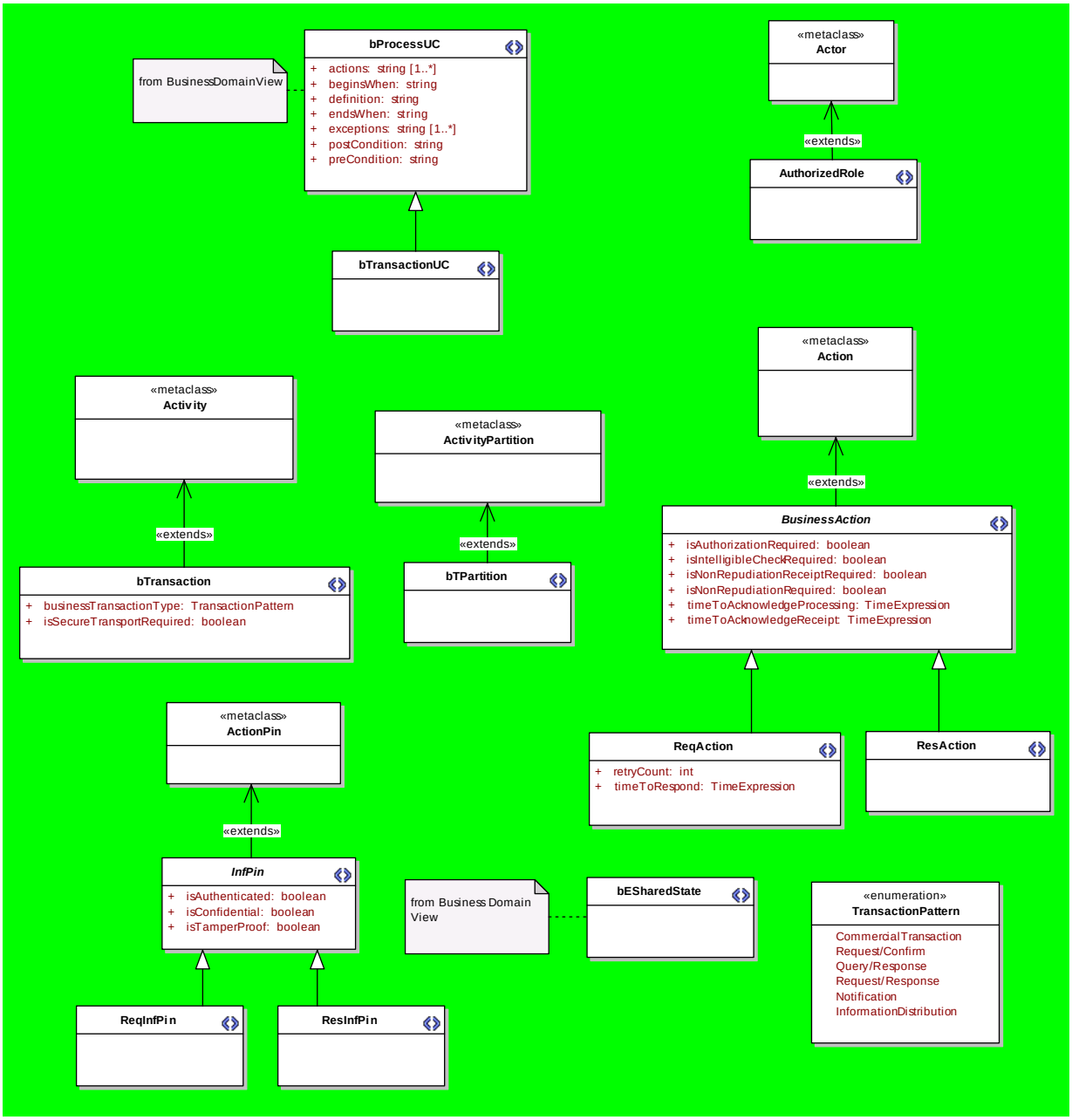


Figure 27 Business Transaction View – Abstract Syntax

Stereotype	bTransactionUC (BusinessTransactionUseCase)
Base Class	UseCase
Parent	bProcessUC
Description	A business transaction use case describes in detail the requirements on a collaboration between exactly two involved partners. A business transaction use case cannot be further refined and describes the requirements on a one-way or two-way information exchange. Business partners take part in a business transaction use case by playing an authorized role in it.
Tag Definition	**Inherited tagged values:**

	- definition - beginsWhen - preCondition - endsWhen - postCondition - exceptions - actions

740

Stereotype	AuthorizedRole (AuthorizedRole)
Base Class	Actor
Parent	N/A
Description	An authorized role (e.g. a "buyer") is a concept which is more generic than a business partner (e.g. a "wholesaler") and allows the reuse of collaborations by mapping an *AuthorizedRole* to a business partner 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 partners (a "wholesaler" or a "broker") in different scenarios of the same domain or even in different domains.
Tag Definition	No tagged values.

741

Stereotype	bTransaction (BusinessTransaction)
Base Class	Activity
Parent	N/A
Description	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 backwards to set the final and irreversible state change. In a business context irreversible means that returning to an original state requires another – compensating – 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 compensating the before sent purchase order. 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. Up to two object flows in the reverse direction are optional. If two object flows are defined as

	response, those are in an XOR relationship with each other (e.g., a purchase order is accepted or declined). According to the business transaction semantics, the requesting business activity does not end after sending the business information - it is still alive. The responding business activity may output the response which is returned to the still living requesting business activity.	
Tag Definition	**businessTransactionType**	
	Type	Enumeration: • Commercial Transaction • Request/Confirm • Query/Response • Request/Response • Notification • Information Distribution
	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
	isSecureTransportRequired	
	Type	Boolean
	Multiplicity	1
	Description	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.

742

Stereotype	**bTPartition (BusinessTransactionPartition)**
Base Class	Partition
Parent	N/A
Description	A business transaction partition is used to define an area of responsibility. An authorized role is appointed to a business transaction swimlane to indicate that this authorized role takes on the responsibility for the business action that is allocated within that area.
Tag Definition	**No Tagged Values**

743

Stereotype	**BusinessAction (BusinessAction, abstract)**	
Base Class	Action	
Parent	N/A	
Description	A 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 action or a responding business action.	
Tag Definition	**isAuthorizationRequired**	
	Type	**Boolean**
	Multiplicity	**1**
	Description	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.
	isNonRepudiationRequired	
	Type	**Boolean**
	Multiplicity	**1**

	Description	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.
	isNonRepudiationReceiptRequired	
	Type	**Boolean**
	Multiplicity	**1**
	Description	The *isNonRepudiationOfReceiptRequired* tag requires the receiver of a business information 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.
	timeToAcknowledge Receipt	
	Type	**TimeExpression**
	Multiplicity	**1**
	Description	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 is sent by a sender until the time a verification of receipt is “properly received” by the sender (of the business information). Accordingly, the time to acknowledge receipt is always specified by the sender’s business action. 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).
	timeToAcknowledgeProcessing	
	Type	**TimeExpression**
	Multiplicity	**1**
	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 affirmant may be one of the two authorized roles. Thus, we use again the terms sender and affirmant 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 acknowledgement of processing is "properly received" by the sender (of the business information). Accordingly, the time to acknowledge processing is always specified by the sender's business action. An affirmant 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 affirmant does not acknowledge processing of business information within the agreed time period.
	isIntelligibleCheckRequired	
	Type	**Boolean**
	Multiplicity	**1**
	Description	In order to define the *isIntelligibleCheckRequired* semantics, we use again the terms sender and affirmant as introduced for the last two tag definitions. Both partners may agree that an affirmant 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.

744

Stereotype	**ReqAction (RequestingBusinessAction)**	
Base Class	Action	
Parent	BusinessAction	
Description	A requesting business action is a business action that is performed by an authorized role requesting business service from another authorized role.	
Tag Definition	**timeToRespond**	
	Type	TimeExpression
	Multiplicity	1
	Description	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.
retryCount	
Type	Integer
Multiplicity	1
Description	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 – i.e., failed business control exceptions.

Inherited tagged values:

- isAuthorizationRequired
- isNonRepudiationRequired
- isNonRepudiationReceiptRequired
- timeToAcknowledgeReceipt
- timeToAcknowledgeProcessing
- isIntelligibleCheckRequired

Default assignment of tagged values for the requesting business action:

	Time to Acknowledge Receipt	Time to Acknowledge Processing	Time to Respond	Is Authorization Required	Is Non Repudiation Required	Is Non Repudiation of Receipt Required	Retry Count	Is Intelligible Check Required
Commercial Transaction	2h	6h	24h	TRUE	TRUE	TRUE	3	TRUE
Request/Confirm	NULL	NULL	24h	FALSE	FALSE	FALSE	3	TRUE
Request/Response	NULL	NULL	4h	FALSE	FALSE	FALSE	3	TRUE
Query/Response	NULL	NULL	4h	FALSE	FALSE	FALSE	3	TRUE
Notification	24h	NULL	NULL	FALSE	TRUE	TRUE	3	TRUE
Information Distribution	NULL	NULL	NULL	FALSE	FALSE	FALSE	0	TRUE

745

746

Stereotype	ResAction (RespondingBusinessAction)
Base Class	Action
Parent	Business Action
Description	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.
Tag Definition	**Inherited tagged values:** - isAuthorizationRequired - isNonRepudiationRequired - isNonRepudiationReceiptRequired - timeToAcknowledgeReceipt - timeToAcknowledgeProcessing - isIntelligibleCheckRequired Default assignment of tagged values for the responding business action:

	Time to Acknowledge Receipt	Time to Acknowledge Processing	Is Authorization Required	Is Non Repudiation Required	Is Non Repudiation of Receipt Required	Is Intelligible Check Required
Commercial Transaction	2h	6hr	TRUE	TRUE	TRUE	TRUE
Request/Confirm	2h	NULL	TRUE	FALSE	TRUE	TRUE
Request/Response	NULL	NULL	FALSE	FALSE	FALSE	TRUE
Query/Response	NULL	NULL	FALSE	FALSE	FALSE	TRUE
Notification	NULL	NULL	FALSE	FALSE	FALSE	TRUE
Information Distribution	NULL	NULL	FALSE	FALSE	FALSE	TRUE

747

Stereotype	InfPin (InformationPin, abstract)
Base Class	Pin
Parent	N/A
Description	The abstract concept information pin represents the incoming/outgoing point for business information in a business action. Business information is sent from the requesting authorized role to the responding authorized role or the reverse way. The actual exchanged information is represented using the type

	business information. Both concrete stereotypes requesting information pin and responding information pin inherit from the abstract stereotype information pin.
Tag Definition	**isConfidential**
	Type: Boolean
	Multiplicity: 1
	Description: If the flag is set, the exchanged information is encrypted so that unauthorized parties cannot view the information**.**
	isTamperProof
	Type: Boolean
	Multiplicity: 1
	Description: If the flag is set, the exchanged information 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.
	isAuthenticated
	Type: Boolean
	Multiplicity: 1
	Description: If the flag is set, there is a digital certificate associated with the document entity. This provides proof of the signer's identity.

748

Stereotype	**ReqInfPin (RequestingInformationPin)**
Base Class	Pin
Parent	InformationPin
Description	The requesting information pin is a container for 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 pin does not mean that the exchanged business information refers to a request in a business sense. The term requesting information pin 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.
Tag Definition	**Inherited tagged values:** - isConfidential - isAuthenticated - isTamperProof

749

Stereotype	**ResInfPin (RespondingInformationPin)**
Base Class	Pin

Parent	InformationPin
Description	The responding information pin 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).
Tag Definition	**Inherited tagged values:** - isConfidential - isAuthenticated - isTamperProof

5.2.2.4 Constraints (normative)

C.34. A *BusinessTransactionView* MUST contain exactly one *BusinessTransactionUseCase*, exactly two *AuthorizedRoles*, and exactly two *participates* associations.

C.35. A *BusinessTransactionUseCase* MUST be associated with exactly two *AuthorizedRoles* via stereotyped binary *participates* associations.

C.36. A *BusinessTransactionUseCase* MUST NOT include further *UseCases*

C.37. A *BusinessTransactionUseCase* MUST be included in at least one *BusinessCollaborationUseCase.*

C.38. A *BusinessTransactionUseCase* MUST NOT be source or target of an extend association.

C.39. The two *AuthorizedRoles* within a *BusinessTransactionView* MUST NOT be named identically

C.40. Exactly one *BusinessTransaction* MUST be placed beneath each *BusinessTransactionUseCase.* This relationship MAY also be visualized by a realize relationship from the *BusinessTransaction* to the *BusinessTransactionUseCase.*

C.41. A *BusinessTransaction* MUST have exactly two partitions. Each of them MUST be stereotyped as *BusinessTransactionPartition*.

C.42. One of the two *BusinessTransactionPartitions* MUST contain one *RequestingBusinessAction* and the other one MUST contain one *RespondingBusinessAction.*

C.43. A *BusinessTransactionPartition* MUST have a classifier, which MUST be one of the associated *AuthorizedRoles* of the corresponding *BusinessTransactionUseCase.*

C.44. The two *BusinessTransactionPartitions* MUST have different classifiers.

C.45. The *BusinessTransactionPartition* containing the *RequestingBusinessAction* MUST contain two or more *FinalStates*. Each of the *FinalStates* MAY have a *SharedBusinessEntityState* as predecessor. One of the *FinalStates* SHOULD reflect a *ControlFailure* – this *FinalState* SHOULD NOT have a predecessing *SharedBusinessEntityState*.

C.46. A *RequestingBusinessAction* MUST embed exactly one *RequestingInformationPin*

C.47. A *RespondingBusinessAction* MUST embed exactly one *RequestingInformationPin*

C.48. If the tagged value *businessTransactionType* of the *BusinessTransaction* is either *Request/Response*, *Query/Response, Request/Confirm, or CommercialTransaction*, then the *RequestingBusinessAction* MUST embed one to many *RespondingInformationPins* and the *RespondingBusinessAction* MUST embed one to many *RespondingInformationPins*.

C.49. If the tagged value *businessTransactionType* of the *BusinessTransaction* is either *Notification* or *InformationDistribution*, then both, the *RequestingBusinessAction* and the *RespondingBusinessAction*, MUST NOT embed a *RespondingInformationPin*

C.50. A RequestingBusinessAction and a RespondingBusinessAction MUST embed same number of RespondingInformationPins.

C.51. The *RequestingInformationPin* of the *RequestingBusinessAction* MUST be connected with the *RequestingInformationPin* of the *RespondingBusinessAction* using an object flow relationship leading from the *RequestingBusinessAction* to the *RespondingBusinessAction*.

C.52. Each RespondingInformationPin of the RespondingBusinessAction MUST be connected with exactly one RespondingInformationPin of the RequestingBusinessAction using an object flow relationship leading from the *RespondingBusinessAction* to the *RequestingBusinessAction*

C.53. If a *BusinessTransactionPartition* contains *SharedBusinessEntityStates*, each *SharedBusinessEntityState* MUST be the target of exactly one control flow relationship starting from the *RequestingBusinessAction* and MUST be the source of exactly one control flow relationship targeting a FinalState.

C.54. Each *FinalState* MUST be the target of one to many control flow relationships starting from the *RequestingBusinessAction* or from a *SharedBusinessEntityState*.

C.55. Each *RequestingInformationPin* and each *RespondingInformationPin* MUST have a classifier, this classifier MUST be an *InformationEnvelope* or a subtype defined in an extension/specialization module.

C.56. Two RequestingInformationPins which are connected using an object flow MUST have the same classifier.

C.57. Two RespondingInformationPins which are connected using an object flow MUST have the same classifier.

5.2.2.5 Worksheets

Form for Business Transaction Use Case	
General	
Name	
Description	
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Details	

Requesting Role	
Responding Role	
Requesting Activity	
Responding Activity	
Is Included In (Name of Business Collaboration)	
Start/End Characteristics	
Affected Business Entities	
Pre-condition	
Post-condition	
Begins When	
Ends When	
Exceptions	

806

Form for Business Transaction	
General	
Name	
Description	
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Details	

Select Business Transaction Pattern	☐ Information Distribution ☐ Notification ☐ RequestResponse ☐ RequestConfirm ☑ QueryResponse ☐ Commercial Transaction
Secure Transport	
Requestor's Side	
Requesting Role	
Requesting Business Action Name	
Time to Respond	
Time to Acknowledge Receipt	
Time to Acknowledge Processing	
Authorization Required	
Non Repudiation Required	
Non Repudiation of Receipt Required	
Intelligible Check Required	
Number of Retries	
Responder's Side	
Responding Role	
Responding Business Action Name	
Time to Acknowledge Receipt	
Time to Acknowledge Processing	

Authorization Required		
Non Repudiation Required		
Non Repudiation of Receipt Required		
Intelligible Check Required		
Business Information Envelopes		
Requesting Information Envelope		
	Name	
	Are Contents Confidential?	
	Is the Envelope Tamperproof?	
	Authentication Required?	
Responding Information Envelope (add more Responding Information Envelopes if different response documents are possible)		
	Name	
	Resulting Business Entity State (including transition condition)	
	Are Contents Confidential?	
	Is the Envelope Tamperproof?	
	Authentication Required?	

807

5.2.2.6 *Example (informative)*

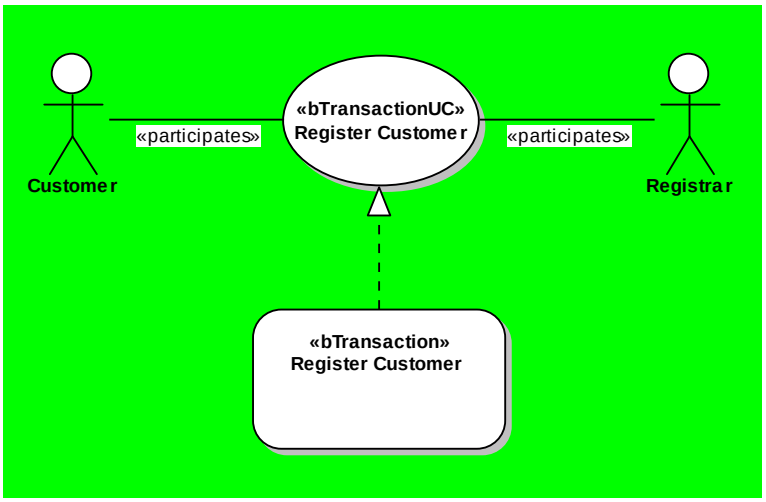


Figure 28 Business Transaction Use Case Example: Register Customer (including the optional realize relationship to the business transaction placed beneath)

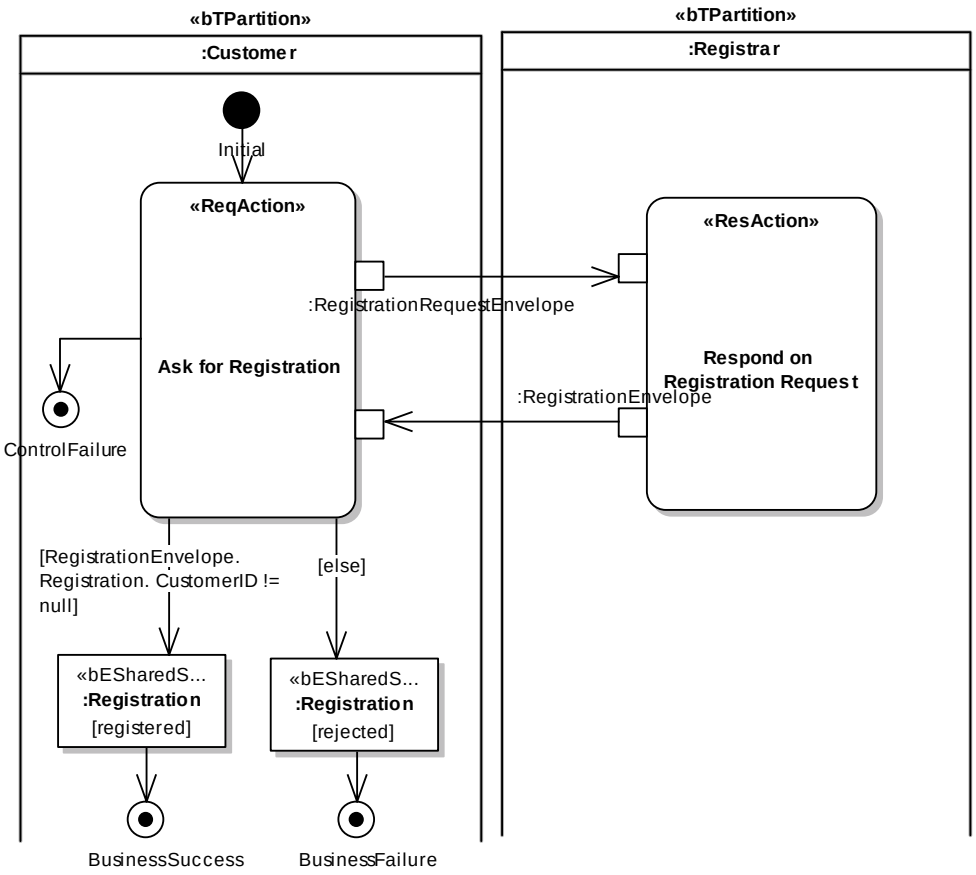
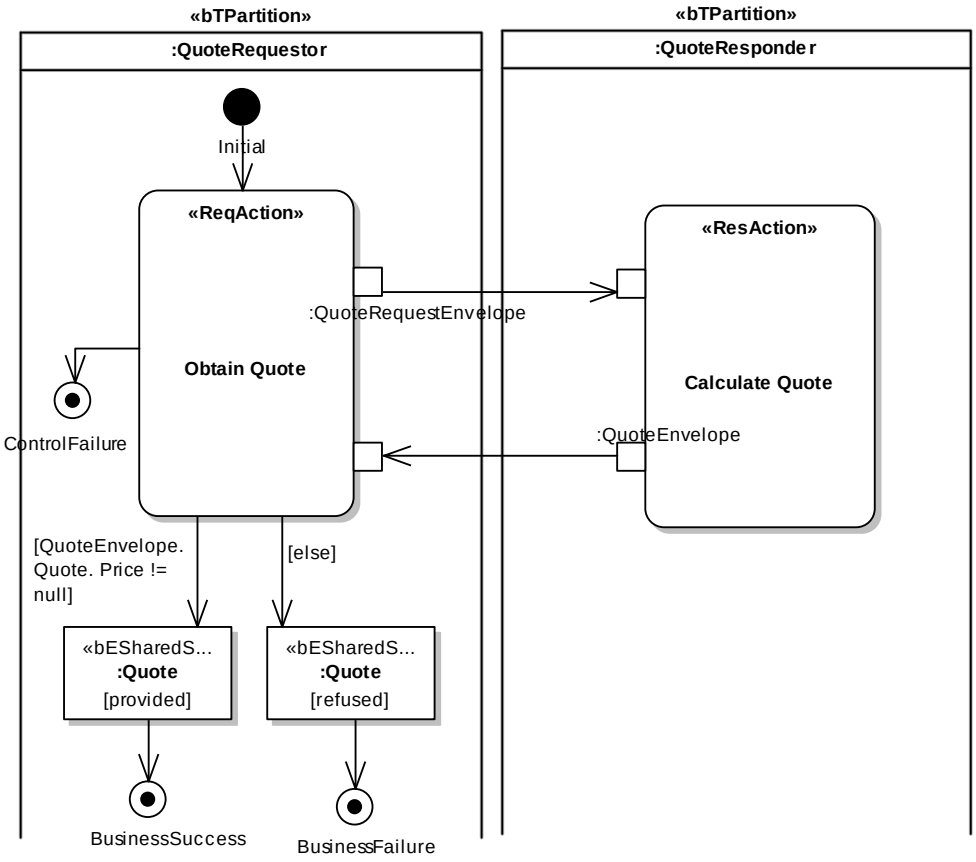


Figure 29 Business Transaction Example: Register Customer

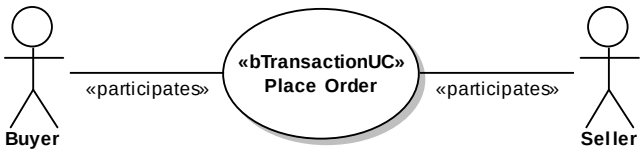
815

816 **Figure 30 Business Transaction Use Case Example: Request For Quote (the optional realize relationship is not shown)**



817

818 **Figure 31 Business Transaction Example: Request For Quote**



819

820 **Figure 32 Business Transaction Use Case Example: Place Order (the optional realize relationship is not shown)**

821

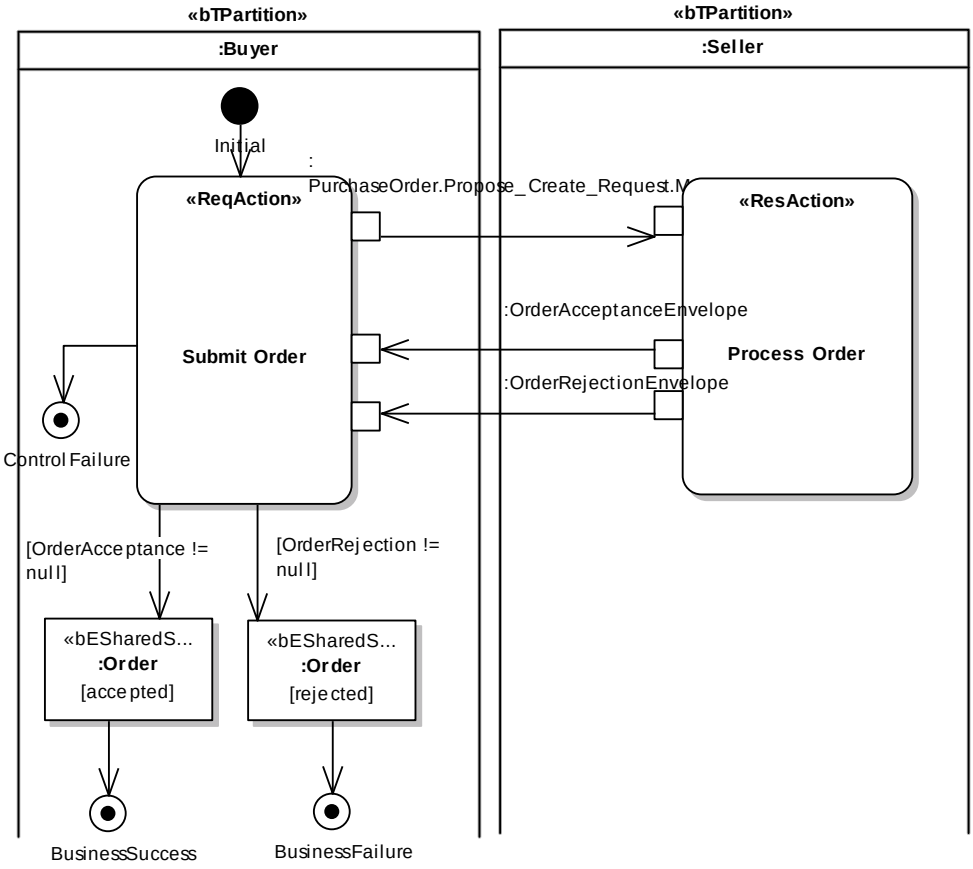


Figure 33 Business Transaction Example: Place Order

5.2.3 Business Collaboration View

5.2.3.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bCollaborationV	BusinessCollaborationView
AuthorizedRole	AuthorizedRole
bCollaborationUC	BusinessCollaborationUseCase
bCollaborationProtocol	BusinessCollaborationProtocol
bCPartion	BusinessCollaborationPartition
bTransactionAction	BusinessTransactionAction
bCollaborationAction	BusinessCollaborationAction
bNestedCollaboration	NestedBusinessCollaboration

5.2.3.2 Conceptual Description (informative)

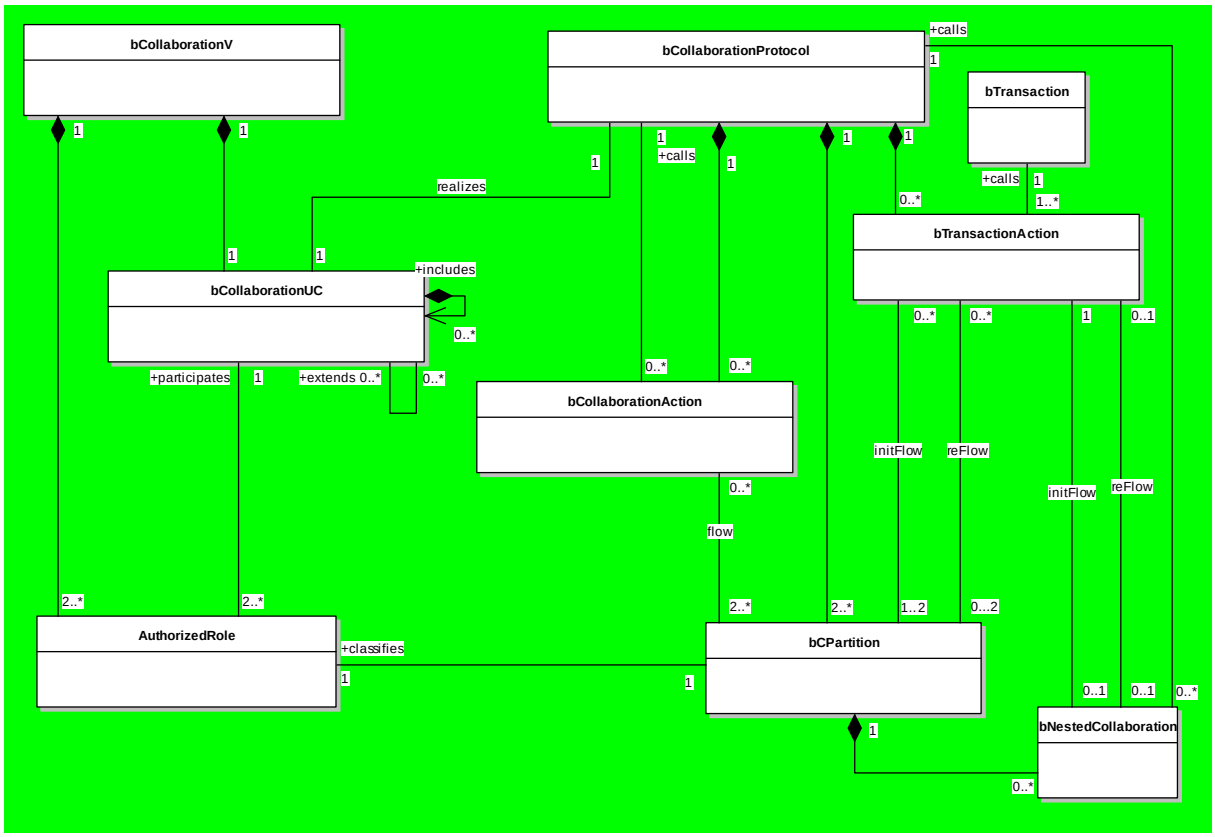


Figure 34 Business Collaboration View - Conceptual Overview

A B*usinessCollaborationView* is used to define the business choreography of exactly one business collaboration. This business choreography is specified by the concept of a *BusinessCollaborationProtocol*. The requirements of a *BusinessCollaborationProtocol* are captured by a *BusinessCollaborationUseCase*.

Each *BusinessCollaborationUseCase* and its corresponding BusinessCollaborationProtocol are defined in their own business collaboration view package. Accordingly, the *BusinessCollaborationView* is composed of exactly one *BusinessCollaborationUseCase* and one *BusinessCollaborationProtocol*.

5.2.3.2.1 Business Collaboration Use Case Diagram

At least two authorized roles participate in a business collaboration use case. These authorized roles must be defined in the same business collaboration view package as the corresponding business collaboration use case. Accordingly, a *BusinessCollaborationView* is composed of two or more *AuthorizedRoles*. This means, if a certain role (e.g. buyer, seller, etc.) participates in multiple business collaborations, it requires a different authorized role for each business collaboration use case. Each authorized role of the same role (i.e., with the same name) is in a different namespace of a corresponding business collaboration view. Therefore, an authorized role participates in only one business collaboration use case – it is the one in the same business collaboration view. Accordingly, *BusinessCollaborationUseCase* and *AuthorizedRole* are related by a 1 to (2..n) association. This association is defined as a *participates* association. It is important to note, that the same authorized role is not associated twice to the same business collaboration use case.

A business collaboration use case may include additional business collaboration use cases. A business collaboration use case may optionally have multiple parent business collaboration use cases. Hence, *BusinesCollaborationUseCase* 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 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.

A business collaboration use case may be extended by additional business collaboration use cases. A business collaboration use case may optionally extend multiple business collaboration use cases. Hence, *BusinesCollaborationUseCase* has a unary (0..n) to (0..n) extend-association.

5.2.3.2.2 Business Collaboration Protocol Diagram

A business choreography view is used to define the business choreography of exactly one business collaboration. The BusinessCollaborationProtocol follows exactly the requirements defined by the corresponding *BusinessCollaborationUseCase.* The business collaboration protocol that describes the business collaboration use case is defined as a child beneath. Accordingly, each *BusinessCollaborationUseCase* has exactly one *BusinessCollaborationProtocol* beneath. A business collaboration protocol is a "composite" UML *Activity*. The diagram of a business collaboration protocol is described by a flow of UML *Actions*.

A business collaboration protocol defines the collaborative business process between two or more authorized roles. The business collaboration protocol must have a *BusinessCollaborationPartiton* for each of the authorized roles defined in the *BusinessCollaborationView*. Hence, a *BusinessCollaborationProtocol* is composed of two or more *BusinessCollaborationPartitions*. Each *BusinessCollaborationPartition* relates to one and only one *AuthorizedRole* defined in the *BusinessCollaborationView*. Each *AuthorizedRole* in the *BusinessCollaborationView* is assigned to exactly one *BusinessCollaborationPartition*. It follows, that the every BusinessCollaborationPartition of a BusinessCollaborationProtocol must be assigned to different authorized roles.

The collaborative actions of a business collaboration protocol are business collaboration actions and/or business transaction actions. Hence, a *BusinessCollaborationProtocol* is composed of zero to many *BusinessCollaborationActions* and of zero to many *BusinessTransactionActions*. However, at least one business collaboration action or at least one business transaction action must be present in a business 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 action is characterized by the fact that it is executed by calling a business collaboration protocol. This calling behavior is accomplished by classifying the behavior of the business collaboration action by the desired business collaboration protocol. Not every business collaboration protocol is a called by a business collaboration action. A business collaboration protocol may be called by multiple business collaboration actions. Thus, the behavioral classifying relationship between *BusinessCollabortionAction* and *BusinessCollaborationProtocol* is (0..n) to 1.

A business transaction action is characterized by the fact that it is executed by calling a business transaction. This calling behavior is accomplished by classifying the behavior of the business transaction action by the desired business transaction. Since the business transaction is a concept of the business transaction view it is described in more detail above. Each business transaction must be at least once used to refine a business transaction action. A business transaction may be called by many business transaction actions. Hence, the behavioral classifying relationship between *BusinessTransactionAction* and *BusinessTransaction* is (1..n) to 1.

In many scenarios, there is a requirement for a nested business collaboration within the scope of execution of a given business transaction action. In other words, before a responding authorized role can send a response as required by the business transaction action that calls a two-way business transaction, the responding authorized role has to first participate in a business collaboration with other business partners. UMM supports this scenario by introducing the concept of a *NestedBusinessCollaboration.* Like the business collaboration action, the NestedBusinessCollaboration is characterized by the fact that it is executed by calling another business collaboration protocol. This calling behavior is accomplished by classifying the behavior of the *NestedBusinessCollaboration* by the desired business collaboration protocol. Not every business collaboration protocol is a called by a NestedBusinessCollaboration. A business collaboration protocol may be called by multiple NestedBusinessCollaborations. Thus, the behavioral classifying relationship between *NestedBusinessCollaboration* and *BusinessCollaborationProtocol* is (0..n) to 1. However, unlike the business collaboration action, the NestedBusinessCollaboration must reside within a business collaboration partition representing the responding authorized role of a given business transaction action. Accordingly, not every business collaboration partition representing a business transaction action responding role includes a NestedBusinessCollaboration. Thus, there is a 1 to (0..n) composition between a *BusinessCollaborationPartiton* and a *NestedBusinessCollaboration*.

In UMM 2.0, role mapping between business collaboration authorized roles and either called business transaction authorized roles or business collaboration protocol authorized roles is defined in the business collaboration protocol and no longer by the business collaboration use case. This role mapping is accomplished by information flows and specializations of information flows, i.e. *InitiatingFlow* and *RespondingFlow*, between either business collaboration partitions or nested collaborations and either business collaboration actions or business transaction actions. Using the approach also enhances the business collaboration protocol by graphically illustrating the relationships between authorized roles and the choreography of actions within a business collaboration protocol. This mapping approach is defined by the following cases:

1. *From business collaboration partitions to business collaboration actions.* The UML 2.0 Information flow is used. The source of Information flow must be the business collaboration partition and the target must be the business collaboration action. This means that the authorized role in this business collaboration protocol is participating in the called business collaboration protocol. Therefore the calling business collaboration protocol must have at least the same number of business collaboration partitions as the number of business collaboration partitions in the called business collaboration protocol. There are two cases for defining a role mapping:

    - The authorized role name of the calling business collaboration protocol maps to an authorized role in the called business collaboration protocol which has exactly the same name. In this case, the information flow association explicitly defines the role mapping between these two roles.

- The authorized role name of the calling business collaboration protocol maps to an authorized role in the called business collaboration protocol which has a different name. In this case, the information flow association must be classified by the authorized role of the called business collaboration protocol.

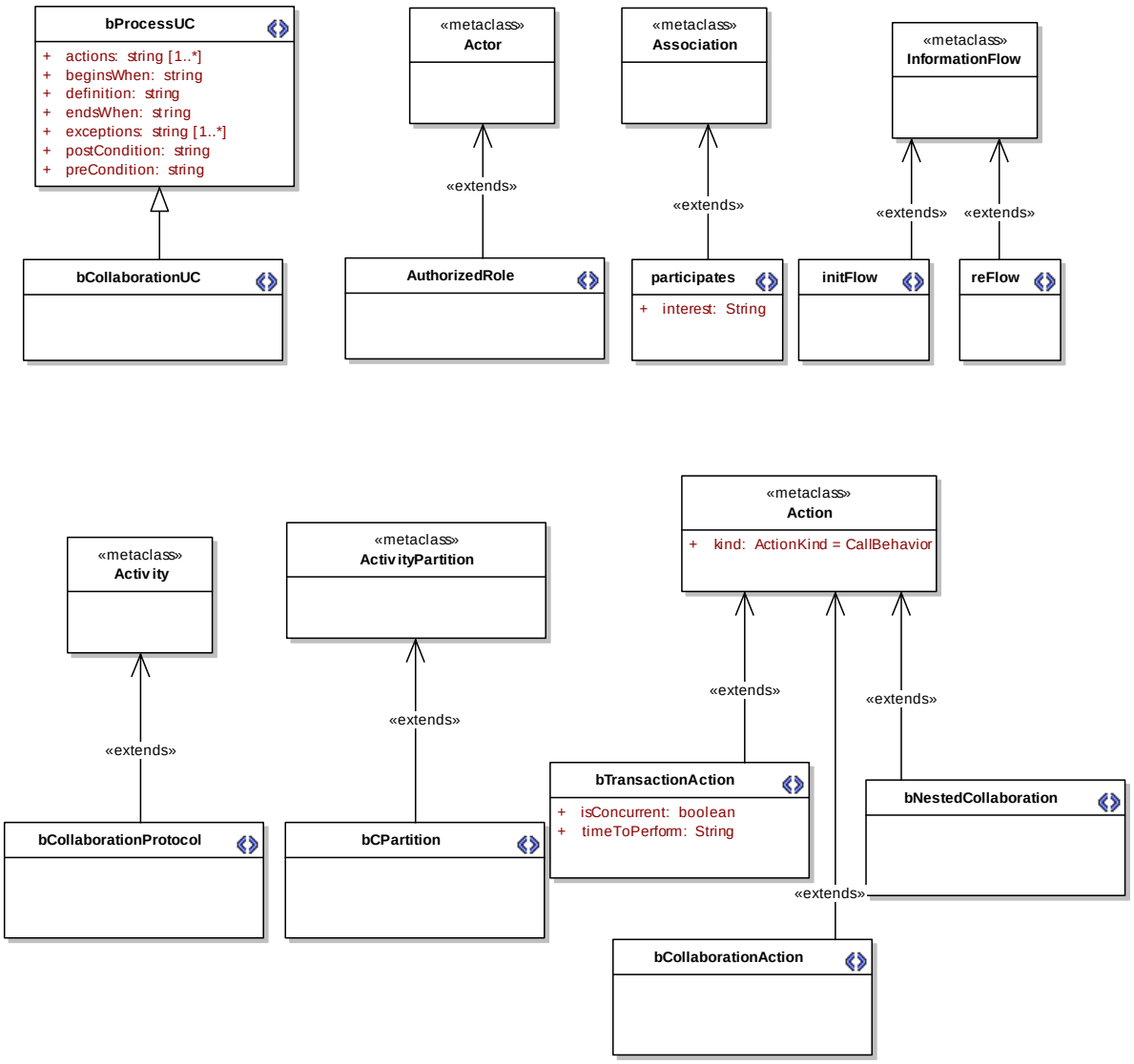
2. *From business collaboration partitions to business transaction actions.* In this case two specializations of the Information flow are used, *InitiatingFlow and RespondingFlow.* In all cases an authorized role of a business collaboration protocol initiates a business transaction action, which calls a business transaction, which also has an initiating authorized role. To provide this role mapping the business collaboration partition classified by an authorized role is the source of the *InitiatingFlow* and the business transaction action is the target. Likewise for two-way business transactions, a business collaboration partition is the source of the *RespondingFlow* and the business transaction action is the target.

3. *From business transaction actions to business collaboration partitions.* In this case two specializations of the Information Flow are used, *InitiatingFlow and RespondingFlow.* In all cases an authorized role of a business collaboration protocol responds to a business transaction action which calls a business transaction which also has a responding authorized role. To provide this role mapping the business transaction action is the source of an InitiatingFlow and a business collaboration partition is the target. Therefore the authorized role that classifies the business collaboration partition maps to the responding business transaction authorized role. Likewise, for two-way business transactions, a business transaction action is the source of a RespondingFlow and a business collaboration partition is the target.

4. *From business transaction actions to nested business collaborations.* In this case only one specialization of the Information Flow is used, *InitiatingFlow.* The source of the InitiatingFlow is a business transaction action and the target is a nested business collaboration. This means that the responding authorized role of the business collaboration initiates the business collaboration protocol called by the nested business collaboration and therefore maps to initiating authorized role of the called business collaboration protocol.

5. *From nested business collaborations to business transaction actions*. In this case only one specialization of the Information Flow is used, *RespondingFlow* and only applies in the case of two-way business transactions. The *RespondingFlow* indicates that the called business collaboration has completed and the responding authorized role can now return the response envelope to the initiating authorized role. The *NestedBusinessCollaboration* is the source of the *RespondingFlow* and a business transaction action is the target.

965 ### *5.2.3.3 Stereotypes and Tag Definitions (normative)*



966

967 **Figure 35 Business Collaboration View - Abstract Syntax**

968

Stereotype	bCollaborationUC (BusinessCollaborationUseCase)
Base Class	UseCase
Parent	bProcessUC (Business Process Use Case)
Description	A business collaboration use case describes in detail the requirements on the 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. A business collaboration use case may extend another business collaboration use case.
Tag Definition	**Inherited tagged values:**

	- definition - beginsWhen - preCondition - endsWhen - postCondition - exceptions - actions

969

Stereotype	AuthorizedRole
Base Class	Actor
Parent	N/A
Description	Already defined before in previous sub-section
Tag Definition	No tagged values.

970

Stereotype	bCollaborationProtocol (BusinessCollaborationProtocol)
Base Class	Activity
Parent	N/A
Description	A business collaboration protocol choreographs business transaction actions and/or business collaboration actions. At least one action 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.
Tag Definition	**No Tagged Values**

971

Stereotype	bCPartition (BusinessCollaborationPartition)
Base Class	ActivityPartition
Parent	N/A
Description	A business collaboration partition is used to define an area of responsibility. The business collaboration partition is always classified by an authorized role defined as a participant in the corresponding business collaboration use case. A business collaboration partition may be empty. It is not empty in the special case of a nested collaboration. A nested collaboration must be placed within the business collaboration partition of the authorized role which is the responding authorized role in the triggering business transaction action and which will initiate the nested collaboration.
Tag Definition	No tagged values.

972

973

Stereotype	bTransactionAction (BusinessTransactionAction)

Base Class	Action with call behaviour action kind (CallBehaviorAction)
Parent	N/A
Description	A business transaction action is an action within a business collaboration protocol. This action is refined by a using the call behaviour to classify the behaviour of this business transaction action by one and only one business transaction. The business transaction action executes the called business transaction. The business transaction action can be executed more than once at the same time if the "isConcurrent" property is true.

Tag Definition		
	timeToPerform	
	Type	TimeExpression
	Multiplicity	1
	Description	A business transaction action 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 action, i.e. sending the requesting business information envelope, and the moment the requesting authorized role receives a substantive response. The substantive response is the responding business information envelope 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.
	isConcurrent	
	Type	Boolean
	Multiplicity	1
	Description	If the business transaction action is concurrent then more than one business transaction action 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 action is not concurrent then only one business transaction action of the same underlying business transaction can be open at one time.

974

Stereotype	**bCollaborationAction (BusinessCollaborationAction)**
Base Class	Action with call behaviour action kind (CallBehaviorAction)
Parent	N/A
Description	A business collaboration action is an action within a business collaboration protocol. This business collaboration action is refined by using the call behaviour to classify the behaviour of this business collaboration action by one and only one business collaboration protocol. The business choreography action executes the called business collaboration protocol exactly once. It follows, that business collaboration protocols might be recursively nested.
Tag Definition	**No Tagged Values**

975

Stereotype	**NestedBCollaboration (NestedBusinessCollaboration)**

Base Class	Action with call behaviour action kind (CallBehaviorAction)
Parent	N/A
Description	A nested business collaboration represents the case where the responding authorized role of a business transaction action after receiving a requesting information envelope which is represented as an initFlow (InitiatingFlow) must carry out an additional business collaboration protocol with other business partners before responding to the initiating authorized role of a given business transaction action indicated by a reFlow (RespondingFlow). This nested business collaboration is refined by using the call behaviour to classify the behaviour of this nested business collaboration by one and only one business collaboration protocol. The nested collaboration executes the called business collaboration protocol exactly once.
Tag Definition	No tagged values.

976

Stereotype	**initFlow (InitiatingFlow)**
Base Class	Information Flow
Parent	N/A
Description	The initiating flow represents the following two cases: • The initiating flow of information that triggers the execution of a business transaction action. The source of the initiating flow is the business collaboration partition that is classified by the authorized role initiating the business transaction action and the target is the business transaction action. In this case the initiating flow provides the authorized role mapping between the initiating role of the business transaction action and the initiating authorized role of the business transaction that is called by this business transaction action. • The initiating flow of information that triggers an execution on the responder's side. The source of this initiating flow is the business transaction action and the target is the business collaboration partition which is classified by the authorized role responding in the business transaction action. In this case the initiating flow provides the authorized role mapping between the responding role of the business transaction action and the responding authorized role of the business transaction that is called by this business transaction action. In the special case that the initiating flow triggers a nested business collaboration, the target of the initiating flow is not the business collaboration partition, but the nested business collaboration residing within this business collaboration partition.
Tag Definition	No tagged values.

977

Stereotype	**reFlow (RespondingFlow)**
Base Class	Information Flow
Parent	N/A
Description	The responding flow in case of two-way transactions represents the following two cases: • The responding flow of information that completes the execution of a business transaction action. The source of the responding flow is the business collaboration partition that is classified by the authorized role responding in the business transaction action and the target is the business transaction action. In the special case that the responding flow is started after a nested business collaboration has completed, the source of the responding flow is not the business

	collaboration partition, but the nested business collaboration residing within this business collaboration partition. • The responding flow of information that completes the business transaction action on the initiator's side. The source of this initiating flow is the business transaction action and the target is the business collaboration partition which is classified by the authorized role initiating the business transaction action.
Tag Definition	No tagged values.

5.2.3.4 Constraints (normative)

C.58. A *BusinessCollaborationView* MUST contain exactly one *BusinessCollaborationUseCase*.

C.59. A *BusinessCollaborationView* MUST contain two to many *AuthorizedRoles*.

C.60. A *BusinessCollaborationUseCase* MUST have two to many *participates* associations to *AuthorizedRoles* contained in the same *BusinessCollaborationView*.

C.61. Each *AuthorizedRole* contained in the *BusinessCollaborationView* MUST have exactly one *participates* association to the *BusinessCollaborationUseCase* included in the same *BusinessCollaborationView*.

C.62. A *BusinessCollaborationUseCase* MUST have one to many *include* relationships to another *BusinessCollaborationUseCase* or to a *BusinessTransactionUseCase*.

C.63. Exactly one *BusinessCollaborationProtocol* MUST be placed beneath each *BusinessCollaborationUseCase*. This relationship MAY also be visualized by a realize relationship from the *BusinessCollaborationProtocol* to the *BusinessCollaborationUseCase*.

C.64. A *BusinessCollaborationProtocol* MUST contain one to many *BusinessTransactionActions* and/or *BusinessCollaborationAction*.

C.65. Each *BusinessTransactionAction* MUST call exactly one *BusinessTransaction*

C.66. Each BusinessTransaction called by a *BusinessTransactionAction* MUST be placed beneath a BusinessTransactionUseCase which is included in the *BusinessCollaborationUseCase* that covers the corresponding *BusinessCollaborationProtocol*.

C.67. Each *BusinessCollaborationProtocol* called by a *BusinessCollaborationAction* MUST be placed beneath a *BusinessCollaborationProtocolUseCase* which is included in the *BusinessCollaborationUseCase* that covers the corresponding *BusinessCollaborationProtocol*.

C.68. A *BusinessCollaborationProtocol* MUST contain two to many *BusinessCollaborationPartions*.

C.69. The number of *AuthorizedRoles* in the *BusinessCollaborationView* MUST match the number of *BusinessCollaborationPartitions* in the *BusinessCollaborationProtocol* which is placed beneath the *BusinessCollaborationUseCase* of the same *BusinessCollaborationView*.

C.70. Each AuthorizedRole in the *BusinessCollaborationView* MUST be assigned to a *BusinessCollaborationPartition* in the *BusinessCollaborationProtocol* which is placed beneath the *BusinessCollaborationUseCase* of the same *BusinessCollaborationView*.

C.71. Each *BusinessCollaborationPartition* MUST be classified by exactly one AuthorizedRole included in the same *BusinessCollaborationView* as the *BusinessCollaborationUseCase* covering the *BusinessCollaborationProtocol* containing this *BusinessCollaborationPartition*.

C.72. A *BusinessCollaborationPartition* MUST be either empty or contain one to many *NestedBusinessCollaborations*.

C.73. Each *BusinessTransactionAction* MUST be the target of exactly one *InitialFlow* which source MUST be a *BusinessCollaborationPartition*.

C.74. Each *BusinessTransactionAction* MUST be the source of exactly one *InitialFlow* which target MUST be either a *BusinessCollaborationPartition* or a *NestedBusinessCollaboration*.

C.75. The *InitialFlow* sourcing from a *BusinessTransactionAction* and the *InitialFlow* targeting a *BusinessTransactionAction* MUST NOT be targeting to / sourcing from the same *BusinessCollaborationPartition*, nor targeting to a *NestedBusinessCollaboration* within the same *BusinessCollaborationPartition*.

C.76. If a *BusinessTransactionAction* calls a two-way *BusinessTransaction*, this *BusinessTransactionAction* MUST be the source of exactly one *RespondingFlow* which target MUST be a *BusinessCollaborationPartition*.

C.77. If a *BusinessTransactionAction* calls a two-way *BusinessTransaction*, this *BusinessTransactionAction* MUST be the target of exactly one *RespondingFlow* which source MUST be either a *BusinessCollaborationPartition* or a *NestedBusinessCollaboration*.

C.78. The *RespondingFlow* sourcing from a *BusinessTransactionAction* and the *RespondingFlow* targeting a *BusinessTransactionAction* MUST NOT be targeting to /sourcing from the same *BusinessCollaborationPartition*, nor targeting to a *NestedBusinessCollaboration* within the same *BusinessCollaborationPartition*.

C.79. If a *BusinessTransactionAction* calls a one-way BusinessTransaction, this *BusinessTransactionAction* MUST NOT be the source of a *RespondingFlow* and MUST NOT be the target of a *RespondingFlow*.

C.80. The *RespondingFlow* targeting a *BusinessTransactionAction* must start from the *BusinessCollaborationPartition* / *NestedBusinessCollaboration* which is the target of the *InitialFlow* starting from the same *BusinessTransactionAction*.

C.81. The *RespondingFlow* starting from a *BusinessTransactionAction* must target the *BusinessCollaborationPartition* which is the source of the *InitialFlow* targeting to the same *BusinessTransactionAction*.

C.82. A *NestedBusinessCollaboration* MUST be the target of exactly one *InitialFlow*.

C.83. A *NestedBusinessCollaboration* MAY be the source of a *RespondingFlow*, but MUST NOT be the source of more than one *RespondingFlow*.

C.84. A *BusinessCollaborationAction* MUST be the target of two to many *InformationFlows* (UML standard: <<flow>>).

C.85. A *BusinessCollaborationAction* MUST not be the source of an *InformationFlow*.

C.86. A *BusinessCollaborationAction* MUST not be the source and MUST not be the target of an *InitialFlow*.

C.87. A *BusinessCollaborationAction* MUST not be the source and MUST not be the target of a *RespondingFlow*.

C.88. A *BusinessTransactionAction* MUST not be the source and MUST not be the target of an *InformationFlow* (<<flow>>) that is neither stereotyped as *InitialFlow* nor as *RespondingFlow* nor is of type <<flow>>.

C.89. A *NestedBusinessCollaboration* MUST not be the source and MUST not be the target of an *InformationFlow* that targets to / sources from a *BusinessCollaborationAction*.

C.90. The number of *InformationFlows* targeting a *BusinessCollaborationAction* MUST match the number of *BusinessCollaborationPartitions* contained in the *BusinessCollaborationProtocol* that is called by this *BusinessCollaborationAction*.

C.91. Either an AuthorizedRole classifying a *BusinessCollaborationPartition* that is the source of an *InformationFlow* (UML standard: <<flow>>) targeting a *BusinessCollaborationAction* MUST match an

AuthorizedRole classifying a *BusinessCollaborationPartition* in the *BusinessCollaborationProtocol* that is called by this *BusinessCollaborationAction* or the *InformationFlow* must be classified by an AuthorizedRole classifying a *BusinessCollaborationPartition* in the *BusinessCollaborationProtocol* that is called by this *BusinessCollaborationAction*.

5.2.3.5 Worksheets

Form for Business Collaboration Use Case	
General	
Name	
Description	
Business Library Information	
UniqueIdentifier	
BusinessTerm	
VersionIdentifier	
Status	
Owner	
Copyright	
Reference(s)	
Participants	
Participating Role	
Participating Role	
[add more participating roles in case of a multiparty collaboration]	
Is Included In (Name of parent Business Collaboration – if there is any)	
Start/End Characteristics	
Affected Business Entities	
Pre-condition	
Post-condition	

Begins When	
Ends When	
Exceptions	
Included Business Transaction Use Cases (add more Business Transaction Use Cases if needed)	
Business Transaction Use Case Name	
Business Transaction Use Case Name	
Business Transaction Use Case Name	
Business Transaction Use Case Name	

1064

Form for Business Collaboration Protocol	
General	
Name	
Description	
Participants (copy from Business Collaboration Use Case Worksheet)	
Participating Role	
Participating Role	
[add more participating roles if elicited in the Business Collaboration Use Case Worksheet]	
Included Business Transaction Actions / Business Collaboration Actions	
Business Transaction Action	
Name	
Preceding Action(s) including transition condition	
Initiating Role	[select one participating role from above]

<table>
<tr><td>Reacting Role</td><td>[select one participating role from above]</td></tr>
<tr><td colspan="2">Business Transaction Action</td></tr>
<tr><td>Name</td><td></td></tr>
<tr><td>Preceding Action(s) including transition condition</td><td></td></tr>
<tr><td>Initiating Role</td><td>[select one participating role from above]</td></tr>
<tr><td>Reacting Role</td><td>[select one participating role from above]</td></tr>
<tr><td colspan="2">Business Transaction Action [add more if needed]</td></tr>
<tr><td>Name</td><td></td></tr>
<tr><td>Preceding Action(s) including transition condition</td><td></td></tr>
<tr><td>Initiating Role</td><td>[select one participating role from above]</td></tr>
<tr><td>Reacting Role</td><td>[select one participating role from above]</td></tr>
<tr><td colspan="2">Business Collaboration Action [delete if not required or add more if needed]</td></tr>
<tr><td>Name</td><td></td></tr>
<tr><td>Preceding Action(s) including transition condition</td><td></td></tr>
<tr><td>Role Mapping</td><td><table><tr><th>Role in this Business Collaboration</th><th>Role in the nested Business Collaboration</th></tr><tr><td></td><td></td></tr></table></td></tr>
<tr><td>Role Mapping</td><td><table><tr><th>Role in this Business Collaboration</th><th>Role in the nested Business Collaboration</th></tr><tr><td></td><td></td></tr></table></td></tr>
<tr><td>[add more Role Mappings if required]</td><td></td></tr>
</table>

1065

5.2.3.6 Normal Business Collaboration View Example (informative)

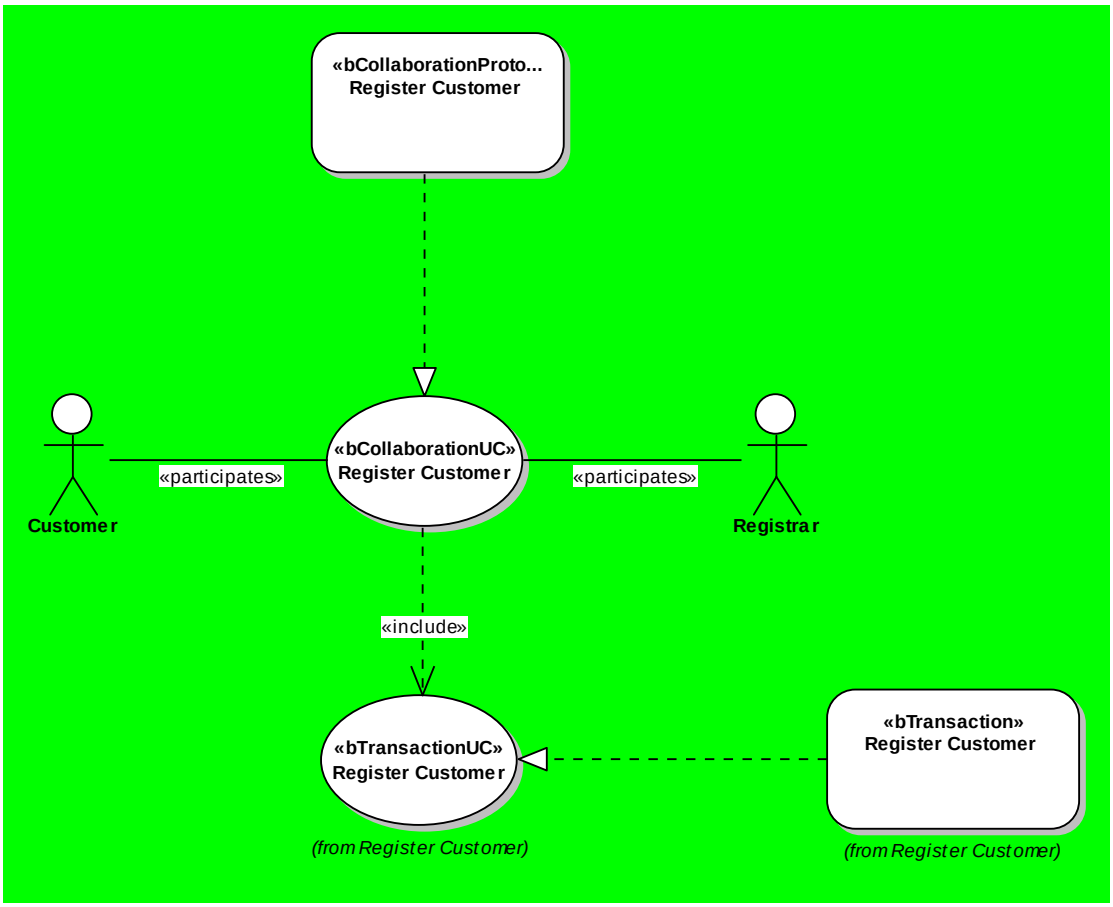


Figure 36 Business Collaboration Use Case Example: Register Customer (including the optional realize relationships)

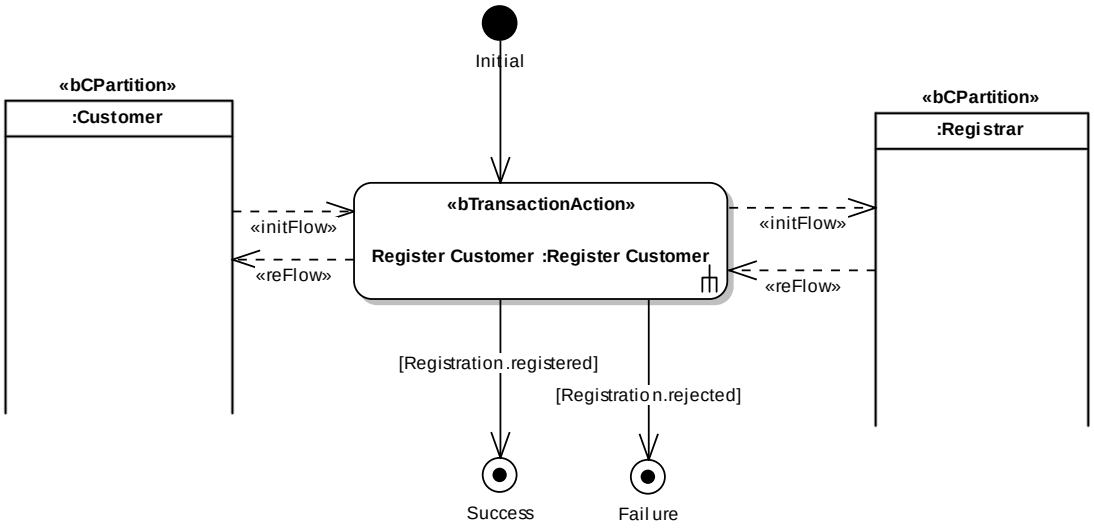


Figure 37 Business Collaboration Protocol Example: Register Customer

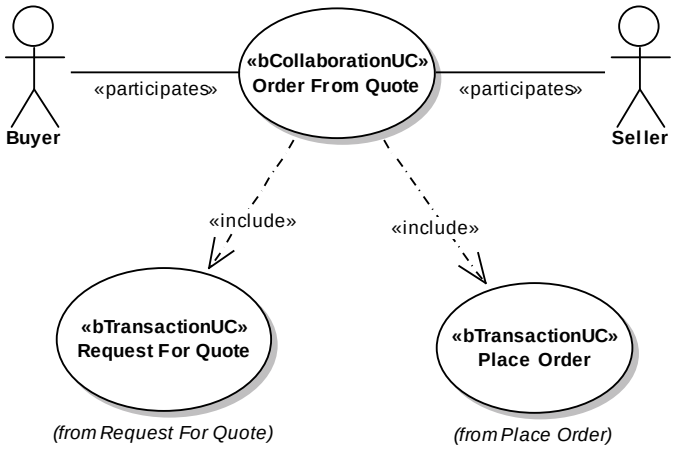


Figure 38 Business Collaboration Use Case Example: Order From Quote (the optional realize relationships are not shown)

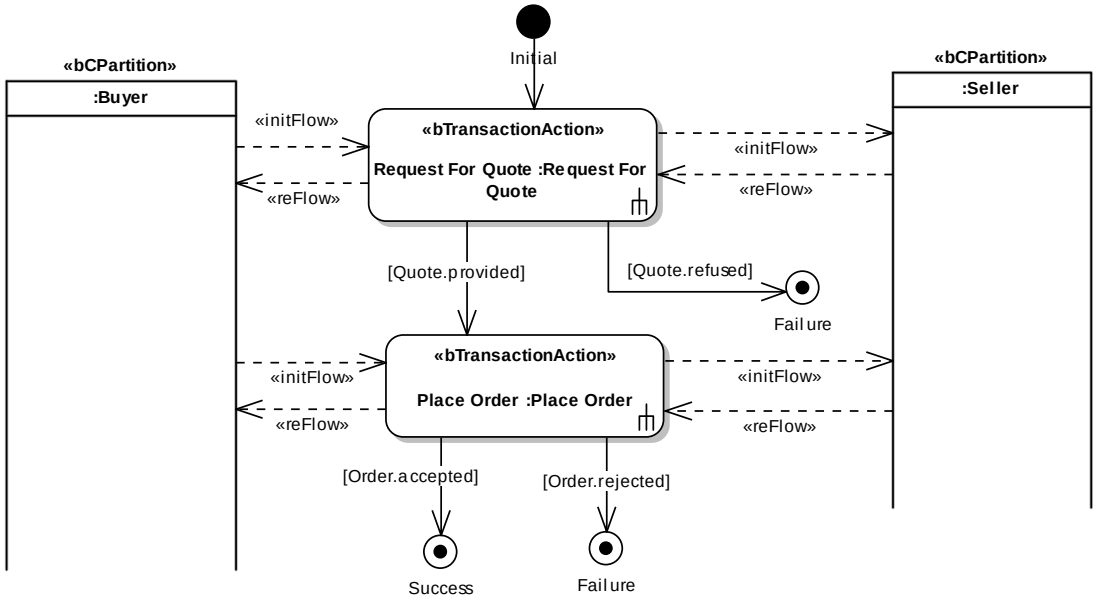
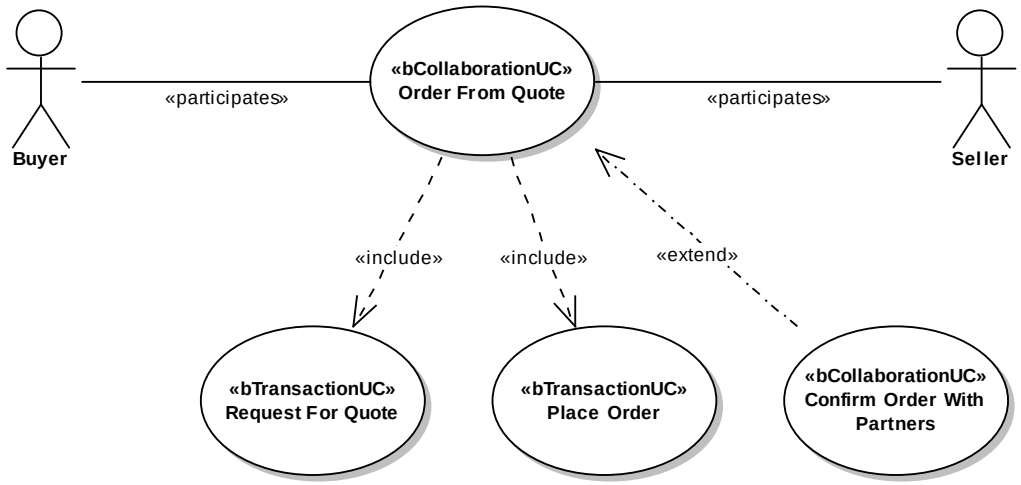


Figure 39 Business Collaboration Protocol Example: Request From Quote

5.2.3.7 Nested Business Collaboration View Example (informative)

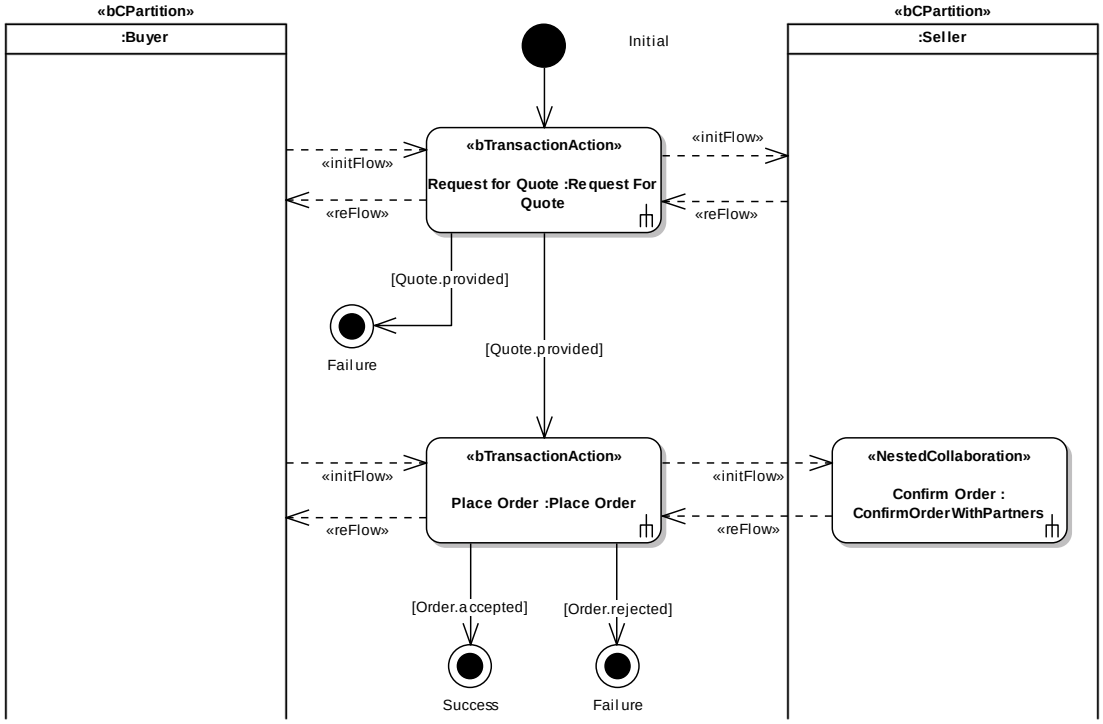
In this example the Order from Quote example is modified to show an example of a Nested Collaboration. Before the Seller can either accept or reject the Buyer's order, the seller must confirm the order with his/her business partners. If this confirmation is unsuccessful, it follows that the Seller will respond by sending a negative response message (e.g. *OrderRejectEnvelope*) to the Buyer.



1082

1083 **Figure 40 Business Collaboration Use Case Example: Order From Quote (the optional realize relationships are not shown)**

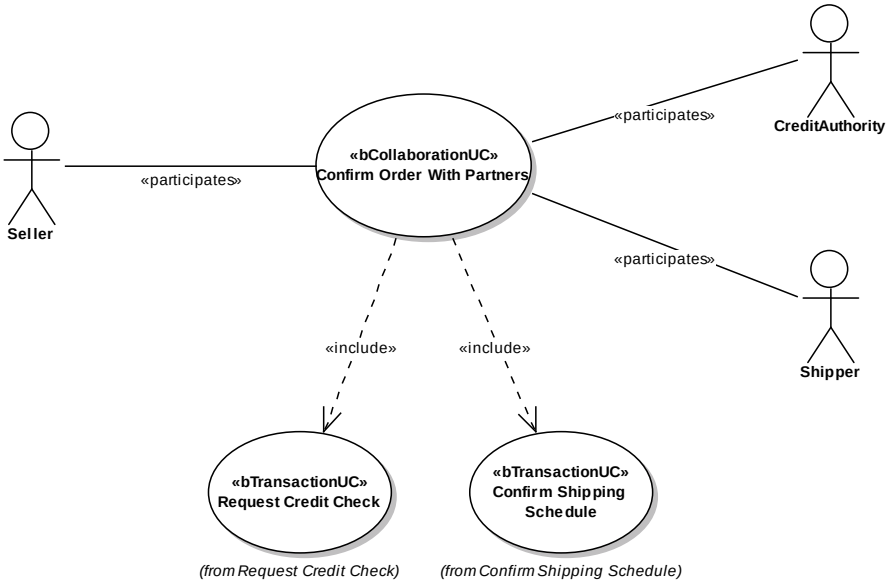
1084



1085

1086 **Figure 41 Business Collaboration Protocol Example: Request For Quote**

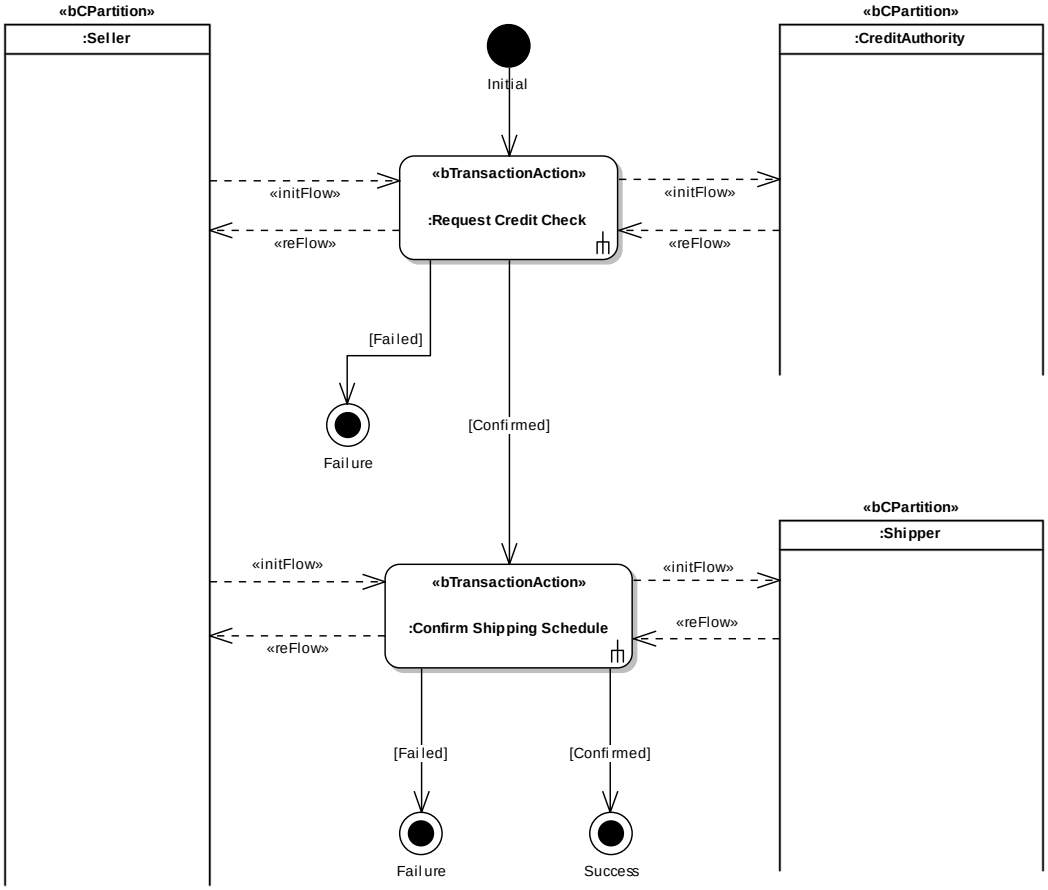
1087



1088

1089 **Figure 42 Business Collaboration Use Case Example: Confirm Order With Partners the optional realize relationships are not**
1090 **shown)**

1091



1092

Figure 43 Business Collaboration Protocol Example: Confirm Order with Partners

5.2.4 Business Realization View

5.2.4.1 Abbreviations and Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bChoreographyV	BusinessChoreographyView
bRealizationV	BusinessRealizationView
bRealizationUC	BusinessRealizationUseCase
bCollaborationUC	BusinessCollaborationUseCase
bPartner	BusinessPartner
bProcessUC	BusinessProcessUseCase

5.2.4.2 Conceptual Description (informative)

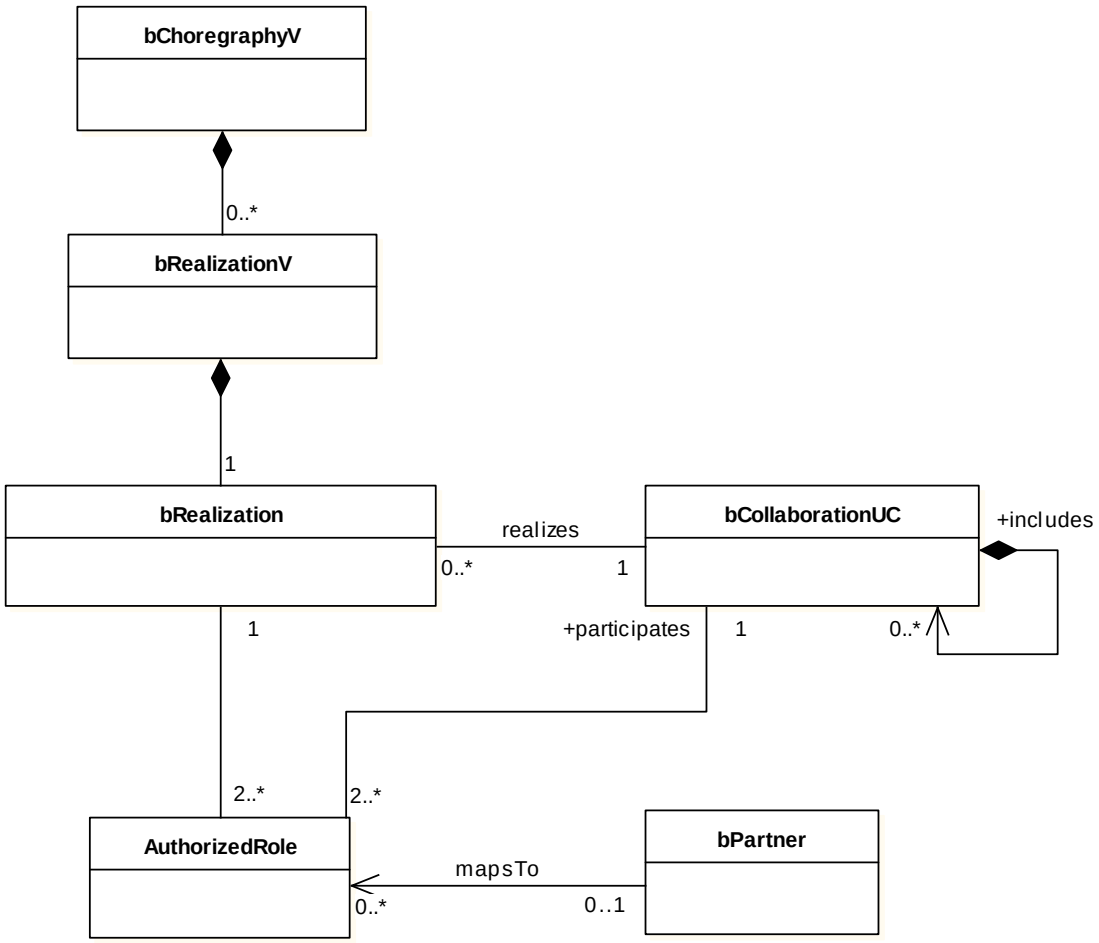


Figure 44 BusinessRealizationView - Conceptual Overview

Business partners identified in the previous business requirements view must not directly be associated with business collaboration use cases and business transaction use cases.

In order to specify that a specific set of business partners collaborate, we use the concept of a business realization. Each business realization is defined in its own business realization view. Accordingly, the *BusinessRealizationView* is composed of exactly one *BusinessRealization*. A business realization realizes exactly one business collaboration use case. Each business collaboration use case may be realized by multiple business realizations. Not each business collaboration use cased (e.g. one that is nested within another one) needs to have a corresponding business realization. As a consequence, the *realizes*-association between a BusinessCollaborationUseCase and BusinessRealization is a 1 to (0..n).

Two or more authorized roles participate in a business realization. These authorized roles (e.g. seller, payee) must be defined in the same business realization view package as the corresponding business realization. Accordingly a *BusinessRealizationView* 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 realization (e.g. payer and payee) realizing it. However, the authorized roles participating in the business collaboration use case and the business realization will be defined in different namespaces – each in the package of the corresponding view. In **Figure** 46 the authorized role *Buyer* on the lower left hand side participates in the business collaboration use case. It is defined in a different namespace than the *Buyer* participating in the business realization.

Similar to the business collaboration use case, the *BusinessRealization* and *AuthorizedRole* are related by an 1 to (2..n) association. Furthermore, the number of actors participating in a business collaboration use case must be the same as the number of actors participating in the business realization realizing it.

In order to bind a business realization to the business partners executing it, business partners are mapped to the authorized roles participating in the business realization. It is required that each authorized role of a business realization (but not an authorized role in general) is target of exactly one *mapsTo*-association from a business partner. A business partner may play multiple authorized roles of a business realization. Consequently, there is a (0..1) to (0..n) *mapsTo*-association between *BusinessPartner* and *AuthorizedRole*.

5.2.4.3 Stereotypes and Tag Definitions (normative)

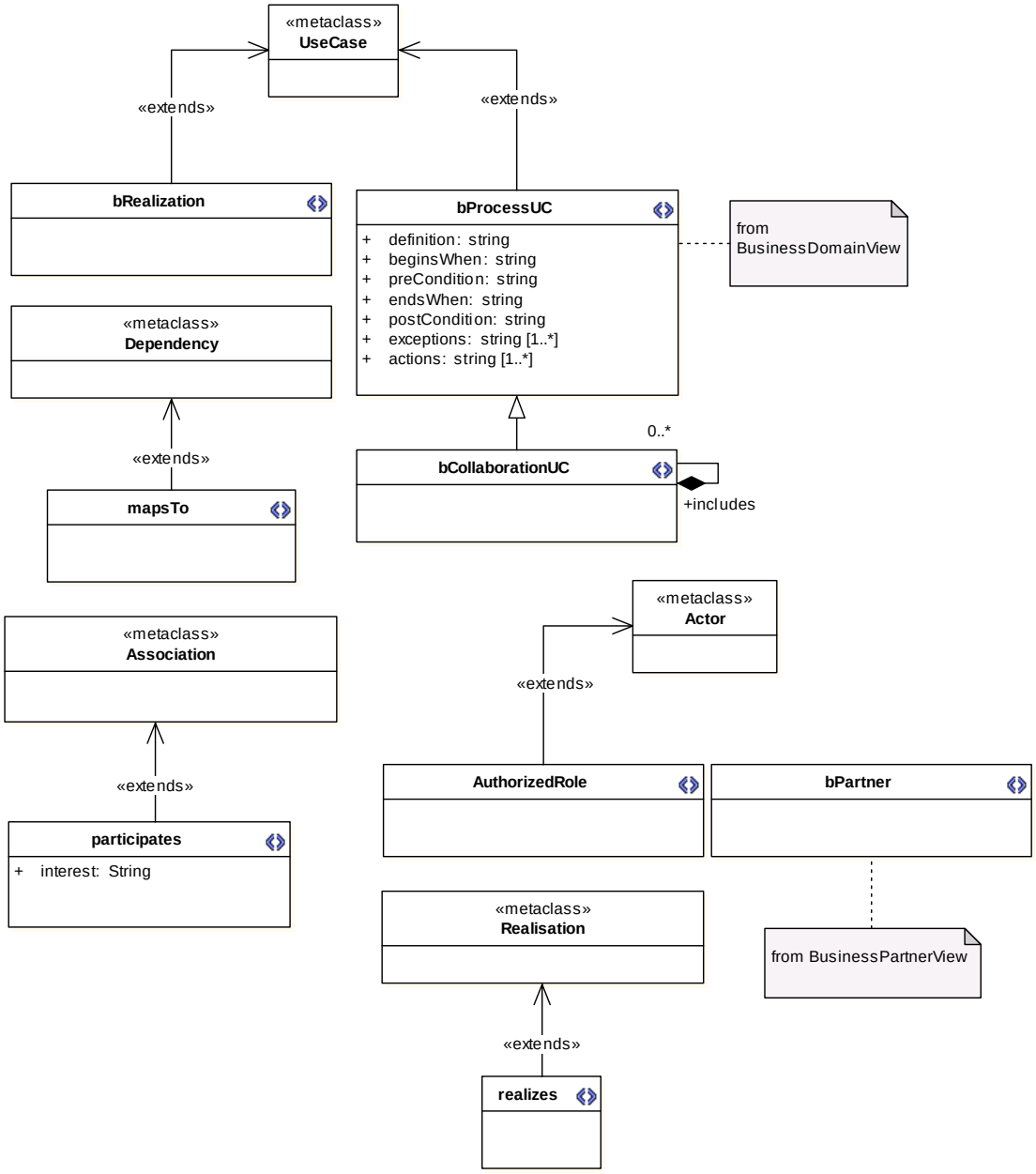


Figure 45 BusinessRealizationView - Abstract Syntax

Stereotype	bRealization (BusinessRealization)
Base Class	UseCase
Parent	N/A
Description	A business realization realizes a business collaboration use case between a specific set of business partners. The requirements of the business realization are the ones defined in the tags of the corresponding business collaboration use case. Thus, the business realization does not include any tag

	definitions for capturing requirements.
Tag Definition	No tagged values

Stereotype	**AuthorizedRole (AuthorizedRole)**
Base Class	Actor
Parent	N/A
Description	An authorized role (e.g. a "buyer") is a concept which is more generic than a business partner (e.g. a "broker") and allows the reuse of collaborations by mapping an *AuthorizedRole* to a business partner 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 partners (a "broker" or a "custodian") in different scenarios of the same domain or even in different domains.
Tag Definition	No tagged values.

Stereotype	**mapsTo (mapsTo)**
Base Class	Dependency
Parent	N/A
Description	A mapsTo dependency represents (1) the fact, that a business partner plays a certain authorized role in a business 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.
Tag Definition	No tagged values.

5.2.4.4 Constraints (normative)

C.92. A *BusinessRealizationView* MUST contain exactly one *BusinessRealization*, two to many *AuthorizedRoles*, and two to many *participates* associations.

C.93. A *BusinessRealization* MUST be associated with two to many *AuthorizedRoles* via stereotyped binary *participates* associations.

C.94. A *BusinessRealization* MUST be the source of exactly one realization dependency to a *BusinessCollaborationUseCase*.

C.95. A *BusinessRealization* MUST NOT be the source or target of an *include* or *extends* association.

C.96. All dependencies from/to an *AuthorizedRole* must be stereotyped as *mapsTo*.

C.97. An *AuthorizedRole*, which participates in a *BusinessRealization*, must be the target of exactly one *mapsTo* dependency starting from a *BusinessPartner*. Furthermore the *AuthorizedRole*, which participates in the *BusinessRealization* must be the source of exactly one *mapsTo* dependency targeting an *AuthorizedRole* participating in a *BusinessCollaborationUseCase*.

C.98. *AuthorizedRoles* in a *BusinessRealizationView* must have a unique name within the scope of the package, they are located in

C.99. The number of *AuthorizedRoles* participating in a *BusinessCollaborationUseCase* MUST match the number of *AuthorizedRoles* participating in the *BusinessRealization* realizing this *BusinessCollaborationUseCase*

5.2.4.5 Example (informative)

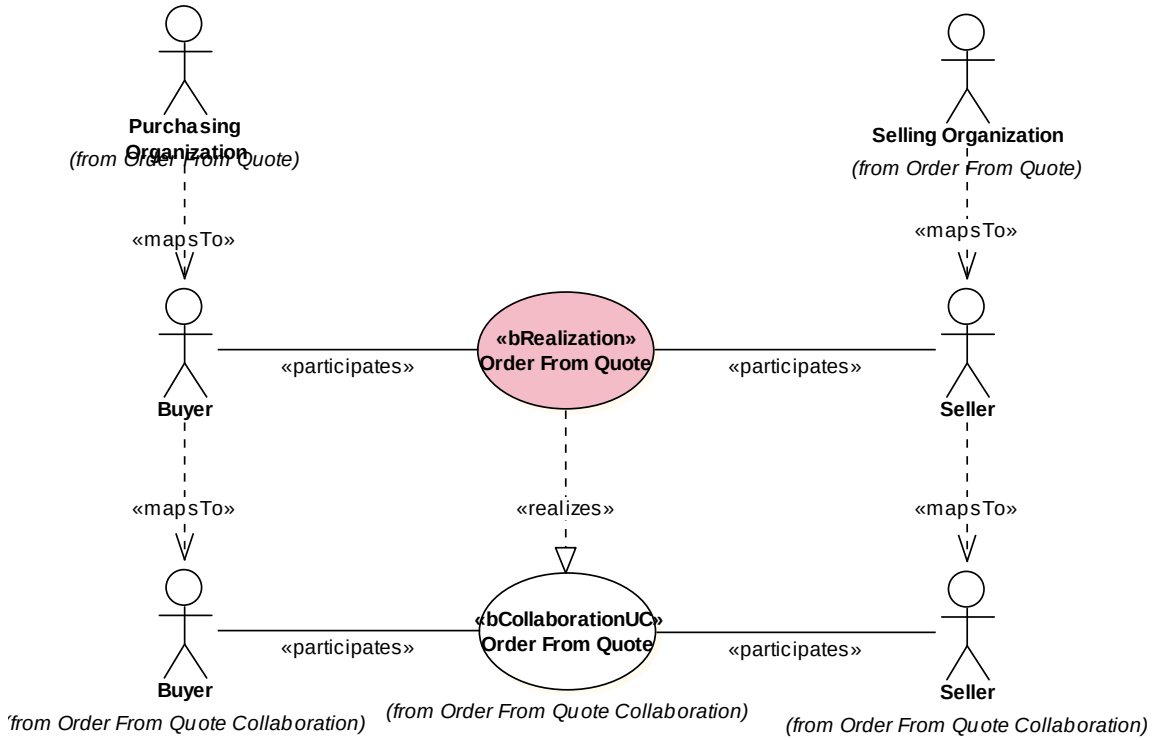


Figure 46 BusinessRealizationView - Example: Realization of the OrderFromQuote Collaboration between Purchasing Organization and SellingOrganization

5.3 Business Information View

5.3.1 Abbreviations of Stereotypes

Stereotype Abbreviation	Full Stereotype Name
bInformationV	BusinessInformationView
bInformation	BusinessInformation
InfEnvelope	Information Envelope

5.3.2 Conceptual Description (informative)

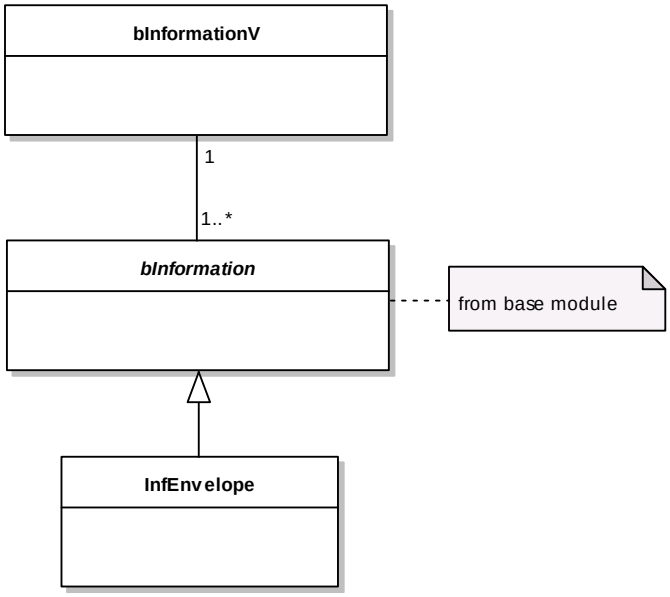


Figure 47 BusinessInformationView - Conceptual Overview

A *BusinessInformationView* is a container of artifacts that describe the information exchanged in a *BusinessTransaction*. As previously mentioned; *RequestingInformationPin* and *RespondingInformationPin* are classified by an *InformationEnvelope* which is a subclass of a *BusinessInformation*. A *BusinessInformation* serves as an abstract container for all of the information exchanged between the *RequestingAction* and the *RespondingAction* or vice versa, respectively. The stereotypes *BusinessInformation* and *InformationEnvelope* is part of the UMM base module and imported into the UMM foundation module.

The current UMM foundation module does not mandate a specific business information modeling approach. All methodologies and rules to build quality class diagrams can be used in order to model the exchanged information, as long as the root element of the data structure generalizes the *InformationEnvelope* class being part of the UMM base module.

However, UMM strongly suggests using UN/CEFACT's Core Components and Core Components Message Assembly artifacts to model the business information. Because Core Components are syntax independent and stereotyped, the usage of the UML Profile for Core Components is suggested within the *BusinessInformationView*.

5.3.3 Stereotypes and Tag Definitions (normative)

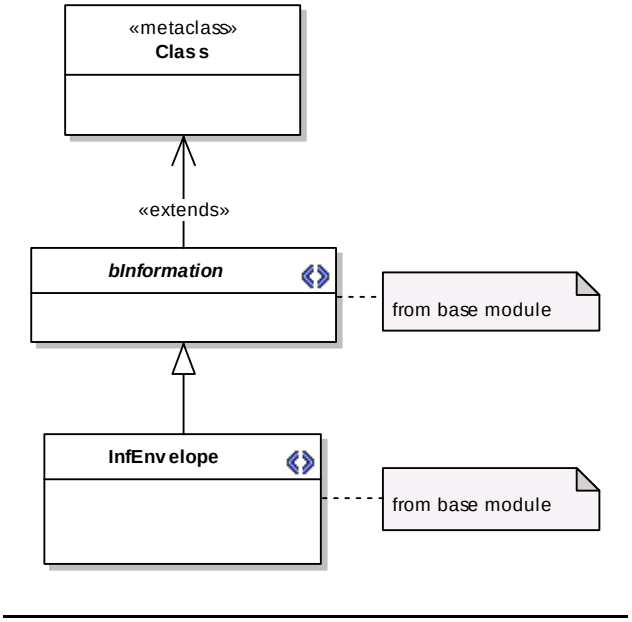


Figure 48 BusinessInformationView - Abstract Syntax

Stereotype	bInformation (BusinessInformation)
Base Class	Class
Parent	N/A
Description	A *BusinessInformation* realizes abstract business document information that is exchanged between authorized roles performing activities in a business transaction. Since a *BusinessInformation* is defined as abstract it cannot be used directly in order to set the type of exchanged information in a *BusinessInformation*. Instead the concept of an *InformationEnvelope* is used.

Stereotype	InfEnvelope (InformationEnvelope)
Base Class	Class
Parent	BusinessInformation
Description	An *InformationEnvelope* is a subtype of a *BusinessInformation* and represents a concrete business message which is exchanged in a UMM business transaction. Any business document artifacts are connected to an *InformationEnvelope* using associations.

5.3.4 Constraints (normative)

C.100. A *BusinessInformationView* MUST contain one to many *InformationEnvelopes* or subtypes thereof defined in any other extension/specialization module. Furthermore, it MAY contain any other document modeling artifacts.

5.3.5 Example using UPCC (UML Profile for Core Components) and CCMA (Core Components Message Assembly) (informative)

The following example shows how to model the information exchanged in a *BusinessTransaction* using the UML Profile for Core Components (UPCC) and the Core Component Message Assembly (CCMA). Figure 49 shows how UMM, UPCC and CCMA are related to each other.

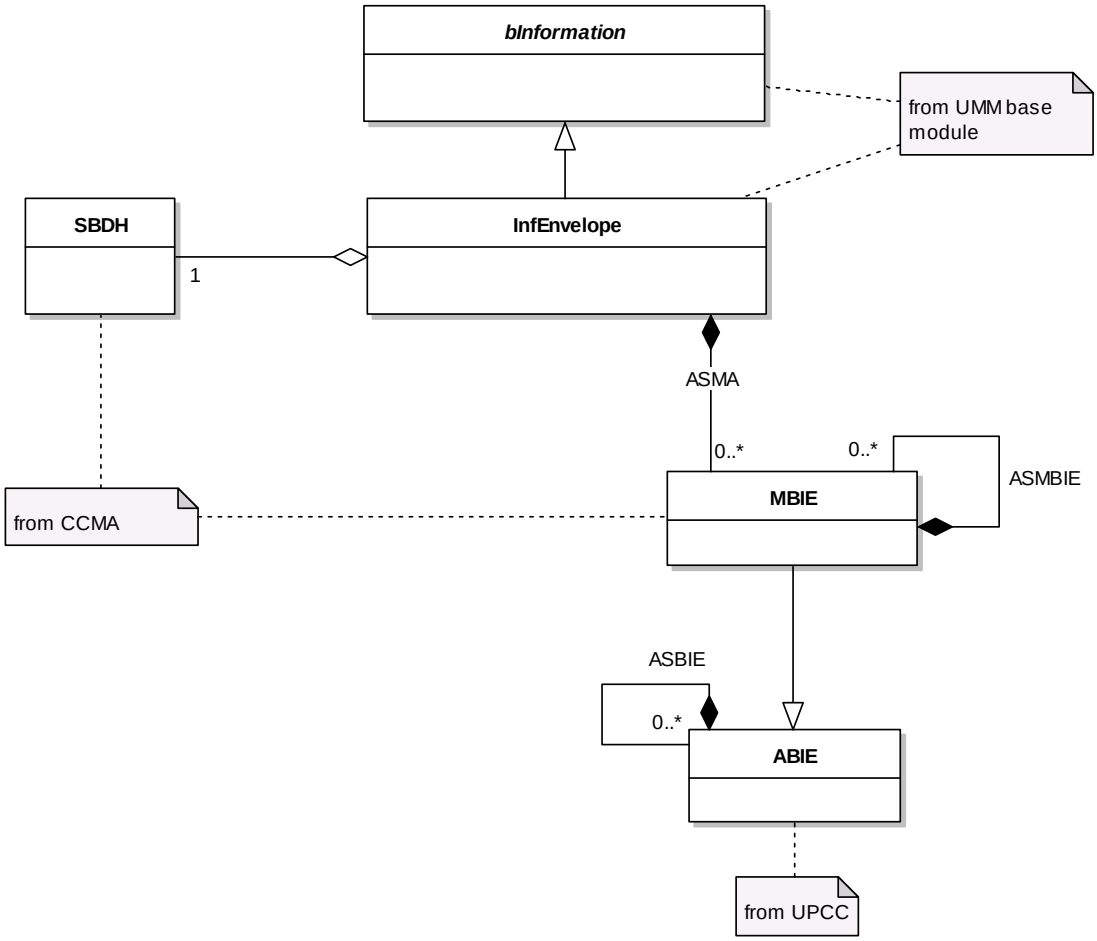


Figure 49 Abstract example for using UPCC artifacts to model business information

An *information envelope* has a *standard business document header* (SBDH) which serves for identification purposes of technical sender and receiver, document type etc.

The body of an *information envelope* consists of zero or more elements – so called *messaging business information entities* (MBIE). MBIEs are connected to an *information envelope* using so called *association message assemblies* (ASMA). If different *messaging business information entities* are connected to each other the concept of *association messageing business information entities* (ASMBIE) is used. A *messaging business information entity* inherits from an *aggregate business information entity* (ABIE). ABIEs are part of the UML Profile for Core Components (UPCC) standard. In order to relate different *aggregate business information entities* to each other so called *association business information entities* (ASBIE) are used.

Standard business document header and *messaging business information entities* are part of the core components message assembly (CCMA). *Aggregate business information entities* are part of the UML Profile for Core Components (UPCC).

Figure 50 shows and example for a QuoteEnvelope modelled using concepts from the Core Components Message Assembly standard. Not shown in Figure 50 are the dependencies between ABIEs and MBIEs.

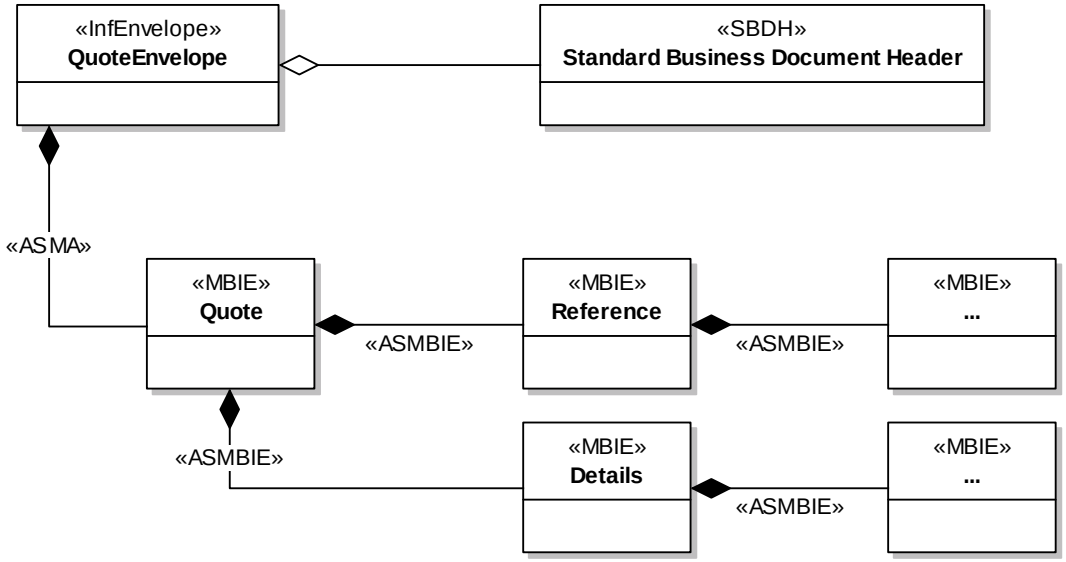


Figure 50 CCMA example

6 OCL Constraints

```
-- Constraint 1
-- A BusinessCollaborationModel MUST contain one to many
   BusinessChoreographyViews.
context Package inv: self.isBCollModel()
    implies self.nestedPackage->exists(a|a.isBChoreographyV())
```

```
-- Constraint 2
-- A BusinessCollaborationModel MUST contain one to many
   BusinessInformationViews.
context Package inv: self.isBCollModel()
    implies self.nestedPackage->exists(a|a.isBInformationV())
```

```
-- Constraint 3
-- A BusinessCollaborationModel MAY contain zero to many
   BusinessRequirementsViews
context Package inv: self.isBCollModel()
    implies self.nestedPackage->select(a|a.isBRequirementsV())->
   size>=0
```

```
-- Constraint 4
-- A BusinessRequirementsView, a BusinessChoreographyView and a
   BusinessInformationView
-- MUST be directly located under a BusinessCollaborationModel
context Package inv: (self.isBChoreographyV() or
   self.isBInformationV() or self.isBRequirementsV())
    implies self.hlpOwningPackage().isBCollModel()
```

```
-- Constraint 5
--A BusinessRequirementsView MAY contain zero or one
   BusinessDomainViews.
context Package inv: self.isBRequirementsV()
    implies self.nestedPackage->select(a|a.isBDomainV())->size <=1
```

```
-- Constraint 6
-- A BusinessChoreographyView MAY contain zero or one
   BusinessPartnerViews.
context Package inv: self.isBRequirementsV()
    implies self.nestedPackage->select(a | a.isBPartnerV())->size <=1
```

```
-- Constraint 7
-- A BusinessRequirementsView MAY contain zero to many
   BusinessEntityViews.
context Package inv: self.isBRequirementsV()
    implies self.nestedPackage->select( a | a.isBEntityV())->size >= 0
```

```
-- Constraint 8
```

```
-- A BusinessDomainView, a BusinessPartnerView, and a
   BusinessEntityView
-- MUST be located directly under a BusinessRequirementsView
context Package inv:
    self.isBDomainV() or self.isBPartnerV() or self.isBEntityV()
    implies self.hlpOwningPackage().isBRequirementsV()
```

```
-- Constraint 9
-- A BusinessDomainView MUST include one to many BusinessAreas.
context Package inv: self.isBDomainV()
    implies self.nestedPackage->exists( a | a.isBArea())
```

```
-- Constraint 10
-- A BusinessArea MUST include one to many BusinessAreas or one to
-- many ProcessAreas or one to many BusinessProcessUseCases.
context Package inv: self.isBArea()
    implies ( self.nestedPackage->exists( a | a.isBArea()
    or a.isProcessArea()) or ( self.ownedElement ->
    exists( b | b.oclAsType(UseCase).isBProcessUseCase()) ) )
```

```
-- Constraint 11
-- A ProcessArea MUST contain one to many other ProcessAreas or one to
   many BusinessProcessUseCases
context Package inv: self.isProcessArea()
    implies (self.nestedPackage -> exists( a | a.isProcessArea())) or
   (self.ownedElement ->
    exists( b | b.oclAsType(UseCase).isBProcessUseCase()))
```

```
-- Constraint 12
--A BusinessProcessUseCase MUST be associated with one to many
   BusinessPartners using the participates relationship
context UseCase inv: self.isBProcessUseCase() and
   not(self.isBCollaborationUC())
    and not(self.isBTransactionUC())
    implies self.owner.ownedElement->exists(a|
   a.oclAsType(Association).isParticipates()
    and a.oclAsType(Association).ownedEnd.type->
   exists(t|t.oclAsType(Actor).isBPartner())
    and a.oclAsType(Association).ownedEnd.type->
   exists(t|t.oclAsType(UseCase)=self))
```

```
-- Constraint 13
-- A BusinessProcessUseCase may be associated with zero to many
   Stakeholders using the isOfInterestedTo relationship
context UseCase inv: self.isBProcessUseCase()
    implies if self.owner.ownedElement->
   exists(a|a.oclAsType(Actor).isStakeholder())
   then self.oclAsType(UseCase).clientDependency->
   exists(p|p.oclAsType(Dependency).isOfInterestTo()
   and p.client->exists(t|t.oclAsType(Actor).isStakeholder() ))
   else true endif
```

```
-- Constraint 14
-- A BusinessProcessUseCase SHOULD be refined by zero to many
   BusinessProcesses. These relationships MAY also be visualized by
   realize relationships from each of the owned BusinessProcesses to
   the owning BusinessProcessUseCase
context UseCase inv: self.isBProcessUseCase()
    implies self.ownedElement->select( process |
   process.oclAsType(Activity).isBProcess())->size()>=0
```

```
-- Constraint 15
-- A BusinessProcess MUST be modeled as a child of a
   BusinessProcessUseCase
context Activity inv: self.isBProcess()
    implies self.owner.oclAsType(UseCase).isBProcessUseCase()
```

```
Constraint 16 refers to a Diagram and is therefore not represented in
   OCL
```

```
-- Constraint 17
-- A BusinessProcess MAY contain zero to many ActivityPartitions
context Activity inv: self.isBProcess()
    implies self.ownedElement->select( partitions |
   partitions.oclIsTypeOf(ActivityPartition))->size()>=0
```

```
-- Constraint 18
-- A BusinessProcess, which has no ActivityPartitions, MUST contain
   one or more BusinessProcessActions and
-- MAY contain zero to many InternalBusinessEntityStates and zero to
   many SharedBusinessEntityStates.
context Activity inv: self.isBProcess()
    implies if self.ownedElement->
   select(b|b.oclAsType(ActivityPartition).isActivityPartition())->
   size=0
   then self.ownedElement-
   exists(bp|bp.oclAsType(Action).isBProcessAction()) else true endif
```

```
-- Constraint 19
-- An ActivityPartition being part of a BusinessProcess MUST contain
   one to many BusinessProcessActions and
-- MAY contain zero to many InternalBusinessEntityStates.
context ActivityPartition inv: self.isActivityPartition() and
   self.owner.oclAsType(Activity).isBProcess()
    implies self.ownedElement->exists( act |
   act.oclAsType(Action).isBProcessAction())
```

```
-- Constraint 20
-- A SharedBusinessEntityState MUST NOT be located in an
   ActivityPartition. (They must be contained within
```

```
-- the BusinessProcess even if this BusinessProcess contains
   ActivityPartitions.)
context ObjectNode inv: self.oclAsType(ObjectNode).isBESharedState()
   and self.owner.oclAsType(Activity).isBProcess()
    implies
   not(self.owner.oclAsType(ActivityPartition).isActivityPartition())
```

```
-- Constraint 21
-- A BusinessPartnerView MUST contain at least two to many
   BusinessPartners. If the BusinessPartnerView is hierarchically
   decomposed into subpackages these BusinessPartners MAY be contained
   in any of these subpackages.
Missing on purpose
```

```
-- Constraint 22
-- A BusinessPartnerView MAY contain zero to many Stakeholders
Missing on purpose
```

```
-- Constraint 23
-- A BusinessEntityView must contain one to many BusinessEntities
context Package inv: self.isBEntityV()
    implies self.ownedElement->
   exists(a|a.oclAsType(Class).isBEntity())
```

```
Constraint 24 refers to a diagram and is therefore not represented in
   OCL
```

```
-- Constraint 25 (Since the first part of the constraint refers to a
   diagram, only the second part of the constraint is represented in
   OCL)
-- A UML State Diagram describing the lifecycle of a BusinessEntity
   MUST contain one to many BusinessEntityStates. The parent of a
   BusinessEntityState MUST be a BusinessEntity
context Class inv: self.isBEntityState()
   and self.owner.isBEntity()
```

```
-- Constraint 26
-- A BusinessEntity MAY contain zero to many BusinessDataViews that
   describes its conceptual design
context Package inv: self.isBEntityV()
    implies self.nestedPackage->select( package |
   package.oclAsType(Package).isBDataV())->size>=0
```

```
-- Constraint 27
-- The parent of a BusinessDataView MUST be a BusinessEntityView
context Package inv: self.isBDataV()
    implies self.owner.isBEntityV()
```

```
-- Constraint 29
-- A BusinessDataView SHOULD contain one to many classes.
context Package inv: self.isBDataV()
    implies self.ownedElement->select( elem |
   elem.oclIsTypeOf(Class))->size()>=1
```

```
-- Constraint 30
-- A BusinessChoreographyView MUST contain one to many
   BusinessCollaborationViews
context Package inv: self.isBChoreographyV()
    implies self.nestedPackage->exists(c|c.isBCollaborationV())
```

```
-- Constraint 31
--A BusinessChoreographyView MUST contain one to many
   BusinessTransactionViews
context Package inv: self.isBChoreographyV()
    implies self.nestedPackage->exists(c|c.isBTransactionV())
```

```
-- Constraint 32
-- A BusinessChoreographyView MAY contain zero to many
   BusinessRealizationViews
context Package inv: self.isBChoreographyV()
    implies self.nestedPackage->select( package |
   package.oclAsType(Package).isBRealizationV())->size()>=0
```

```
-- Constraint 33
-- A BusinessTransactionView, a BusinessCollaborationView, and a
   BusinessRealizationView
-- MUST be directly located under a BusinessChoreographyView
context Package inv: self.isBTransactionV() or
   self.isBCollaborationV() or self.isBRealizationV()
    implies self.owner.isBChoreographyV()
```

```
-- Constraint 34
-- A BusinessTransactionView MUST contain exactly one
   BusinessTransactionUseCase, exactly two AuthorizedRoles,
-- and exactly two participates associations.
context Package inv: self.isBTransactionV()
    implies self.ownedElement->
    select(a|a.oclAsType(UseCase).isBTransactionUC())->size=1
    and self.ownedElement->
    select(b|b.oclAsType(Actor).isAuthorizedRole())->size=2
    and self.ownedElement->select(c|c.oclIsTypeOf(Association)
    and c.oclAsType(Association).isParticipates())->size=2
    and self.ownedElement->size=5
```

```
-- Constraint 35
-- A BusinessTransactionUseCase MUST be associated with exactly two
   AuthorizedRoles via stereotyped binary participate associations.
context UseCase inv: self.isBTransactionUC()
```

```
    implies self.owner.ownedElement->
    select(q|q.oclIsKindOf(Association))->
    forAll(a|a.oclAsType(Association).isParticipates()
    and a.oclAsType(Association).ownedEnd.type->
    forAll(t|t.oclAsType(Actor).isAuthorizedRole()
    or t.oclAsType(UseCase)=self))
```

```
-- Constraint 36
-- A BusinessTransactionUseCase MUST NOT include further UseCases
context UseCase inv: self.isBTransactionUC()
    implies self.include->size=0
```

```
-- Constraint 37
-- A BusinessTransactionUseCase MUST be included in at least one
   BusinessCollaborationUseCase.
context UseCase inv: self.isBTransactionUC()
    implies self.owner.owner.oclAsType(Package).nestedPackage
    ->select(collV|collV.isBCollaborationV())->
    exists(c|c.ownedElement
    ->exists(k|k.oclAsType(UseCase).isBCollaborationUC()
    and k.oclAsType(UseCase).include.addition
    ->exists(s|s.oclAsType(UseCase)=self)))
```

```
-- Constraint 38
--A BusinessTransactionUseCase MUST NOT be source or target of an
   extend association.
context UseCase inv: self.isBTransactionUC()
    implies self.extend->size=0 and UseCase.allInstances->
    forAll(u|u.oclAsType(UseCase).extend.extendedCase->
    forAll(t|t.oclAsType(UseCase).isNotBTransactionUC()))
```

```
-- Constraint 39
-- The two AuthorizedRoles within a BusinessTransactionView MUST NOT
   be named identically
context Actor inv: self.isAuthorizedRole() and
    self.owner.oclAsType(Package).isBTransactionV()
    implies self.owner.ownedElement->
    select(k|k.oclAsType(Actor).isAuthorizedRole())->
    collect(p|p.oclAsType(Actor).name)->asSet->size()=2
```

```
-- Constraint 40
-- A BusinessTransactionUseCase MUST be described by exactly one
   BusinessTransaction defined as a child element of this
   BusinessTransactionUseCase.
context UseCase inv: self.isBTransactionUC()
    implies self.ownedElement->select( act |
   act.oclAsType(Activity).isBTransaction()) -> size = 1
```

```
-- Constraint 41
-- A BusinessTransaction MUST have exactly two partitions.
```

```
-- Each of them MUST be stereotyped as BusinessTransactionPartition.
context Activity inv: self.isBTransaction()
    implies self.ownedElement->select( part |
    part.oclAsType(ActivityPartition).isBTPartition())->size=2
```

```
-- Constraint 42
-- One of the two BusinessTransactionPartitions MUST contain one
   RequestingBusinessAction and
-- the other one MUST contain one RespondingBusinessAction.
context Activity inv:
    let ReqAct : Action = self.ownedElement->
    select( act | act.oclAsType(Action).isReqAction())->
    asSequence->first().oclAsType(Action)
    in let ResAct : Action = self.ownedElement->
    select( act | act.oclAsType(Action).isResAction())->
    asSequence->first().oclAsType(Action)
    in let Part1 : ActivityPartition = self.ownedElement->
    select( part | part.oclAsType(ActivityPartition).isBTPartition())
    ->asSequence->first().oclAsType(ActivityPartition)
    in let Part2 : ActivityPartition = self.ownedElement->
    select( part | part.oclAsType(ActivityPartition).isBTPartition())
    ->asSequence->last().oclAsType(ActivityPartition)
    in self.isBTransaction()
    implies self.ownedElement->
    select( act | act.oclAsType(Action).isReqAction())->size=1
    and self.ownedElement->
    select( act | act.oclAsType(Action).isResAction())->size=1
    and ReqAct.inPartition->size=1 and ResAct.inPartition->size=1
    and ( (ReqAct.inPartition->asSequence
    ->first().oclAsType(ActivityPartition)=Part1
    and ResAct.inPartition->asSequence
    ->first().oclAsType(ActivityPartition)=Part2)
    or (ResAct.inPartition->asSequence
    ->first().oclAsType(ActivityPartition)=Part1
    and ReqAct.inPartition->asSequence
    ->first().oclAsType(ActivityPartition)=Part2 ) )
```

```
-- Constraint 43
-- A BusinessTransactionPartition MUST have a classifier, which MUST
   be one of the associated
-- AuthorizedRoles of the corresponding BusinessTransactionUseCase.
context ActivityPartition inv:
    let authRoles : Sequence(Element) =
    self.owner.owner.owner.ownedElement->
    select(roles | roles.oclAsType(Actor).isAuthorizedRole())
    ->asSequence in self.isBTPartition()
    implies self.name=authRoles->first().oclAsType(Actor).name
    or self.name=authRoles->last().oclAsType(Actor).name
```

```
-- Constraint 44
-- The two BusinessTransactionPartitions MUST have different
   classifiers.
context Activity inv: self.isBTransaction()
    implies not(self.ownedElement->
```

```
    select( part | part.oclAsType(ActivityPartition).isBTPartition())
    -> asSequence->first().oclAsType(ActivityPartition).name =
    self.ownedElement->
    select( part | part.oclAsType(ActivityPartition).isBTPartition())
    -> asSequence->last().oclAsType(ActivityPartition).name)
```

```
-- Constraint 45
-- The BusinessTransactionPartition containing the
   RequestingBusinessAction MUST contain two or more FinalStates.
-- Each of the FinalStates MAY have a SharedBusinessEntityState as
   predecessor.
-- One of the FinalStates SHOULD reflect a ControlFailure - this
   FinalState SHOULD NOT have a
-- predecessing SharedBusinessEntityState.
context ActivityPartition inv: self.isBTPartition() and
   self.containedNode->exists( action |
   action.oclAsType(Action).isReqAction())
   implies self.containedNode->select( finNode |
   finNode.oclIsTypeOf(ActivityFinalNode))->size()>=2
```

```
-- Constraint 46
-- A RequestingBusinessAction MUST embed exactly one
   RequestingInformationPin
context Action inv: self.isReqAction()
   implies
   self.ownedElement->select( pin | pin.oclAsType(Pin).isReqInfPin())
   ->size=1
```

```
-- Constraint 47
-- A RespondingBusinessAction MUST embed exactly one
   RequestingInformationPin
context Action inv: self.isResAction()
   implies
   self.ownedElement->select( pin | pin.oclAsType(Pin).isReqInfPin())
   ->size=1
```

```
-- Constraint 48
-- If the tagged value businessTransactionType of the
   BusinessTransaction is either Request/Response, Query/Response,
   Request/Confirm, or CommercialTransaction, then the
   RequestingBusinessAction must embed one to many
   RespondingInformationPins and the RespondingBusinessAction must
   embed one to many RespondingInformationPins.
context Action inv: self.isBTransaction() and
   self.hlpMustHaveResInfPin() implies
   self.ownedElement-> select( action |
   action.oclAsType(Action).isReqAction())
   ->forAll( actions | actions.ownedElement->
   exists( pin | pin.oclAsType(Pin).isResInfPin()))
```

```
-- Constraint 49
```

```
-- If the tagged value businessTransactionType of the
   BusinessTransaction is either Notification or
   InformationDistribution, then both, the RequestingBusinessAction
   and the RespondingBusinessAction, MUST NOT embed a
   RespondingInformationPin
context Action inv: self.isBTransaction() and
   self.hlpMustNotHaveResInfPin()
   implies
   not(self.ownedElement-> select( action |
   ction.oclAsType(Action).isReqAction()
   or action.oclAsType(Action).isResAction())->
   forAll( actions | actions.ownedElement->
   exists( pin | pin.oclAsType(Pin).isResInfPin())))
```

```
-- Constraint 50
-- A RequestingBusinessAction and a RespondingBusinessAction MUST
   embed same
-- number of RespondingInformationPins.
   let ReqAct : Action = self.ownedElement->select( act |
   act.oclAsType(Action).isReqAction())->
   asSequence->first().oclAsType(Action) in
   let ResAct : Action = self.ownedElement->select( act |
   act.oclAsType(Action).isResAction())->
   asSequence->first().oclAsType(Action) in
   self.isBTransaction() implies
   ReqAct.ownedElement->select( resPin |
   resPin.oclAsType(Pin).isResInfPin())->size
   =
   ResAct.ownedElement->select(resPin |
   resPin.oclAsType(Pin).isResInfPin())->size
```

```
-- Constraint 51
--
   The RequestingInformationPin of the RequestingBusinessAction MUST b
   e connected
-- with the
   RequestingInformationPin of the RespondingBusinessAction using an o
   bject
-- flow relationship leading from the RequestingBusinessAction
-- to the RespondingBusinessAction.
Missing on purpose
```

```
-- Constraint 52
-- Each RespondingInformationPin of the RespondingBusinessAction MUST
   be
-- connected with exactly one RespondingInformationPin of the
   RequestingBusinessAction
-- using an object flow relationship leading from the
   RespondingBusinessAction
-- to the RequestingBusinessAction
Missing on purpose
```

```
-- Constraint 53
```

```
-- If a BusinessTransactionPartition contains
   SharedBusinessEntityStates, each SharedBusinessEntityState
-- MUST be the target of exactly one control flow relationship
   starting from the RequestingBusinessAction and
-- MUST be the source of exactly one control flow relationship
   targeting a FinalState.
context ActivityPartition inv: self.isBTPartition()
    implies self.owner.ownedElement->
    select( nodes | nodes.oclIsTypeOf(CentralBufferNode)
    and nodes.oclAsType(CentralBufferNode).isBESharedState()) ->
    forAll( states | states.oclAsType(CentralBufferNode).incoming
    ->size()=1     and states.oclAsType(CentralBufferNode).incoming->
    forAll( income |  income.oclAsType(ObjectFlow).source.
    oclAsType(Action).isReqAction())
    and states.oclAsType(CentralBufferNode).outgoing->
    forAll( outgo | outgo.oclAsType(ObjectFlow).target.
    oclIsTypeOf(ActivityFinalNode)
    and states.oclAsType(CentralBufferNode).outgoing->size()=1) )
```

```
-- Constraint 54
-- Each FinalState MUST be the target of one to many control flow
   relationships starting
-- from the RequestingBusinessAction or from a
   SharedBusinessEntityState.
context ActivityFinalNode inv: self.oclIsTypeOf(ActivityFinalNode)
   and self.owner.oclAsType(Activity).isBTransaction()
   implies self.incoming->forAll( income |
   income.oclAsType(ControlFlow).source.
   oclAsType(Action).isReqAction() or
   income.oclAsType(ObjectFlow).source.oclAsType(CentralBufferNode).is
   BESharedState())
```

```
-- Constraint 55
-- Each RequestingInformationPin and each RespondingInformationPin
   MUST have a classifier, this classifier MUST be an
   InformationEnvelope or a subtype defined in an
   extension/specialization module.

Missing on purpose
```

```
-- Constraint 56
-- Two RequestingInformationPins which are connected using an
-- object flow MUST have the same classifier.

Missing on purpose
```

```
-- Constraint 57
-- Two RespondingInformationPins which are connected using an
-- object flow MUST have the same classifier.

Missing on purpose
```

```
-- Constraint 58
-- A BusinessCollaborationView MUST contain exactly one
   BusinessCollaborationUseCase.
context Package inv: self.isBCollaborationV()
    implies self.ownedElement
    ->select(k|k.oclAsType(UseCase).isBCollaborationUC())->size=1
```

```
-- Constraint 59
-- A BusinessCollaborationView MUST contain two to many
   AuthorizedRoles.
context Package inv: self.isBCollaborationV()
    implies self.ownedElement
    ->select(k|k.oclAsType(Actor).isAuthorizedRole())->size>=2
```

```
-- Constraint 60
-- A BusinessCollaborationUseCase MUST have two to many participates
   associations
-- to AuthorizedRoles contained in the same BusinessCollaborationView.
context UseCase inv: self.isBCollaborationUC()
    implies self.owner.ownedElement
    ->select(c|c.oclIsTypeOf(Association)
    and c.oclAsType(Association).isParticipates()
    and c.oclAsType(Association).ownedEnd.type
    ->exists(t|t.oclAsType(Actor).isAuthorizedRole()))->size>=2
```

```
-- Constraint 61
-- Each AuthorizedRole contained in the BusinessCollaborationView MUST
   have exactly
-- one participates association to the BusinessCollaborationUseCase
   included in the same BusinessCollaborationView.
context Actor inv: self.isAuthorizedRole() and
   self.owner.oclAsType(Package).isBCollaborationV()
   implies self.owner.ownedElement->
   select(as|as.oclIsTypeOf(Association))->
   select(ass| ass.oclAsType(Association).ownedEnd.type->
   exists(end|end.oclAsType(Actor)=self)
   and ass.oclAsType(Association).ownedEnd.type->
   exists(oend|oend.oclAsType(UseCase).isBCollaborationUC())) ->size=1
```

```
-- Constraint 62
-- A BusinessCollaborationUseCase MUST have one to many include
   relationships to another BusinessCollaborationUseCase
-- or to a BusinessTransactionUseCase.
context UseCase inv: self.isBCollaborationUC()
    implies self.include.addition->size>0
    and self.include.addition->
    forAll(Uc |Uc.oclAsType(UseCase).isBCollaborationUC()
    or Uc.oclAsType(UseCase).isBTransactionUC())
```

```
-- Constraint 63
```

```
-- Exactly one BusinessCollaborationProtocol MUST be placed beneath
   each BusinessCollaborationUseCase.
context Activity inv: self.isBCollaborationProtocol()
    implies self.owner.oclAsType(UseCase).isBCollaborationUC()
```

```
-- Constraint 64
-- A BusinessCollaborationProtocol MUST contain one to many
   BusinessTransactionActions and/or BusinessCollaborationAction.
context Activity inv: self.isBCollaborationProtocol()
    implies self.ownedElement->exists( actions |
    actions.oclAsType(CallBehaviorAction).isBTransactionAction()
    or actions.oclAsType(CallBehaviorAction).isBCollaborationAction())
```

```
-- Constraint 65
-- Each BusinessTransactionAction MUST call exactly one
   BusinessTransaction
context Action inv:
   self.oclAsType(CallBehaviorAction).isBTransactionAction()
    implies
   self.oclAsType(CallBehaviorAction).behavior.oclAsType(Activity).isB
   Transaction()
```

```
-- Constraint 66
-- Each BusinessTransaction called by a BusinessTransactionAction MUST
   be placed beneath a
-- BusinessTransactionUseCase which is included in the
   BusinessCollaborationUseCase that covers
-- the corresponding BusinessCollaborationProtocol.
context Activity inv: self.oclAsType(Activity).isBTransaction()
    and CallBehaviorAction.allInstances->
    exists( node | node.oclAsType(CallBehaviorAction).behavior.
    oclAsType(Activity)=self)
    implies self.owner.oclAsType(UseCase).isBTransactionUC()
    and UseCase.allInstances->
    select( uc |   uc.oclAsType(UseCase).isBCollaborationUC())->
    exists( anInclude | anInclude.oclAsType(UseCase).include.addition
    -> exists( elem |
    elem.oclAsType(UseCase)=self.owner.oclAsType(UseCase))
    and anInclude.oclAsType(UseCase).ownedElement->exists( protocol |
    protocol.oclAsType(Activity).isBCollaborationProtocol()) )
```

```
-- Constraint 67
-- Each BusinessCollaborationProtocol called by a
   BusinessCollaborationAction MUST be placed beneath a
-- BusinessCollaborationProtocolUseCase which is included in the
   BusinessCollaborationUseCase that covers the corresponding
-- BusinessCollaborationProtocol.
context CallBehaviorAction inv: let protocol : Activity =
   self.behavior.oclAsType(Activity) in self.isBCollaborationAction()
    implies protocol.owner.oclAsType(UseCase).isBCollaborationUC()
    and self.owner.owner.oclAsType(UseCase).isBCollaborationUC()
    and self.owner.owner.oclAsType(UseCase).include.addition->
```

```
    exists( inc |
   inc.oclAsType(UseCase)=protocol.owner.oclAsType(UseCase))
```

```
-- Constraint 68
-- A BusinessCollaborationProtocol MUST contain two to many
   BusinessCollaborationPartions.
context Activity inv: self.isBCollaborationProtocol()
    implies self.ownedElement->select( partitions |
   partitions.oclAsType(ActivityPartition).
   isBCollaborationPartition())->size()>=2
```

```
-- Constraint 69
-- The number of AuthorizedRoles in the BusinessCollaborationView MUST
   match the number of BusinessCollaborationPartitions
-- in the BusinessCollaborationProtocol which is placed beneath the
   BusinessCollaborationUseCase of the same BusinessCollaborationView.
context Package inv:
    let authRoles : Set(Element) = self.ownedElement->
    select( role | role.oclAsType(Actor).isAuthorizedRole())
    in let collUC : Set(Element) = self.ownedElement->
    select( bCollUC | bCollUC.oclAsType(UseCase).isBCollaborationUC())
    in let protocol : Set(Element) = collUC.ownedElement
    ->select(prot|prot.oclAsType(Activity).isBCollaborationProtocol())
    ->asSet in let partitions : Set(Element) = protocol.ownedElement
    ->select(part|part.oclAsType(ActivityPartition).
    isBCollaborationPartition())->
    asSet in self.isBCollaborationV()
    implies authRoles->size() = partitions->size()
```

```
-- Constraint 70
-- Each AuthorizedRole in the BusinessCollaborationView MUST be
   assigned to a BusinessCollaborationPartition
-- in the BusinessCollaborationProtocol which is placed beneath the
   BusinessCollaborationUseCase
-- of the same BusinessCollaborationView.
context Actor inv:
    let bCollUC : Set(Element) = self.owner.ownedElement->
    select ( uc | uc.oclAsType(UseCase).isBCollaborationUC())
    in let bCollProtocol : Set(Element) = bCollUC.ownedElement->
    select ( protocol |
    protocol.oclAsType(Activity).isBCollaborationProtocol())->asSet
    in let bCollPartition : Set(Element) =
    bCollProtocol.ownedElement -> select( partition |
   partition.oclAsType(ActivityPartition).isBCollaborationPartition())
    ->asSet in self.isAuthorizedRole() and
    self.owner.oclAsType(Package).isBCollaborationV()
    implies bCollPartition.oclAsType(ActivityPartition).represents->
    select( part | part.oclAsType(Actor)=self.oclAsType(Actor))
    ->size()=1
```

```
-- Constraint 71
-- Each BusinessCollaborationPartition MUST be classified by exactly
   one AuthorizedRole included in the same
```

```
-- BusinessCollaborationView as the BusinessCollaborationUseCase
   covering the BusinessCollaborationProtocol
-- containing this BusinessCollaborationPartition.
context ActivityPartition inv: self.isBCollaborationPartition()
    implies self.hlpOwningPackage().ownedElement->
    exists( actor | actor.oclAsType(Actor).isAuthorizedRole()
    and actor.oclAsType(Actor)=self.represents.oclAsType(Actor))
```

```
-- Constraint 72
-- A BusinessCollaborationPartition MUST be either empty or contain
   one to many NestedBusinessCollaborations.
context ActivityPartition inv: self.isBCollaborationPartition()
    implies self.owner.ownedElement->select( action |
    action.oclAsType(Action).inPartition->
    exists(element | element.oclAsType(ActivityPartition)=self))
    ->forAll( elem | elem.oclAsType(Action).isBNestedCollaboration())
    and (self.ownedElement->size()=0 or self.ownedElement->forAll(
    elem | elem.oclAsType(Action).isBNestedCollaboration()) )
```

```
-- Constraint 73
-- Each BusinessTransactionAction MUST be the target of exactly one
   InitialFlow which source MUST be a BusinessCollaborationPartition.
context CallBehaviorAction inv:
   self.oclAsType(CallBehaviorAction).isBTransactionAction()
    implies Dependency.allInstances->select( dependency |
    dependency.oclAsType(Dependency).supplier->
    exists( ends | ends.oclAsType(CallBehaviorAction)=self) and
    dependency.oclAsType(Dependency).isInitFlow()
    and dependency.oclAsType(Dependency).client->exists( end |
    end.oclAsType(ActivityPartition).isBCollaborationPartition()))->
    size()=1
```

```
-- Constraint 74
-- Each BusinessTransactionAction MUST be the source of exactly one
   InitialFlow which target MUST be either a
-- BusinessCollaborationPartition or a NestedBusinessCollaboration.
context CallBehaviorAction inv:
   self.oclAsType(CallBehaviorAction).isBTransactionAction()
    and self.owner.oclAsType(Activity).isBCollaborationProtocol()
    implies Dependency.allInstances->select( dependency |
    dependency.oclAsType(Dependency).client->
    exists( ends | ends.oclAsType(CallBehaviorAction)=self)
    and  dependency.oclAsType(Dependency).isInitFlow()
    and dependency.oclAsType(Dependency).supplier->exists(end |
    end.oclAsType(ActivityPartition).isBCollaborationPartition()
    or end.oclAsType(Action).isBNestedCollaboration()))->size()=1
```

```
-- Constraint 75
-- The InitialFlow sourcing from a BusinessTransactionAction and the
   InitialFlow targeting a BusinessTransactionAction
-- MUST NOT be targeting to / sourcing from the same
   BusinessCollaborationPartition, nor targeting to a
   NestedBusinessCollaboration
```

```
-- within the same BusinessCollaborationPartition.
context Dependency inv: self.isInitFlow()
    implies not(Dependency.allInstances->exists( dep |
   (dep.oclAsType(Dependency).supplier=self.client
    or ( dep.oclAsType(Dependency).supplier->forAll(end |
    end.oclAsType(Action).isBNestedCollaboration())
    and
    dep.oclAsType(Dependency).supplier.oclAsType(Action).inPartition
    ->includesAll(self.client.oclAsType(ActivityPartition)) ) )
    and dep.oclAsType(Dependency).client=self.supplier and
    dep.oclAsType(Dependency).isInitFlow() ))
```

```
-- Constraint 76
-- If a BusinessTransactionAction calls a two-way BusinessTransaction,
   this BusinessTransactionAction MUST be the source of exactly one
   RespondingFlow which target MUST be a
   BusinessCollaborationPartition.
context CallBehaviorAction inv: self.isBTransactionAction() and
   self.behavior.oclAsType(Activity).isTwoWayBTransaction()
   implies self.owner.ownedElement->
   select( dep | dep.oclAsType(Dependency).isReFlow() and
   dep.oclAsType(Dependency).client->
   exists( end | end.oclAsType(CallBehaviorAction)=self)
   and dep.oclAsType(Dependency).supplier->
   exists( end |
   end.oclAsType(ActivityPartition).isBCollaborationPartition()))
   ->size()=1
```

```
-- Constraint 77
-- If a BusinessTransactionAction calls a two-way BusinessTransaction,
   this BusinessTransactionAction MUST be the target of exactly one
   RespondingFlow which source MUST be either a
   BusinessCollaborationPartition or a NestedBusinessCollaboration.
context CallBehaviorAction inv: self.isBTransactionAction() and
   self.behavior.oclAsType(Activity).isTwoWayBTransaction()
   implies self.owner.ownedElement->
   select( dep | dep.oclAsType(Dependency).isReFlow()
   and dep.oclAsType(Dependency).supplier->
   exists( end | end.oclAsType(CallBehaviorAction)=self)
   and dep.oclAsType(Dependency).client->
   exists( end |
   end.oclAsType(ActivityPartition).isBCollaborationPartition()
   or end.oclAsType(Action).isBNestedCollaboration()))->size()=1
```

```
-- Constraint 78
-- The RespondingFlow sourcing from a BusinessTransactionAction and
   the RespondingFlow targeting a BusinessTransactionAction
-- MUST NOT be targeting to /sourcing from the same
   BusinessCollaborationPartition, nor targeting to a
   NestedBusinessCollaboration
-- within the same BusinessCollaborationPartition.
context Dependency inv: self.isReFlow()
    implies not(Dependency.allInstances->exists( dep |
   (dep.oclAsType(Dependency).supplier=self.client
```

```
    or dep.oclAsType(Dependency).supplier.owner.
    oclAsType(ActivityPartition)=self.client.
    oclAsType(ActivityPartition) )
    and ( dep.oclAsType(Dependency).client=self.supplier
    or dep.oclAsType(Dependency).supplier.owner.
    oclAsType(ActivityPartition)=self.supplier.
    oclAsType(ActivityPartition) )
    and dep.oclAsType(Dependency).isReFlow() ))
```

```
-- Constraint 79
-- If a BusinessTransactionAction calls a one-way BusinessTransaction,
   this BusinessTransactionAction MUST NOT be the source of a
   RespondingFlow and MUST NOT be the target of a RespondingFlow.
context CallBehaviorAction inv: self.isBTransactionAction() and
   self.behavior.oclAsType(Activity).isOneWayBTransaction()
   implies not(self.owner.ownedElement->
   exists( dep | dep.oclAsType(Dependency).isReFlow() and (
   dep.oclAsType(Dependency).client->
   exists( end | end.oclAsType(CallBehaviorAction)=self or
   dep.oclAsType(Dependency).supplier->
   exists( end | end.oclAsType(CallBehaviorAction)=self )))
```

```
-- Constraint 80
-- The RespondingFlow targeting a BusinessTransactionAction must start
   from the
-- BusinessCollaborationPartition / NestedBusinessCollaboration which
   is the target of the InitialFlow starting
-- from the same BusinessTransactionAction.
context Dependency inv: self.isReFlow()
    and self.supplier->forAll( end |
    end.oclAsType(CallBehaviorAction).isBTransactionAction())
    implies Dependency.allInstances->
    exists( flow | flow.oclAsType(Dependency).isInitFlow() and (
    flow.oclAsType(Dependency).supplier.oclAsType(ActivityPartition)=
    self.oclAsType(Dependency).client.oclAsType(ActivityPartition) or
    flow.oclAsType(Dependency).supplier.oclAsType(Action)=
    self.oclAsType(Dependency).client.oclAsType(Action) ) and
    flow.oclAsType(Dependency).client.oclAsType(CallBehaviorAction)=
    self.supplier.oclAsType(CallBehaviorAction) )
```

```
-- Constraint 81
-- The RespondingFlow starting from a BusinessTransactionAction must
   target the BusinessCollaborationPartition which is
-- the source of the InitialFlow targeting to the same
   BusinessTransactionAction.
context Dependency inv: self.isReFlow()
    and self.oclAsType(Dependency).client->forAll( end |
    end.oclAsType(CallBehaviorAction).isBTransactionAction())
    implies Dependency.allInstances->exists( flow |
    flow.oclAsType(Dependency).isInitFlow() and
    flow.oclAsType(Dependency).client.oclAsType(ActivityPartition)=
    self.oclAsType(Dependency).supplier.oclAsType(ActivityPartition)
    and flow.oclAsType(Dependency).supplier.
    oclAsType(CallBehaviorAction)=
```

```
    self.client.oclAsType(CallBehaviorAction) )
```

```
-- Constraint 82
-- A NestedBusinessCollaboration MUST be the target of exactly one
   InitialFlow.
context Action inv: self.isBNestedCollaboration()
     implies Dependency.allInstances->
     select( dep | dep.oclAsType(Dependency).supplier->
     exists(end | end.oclAsType(Action)=self.oclAsType(Action)))
     ->size()=1
```

```
-- Constraint 83
-- A NestedBusinessCollaboration MAY be the source of a
   RespondingFlow,
-- but MUST NOT be the source of more than one RespondingFlow.
context Action inv:
    self.isBNestedCollaboration()
    implies Dependency.allInstances->
    select( dep | dep.oclAsType(Dependency).isReFlow()
    and dep.oclAsType(Dependency).client->exists(end |
    end.oclAsType(Action)=self.oclAsType(Action)))->size()<=1
```

```
-- Constraint 84
-- A BusinessCollaborationAction MUST be the target of two to many
   InformationFlows
context CallBehaviorAction inv: self.isBCollaborationAction()
    implies Dependency.allInstances->select( dep |
    dep.oclAsType(Dependency).supplier->
    exists( end | end.oclAsType(CallBehaviorAction)=self))->size()>=2
```

```
-- Constraint 85
-- A BusinessCollaborationAction MUST not be the source of an
   InformationFlow.
context CallBehaviorAction inv: self.isBCollaborationAction()
    implies not(Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).client->
    exists( end | end.oclAsType(CallBehaviorAction)=self)))

```

```
-- Constraint 86
-- A BusinessCollaborationAction MUST not be the source and MUST not
   be the target of an InitialFlow.
context CallBehaviorAction inv: self.isBCollaborationAction()
    implies not(Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).isInitFlow()
    and (dep.oclAsType(Dependency).client->exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())
    or dep.oclAsType(Dependency).supplier->exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())))
```

```
-- Constraint 87
-- A BusinessCollaborationAction MUST not be the source and MUST not
   be the target of a RespondingFlow.
context CallBehaviorAction inv: self.isBCollaborationAction()
    implies not(Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).isReFlow()
    and (dep.oclAsType(Dependency).client->exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())
    or dep.oclAsType(Dependency).supplier->exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())))
```

```
-- Constraint 88
-- A BusinessTransactionAction MUST not be the source and MUST not be
   the target of an InformationFlow that is neither
-- stereotyped as InitialFlow nor as RespondingFlow.
context CallBehaviorAction inv: self.isBTransactionAction()
    implies not( Dependency.allInstances-> exists( dep |
    dep.oclAsType(Dependency).client->
    exists(end | end.oclAsType(CallBehaviorAction)=self)
    and not(dep.oclAsType(Dependency).isInitFlow()
    or dep.oclAsType(Dependency).isReFlow()) ))
    and not( Dependency.allInstances-> exists( dep |
    dep.oclAsType(Dependency).supplier->
    exists(end | end.oclAsType(CallBehaviorAction)=self)
    and not(dep.oclAsType(Dependency).isInitFlow()
    or dep.oclAsType(Dependency).isReFlow()) ))
```

```
-- Constraint 89
-- A NestedBusinessCollaboration MUST not be the source and MUST not
   be the target of an InformationFlow that
-- targets to / sources from a BusinessCollaborationAction.
context Action inv: self.oclAsType(Action).isBNestedCollaboration()
    implies not(Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).supplier->exists(end |
    end.oclAsType(Action)=self)
    and dep.oclAsType(Dependency).client->exists(end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())))
    and not(Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).client->exists(end |
    end.oclAsType(Action)=self)
    and dep.oclAsType(Dependency).supplier->exists(end|
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())))
```

```
-- Constraint 90
-- The number of InformationFlows targeting a
   BusinessCollaborationAction MUST match the number of
   BusinessCollaborationPartitions
-- contained in the BusinessCollaborationProtocol that is called by
   this BusinessCollaborationAction.
context CallBehaviorAction inv: self.isBCollaborationAction()
    implies self.owner.ownedElement->select( dep |
    dep.oclAsType(Dependency).supplier->
    exists( end | end.oclAsType(CallBehaviorAction)=self ))->size() =
    self.behavior.oclAsType(Activity).ownedElement->
```

```
    select( partitions |
    partitions.oclAsType(ActivityPartition).
    isBCollaborationPartition())->size()
```

```
-- Constraint 91
-- Either an AuthorizedRole classifying a
   BusinessCollaborationPartition that is the source of an
   InformationFlow
-- targeting a BusinessCollaborationAction MUST match an
   AuthorizedRole classifying a BusinessCollaborationPartition in the
-- BusinessCollaborationProtocol that is called by this
   BusinessCollaborationAction or the InformationFlow must be
   classified
-- by an AuthorizedRole classifying a BusinessCollaborationPartition
   in the BusinessCollaborationProtocol that is called
-- by this BusinessCollaborationAction.

context ActivityPartition inv: self.isBCollaborationPartition()
    and self.represents.oclAsType(Actor).isAuthorizedRole()
    and Dependency.allInstances->exists( dep |
    dep.oclAsType(Dependency).client->exists( end |
    end.oclAsType(ActivityPartition)=self)
    and dep.oclAsType(Dependency).supplier-> exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction()))
    implies Dependency.allInstances->select( dep |
    dep.oclAsType(Dependency).client->
    exists( end | end.oclAsType(ActivityPartition)=self)
    and dep.oclAsType(Dependency).supplier-> exists( end |
    end.oclAsType(CallBehaviorAction).isBCollaborationAction())) ->
    forAll(dependency | dependency.oclAsType(Dependency).supplier->
    exists(end|end.oclAsType(CallBehaviorAction).behavior.
    oclAsType(Activity).ownedElement->
    exists( partition |  partition.oclAsType(ActivityPartition).
    isBCollaborationPartition()
    and partition.oclAsType(ActivityPartition).represents.
    oclAsType(Actor)=self.represents.oclAsType(Actor))))
```

```
-- Constraint 92
-- A BusinessRealizationView MUST contain exactly one
   BusinessRealizationUC, two to many AuthorizedRoles,
-- and two to many participates associations.
context Package inv: let ownEl : Set(Element) = self.ownedElement in
   self.isBRealizationV()
    implies ownEl -> select ( ass |
    ass.oclAsType(Association).isParticipates())->size>=2
    and ownEl -> select ( bRealUC |
    bRealUC.oclAsType(UseCase).isBRealizationUC()
    and not(bRealUC.oclAsType(UseCase).isBCollaborationUC()))
    ->size()=1 and ownEl
    -> select ( roles |roles.oclAsType(Actor).isAuthorizedRole())
    ->size()>=2
```

```
-- Constraint 93
```

```
-- A BusinessRealization MUST be associated with two to many
   AuthorizedRoles via stereotyped binary participates associations.
context UseCase inv: self.isBRealizationUC() and
   not(self.isBCollaborationUC())
    implies self.owner.ownedElement->
    select(a| a.oclAsType(Association).isParticipates()
    and a.oclAsType(Association).ownedEnd.type
    ->exists(t|t.oclAsType(Actor).isAuthorizedRole())
    and a.oclAsType(Association).ownedEnd.type
    ->exists(t|t.oclAsType(UseCase)=self) )->size()>=2
```

```
-- Constraint 94
-- A BusinessRealization MUST be the source of exactly one realization
   dependency to a BusinessCollaborationUseCase.
context UseCase inv: self.isBRealizationUC()
    implies self.owner.ownedElement->exists( ass |
    ass.oclAsType(Realization).isRealizes()
    and ass.oclAsType(Realization).client->exists(uc |
    uc.oclAsType(UseCase)=self)
    and ass.oclAsType(Realization).supplier->exists(uc |
    uc.oclAsType(UseCase).isBCollaborationUC()))
```

```
-- Constraint 95
-- A BusinessRealization MUST NOT be the source or target of an
   include or extends association.
context UseCase inv: self.isBRealizationUC()
    implies self.include->size()=0
    and UseCase.allInstances
    ->forAll(k|k.oclAsType(UseCase).include.addition
    ->forAll(s|not(s.oclAsType(UseCase)=self)))
    and UseCase.allInstances
    ->forAll(u|u.oclAsType(UseCase).extend.extendedCase
    ->forAll(t|not(t.oclAsType(UseCase)=self)))
    and self.extend->size=0
```

```
-- Constraint 96
-- All dependencies from/to an AuthorizedRole must be stereotyped as
   mapsTo.
context Package inv: self.isBRealizationV()
    implies self.ownedElement->select( dep |
   dep.oclAsType(Dependency).client->
   exists( k | k.oclAsType(Actor).isAuthorizedRole() )
   and dep.oclAsType(Dependency).supplier->
   exists( k | k.oclAsType(Actor).isAuthorizedRole()))->
   forAll(dep| dep.oclAsType(Dependency).isMapsTo())
```

```
-- Constraint 97
-- An AuthorizedRole, which participates in a BusinessRealization,
   must be the target of exactly one mapsTo dependency
-- starting from a BusinessPartner.  Furthermore the AuthorizedRole,
   which participates in the BusinessRealization must
-- be the source of exactly one mapsTo dependency targeting an
   AuthorizedRole participating in a BusinessCollaborationUseCase.
```

```
context Actor inv: self.isAuthorizedRole()
    and self.owner.ownedElement ->
    exists( ass | ass.oclAsType(Association).isParticipates()
    and ass.oclAsType(Association).ownedEnd.type->exists( end |
    end.oclAsType(Actor)=self)
    and ass.oclAsType(Association).ownedEnd.type->exists( end |
    end.oclAsType(UseCase).isBRealizationUC()
    and not(end.oclAsType(UseCase).isBCollaborationUC())))
    implies self.owner.ownedElement->select( dep |
    dep.oclAsType(Dependency).isMapsTo()
    and dep.oclAsType(Dependency).client
    ->exists(t|t.oclAsType(Actor).isBPartner()
    and not(t.oclAsType(Actor).isAuthorizedRole())))-> select(dep |
    dep.oclAsType(Dependency).supplier->
    exists(t|t.oclAsType(Actor)=self))-> size()=1 and
    self.oclAsType(Actor).hlpMapsToAuthRoleParticipates()
```

```
-- Constraint 98
-- AuthorizedRoles in a BusinessRealizationView must have a unique
   name within the scope of the package, they are located in
context Package inv:
    let authRoles : Set(Element) = self.ownedElement->select( roles |
    roles.oclAsType(Actor).isAuthorizedRole())
    in self.isBRealizationV() implies authRoles->size() =
    authRoles->collect(p|p.oclAsType(Actor).name)->asSet->size()
```

```
-- Constraint 99
--  The number of AuthorizedRoles participating in a
   BusinessCollaborationUseCase MUST match the number of
   AuthorizedRoles participating in the BusinessRealization realizing
   this BusinessCollaborationUseCase
context Package inv:
    let ass : Set(Element) = self.ownedElement->select( ass |
    ass.oclAsType(Association).isParticipates())
    in self.isBRealizationV()
    implies ass->select( ends
    |ends.oclAsType(Association).ownedEnd.type->exists( auth |
    auth.oclAsType(Actor).isAuthorizedRole())
    and ends.oclAsType(Association).ownedEnd.type->exists( uc |
    uc.oclAsType(UseCase).isBRealizationUC())) ->
    size() = ass->select( ends
    |ends.oclAsType(Association).ownedEnd.type->exists( auth |
    auth.oclAsType(Actor).isAuthorizedRole())
    and ends.oclAsType(Association).ownedEnd.type->exists( uc |
    uc.oclAsType(UseCase).isBCollaborationUC())) -> size()
```

```
-- Constraint 100
-- A BusinessInformationView MUST contain one to many
   InformationEnvelopes or subtypes thereof defined in any other
-- extension/specialization module. Furthermore, it MAY contain any
   other document modeling artifacts.
context Class inv: self.isBInformationV()
    implies self.ownedElement->exists( doc | doc.isInfEnvelope())
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bCollMode
def:
    let: isBCollModel() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bCollModel)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bCollaborationV
def:
    let: isBCollaborationV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bCollaborationV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bRealizationV
def:
    let: isBRealizationV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bRealizationV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bTransactionV
def:
    let: isBTransactionV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bTransactionV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bChoreographyV
def:
    let: isBChoreographyV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bChoreographyV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bInformationV
def:
    let: isBInformationV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bInformationV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bRequirementsV
def:
    let: isBRequirementsV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bRequirementsV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bDomainV
def:
    let: isBDomainV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bDomainV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bPartnerV
def:
    let: isBPartnerV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bPartnerV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bEntityV
def:
    let: isBEntityV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bEntityV)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bArea
def:
    let: isBArea() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bArea)
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as ProcessArea
def:
    let: isProcessArea() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__ProcessArea)
```

```
-- Convenience method:
-- Evaluates if a use case is stereotyped as bProcessUC
def:
    let: isBProcessUseCase() : Boolean =
    self.oclAsType(UseCase).extension_bProcessUC.isDefined
```

```
-- Convenience method:
-- Evaluates if an activity is stereotyped as bProcess
def:
    let: isBProcess() : Boolean =
    self.oclAsType(Activity).extension_bProcess.isDefined
```

```
-- Convenience method:
-- Evaluates if an actor is stereotyped as bPartner
```

```
def:
    let: isBPartner() : Boolean =
    self.oclAsType(Actor).extension_Stakeholder.
    oclIsKindOf(UMM2_Profile__bPartner)
```

```
-- Convenience method:
-- Evaluates if an association is stereotyped as participates
def:
    let: isParticipates() : Boolean =
    self.oclAsType(Association).extension_participates.isDefined
```

```
-- Convenience method:
-- Evaluates if an actor is stereotyped as stakeholder
def:
    let: isStakeholder() : Boolean =
    self.oclAsType(Actor).extension_Stakeholder.isDefined
```

```
-- Convenience method:
-- Evaluates if a dependency is stereotyped as isOfInterestTo
def:
    let: isOfInterestTo() : Boolean =
    self.oclAsType(Dependency).extension_isOfInterestTo.isDefined
```

```
-- Convenience method:
-- Evaluates if an actor is stereotyped as authorized role
def:
    let: isAuthorizedRole() : Boolean =
    self.oclAsType(Actor).extension_AuthorizedRole.isDefined
```

```
-- Convenience method:
-- Evaluates if an use case is stereotyped as BCollaborationUC
def:
    let: isBCollaborationUC() : Boolean =
    self.oclAsType(UseCase).extension_bProcessUC.
    oclIsKindOf(UMM2_Profile__bCollaborationUC)
```

```
-- Convenience method:
-- Evaluates if a class is stereotyped as bEntity
def:
    let: isBEntity() : Boolean =
    self.oclAsType(Class).extension_bEntity.isDefined
```

```
-- Convenience method:
-- Evaluates if a package is stereotyped as bDataV
def:
    let: isBDataV() : Boolean =
    self.oclAsType(Package).extension_bLibrary.
    oclIsKindOf(UMM2_Profile__bDataV)
```

```
-- Convenience method:
-- Evaluates if an use case is stereotyped as bTransactionUC
def:
    let: isBTransactionUC() : Boolean =
    self.oclAsType(UseCase).extension_bProcessUC.
    oclIsKindOf(UMM2_Profile__bTransactionUC)
```

```
-- Convenience method:
-- Evaluates if a modeling is an activity partition
def:
    let: isActivityPartition() : Boolean =
    self.oclIsTypeOf(ActivityPartition)
```

```
-- Convenience method:
-- Evaluates if an action is stereotyped as bProcessAction
def:
    let: isBProcessAction() : Boolean =
    self.oclAsType(Action).extension_bProcessAction.isDefined
```

```
-- Convenience method:
-- Evaluates if an use case is not stereotyped as bTransactionUC
def:let: isNotBTransactionUC() : Boolean =
    not(self.oclAsType(UseCase).isBTransactionUC())
```

```
-- Convenience method:
-- Evaluates if an activity is stereotyped as bCollaborationProtocol
def:let: isBCollaborationProtocol() : Boolean =

   self.oclAsType(Activity).extension_bCollaborationProtocol.isDefined
```

```
-- Convenience method:
-- Evaluates if a call behavior action is stereotyped as
   bTransactionAction
def:
    let: isBTransactionAction() : Boolean =

   self.oclAsType(CallBehaviorAction).extension_bTransactionAction.isD
   efined
```

```
-- Convenience method:
-- Evaluates if a call behavior action is stereotyped as
   bCollaborationAction
def:
    let: isBCollaborationAction() : Boolean =
    self.oclAsType(CallBehaviorAction).
    extension_bCollaborationAction.isDefined
```

```
-- Convenience method:
-- Evaluates if a class is stereotyped as InfEnvelope
```

```
def:
    let: isInfEnvelope() : Boolean =
    self.oclAsType(Class).extension_InfEnvelope.isDefined
```

```
-- Convenience method:
-- Evaluates if an object node is stereotyped as bESharedState
def:
    let: isBESharedState() : Boolean =
    self.oclAsType(ObjectNode).extension_bESharedState.isDefined
```

```
-- Convenience method:
-- Evaluates if a state is stereotyped as bEState
def:
    let: isBEState() : Boolean =
    self.oclAsType(State).extension_bEState.isDefined
```

```
-- Convenience method:
-- Evaluates if an activity is stereotyped as bTransaction
def:
    let: isBTransaction() : Boolean =
    self.oclAsType(Activity).extension_bTransaction.isDefined
```

```
-- Convenience method:
-- Evaluates if an activity partition is stereotyped as bTPartition
def:
    let: isBTPartition() : Boolean =
    self.oclAsType(ActivityPartition).extension_bTPartition.isDefined
```

```
-- Convenience method:
-- Evaluates if an action is stereotyped as ReqAction
def:
    let: isReqAction() : Boolean =
    self.oclAsType(Action).extension_BusinessAction.
    oclIsKindOf(UMM2_Profile__ReqAction)
```

```
-- Convenience method:
-- Evaluates if an action is stereotyped as ResAction
def:
    let: isResAction() : Boolean =
    self.oclAsType(Action).extension_BusinessAction.
    oclIsKindOf(UMM2_Profile__ResAction)
```

```
-- Convenience method:
-- Evaluates if an use case is stereotyped as bRealizationUC
def:
    let: isBRealizationUC() : Boolean =
    self.oclAsType(UseCase).extension_bRealizationUC.isDefined
```

```
-- Convenience method:
-- Evaluates if a dependency is stereotyped as realizes
def:
    let: isRealizes() : Boolean =
    self.oclAsType(Realization).extension_realizes.isDefined
```

```
-- Convenience method:
-- Evaluates if a depedency is stereotyped as mapsTo.
def:
    let: isMapsTo() : Boolean =
    self.oclAsType(Dependency).extension_mapsTo.isDefined
```

```
-- Convenience method for evaluating if there is a mapsTo dependency
   from an authorized role
-- that leads to another authorized role participating in a business
   collaboration use case
def:
    let: hlpMapsToAuthRoleParticipates() : Boolean =
    self.owner.ownedElement->
    select( dep | dep.oclAsType(Dependency).isMapsTo()
    and dep.oclAsType(Dependency).client->
    exists( t | t.oclAsType(Actor)=self)
    and dep.oclAsType(Dependency).supplier->
    exists(t | t.oclAsType(Actor).isAuthorizedRole()
    and t.oclAsType(Actor).hlpParticipatesBCollaborationUC() ))
    ->size()=1
```

```
-- Convenience method for evaluating if a certain authorized role
   participates in a business
-- collaboration use case
def:
    let: hlpParticipatesBCollaborationUC() : Boolean =
    self.owner.ownedElement->select( ass |
    ass.oclAsType(Association).isParticipates()
    and ass.oclAsType(Association).ownedEnd.type->exists( end |
    end.oclAsType(Actor)=self)
    and ass.oclAsType(Association).ownedEnd.type->exists( end |
    end.oclAsType(UseCase).isBCollaborationUC()) )->size()=1
```

```
-- Convenience method:
-- Evaluates if an activity partition is stereotyped as
   bCollaborationPartition
def:
    let: isBCollaborationPartition() : Boolean =
    self.oclAsType(ActivityPartition).
    extension_bCollaborationPartition.isDefined
```

```
-- Convenience method:
-- Evaluates if a business transaction MUST have an responding
   information pin
```

```
-- (i.e., is a two-way transaction) according to its business
   transaction pattern
def:
    let: hlpMustHaveResInfPin() : Boolean =
    self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='RequestResponse'
    or self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='QueryResponse'
    or self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='RequestConfirm'
    or self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='CommercialTransaction'
```

```
-- Convenience method:
-- Evaluates if a business transaction MUST NOT have an responding
   information pin
-- (i.e., is an one-way transaction) according to its business
   transaction pattern
def:
    let: hlpMustNotHaveResInfPin() : Boolean =
    self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='Notification'
    or self.oclAsType(Activity).extension_bTransaction.
    businessTransactionType='InformationDistribution'
```

```
-- Convenience method:
-- Evaluates if an action is stereotyped as bNestedCollaboration
def:
    let: isBNestedCollaboration() : Boolean =
    self.oclAsType(Action).extension_bNestedCollaboration.isDefined
```

```
-- Convenience method:
-- Evaluates if a dependency is stereotyped as InitFlow
def:
    let: isInitFlow() : Boolean =
    self.oclAsType(Dependency).extension_initFlow.isDefined
```

```
-- Convenience method:
-- Evaluates if a dependency is stereotyped as ReFlow
def:
    let: isReFlow() : Boolean =
    self.oclAsType(Dependency).extension_reFlow.isDefined
```

Copyright Statement