Treasury Board of Canada Secretariat
Government Strategic Reference Model (GSRM)
GSRM Meta model – Extension Module to UMM Foundation
Version 2.0

2007-04-23

Table of Contents

Table of Figures

1. About this Document

1.1. Status of this Document

This Document is currently in Final – Pre-Implementation Verfication stage. As of the time of this writing, there are still work items to be completed. A non-comprehensive list of items includes:

- Development of the complete set of tag items to accompany the meta-model. These will be developed in accordance and parallel to the item above
- A set of worksheets which value add to the development of Core Components and Message schemas as an output of the business modeling exercise.

Any comment to this document – remark and suggested solution – should be sent to the following e-mail address:

Buchinski.Ed@tbs-sct.gc.ca

1.2. Document Context

The GSRM Meta Model is a single functional Module, extending the UMM Foundation module for Government of Canada implementation. The following modules have been defined by UN/CEFACT to structure meta-modules:

- Base: Covers the fundamental principles that are shared across all the other modules.
- Foundation: Includes the core concepts of the GSRM. Defines all the concepts that are used as part of the minimal methodology to produce a GSRM compliant business collaboration module.
- Specialization: Multiple specialization modules might define add-on concepts to the foundation. Each specialization module addresses a specialized type of analysis that extends the foundation module at a well-defined extension point for a certain topic. Specialization modules might become candidates for later inclusion into the foundation module.
- **Extension (GSRM 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 (e.g. Tag definitions, enumerated lists, etc.) that are created and maintained by external organizations. (E.g. GSRM was created and is being maintained by Treasury Board of Canada Secretariat).

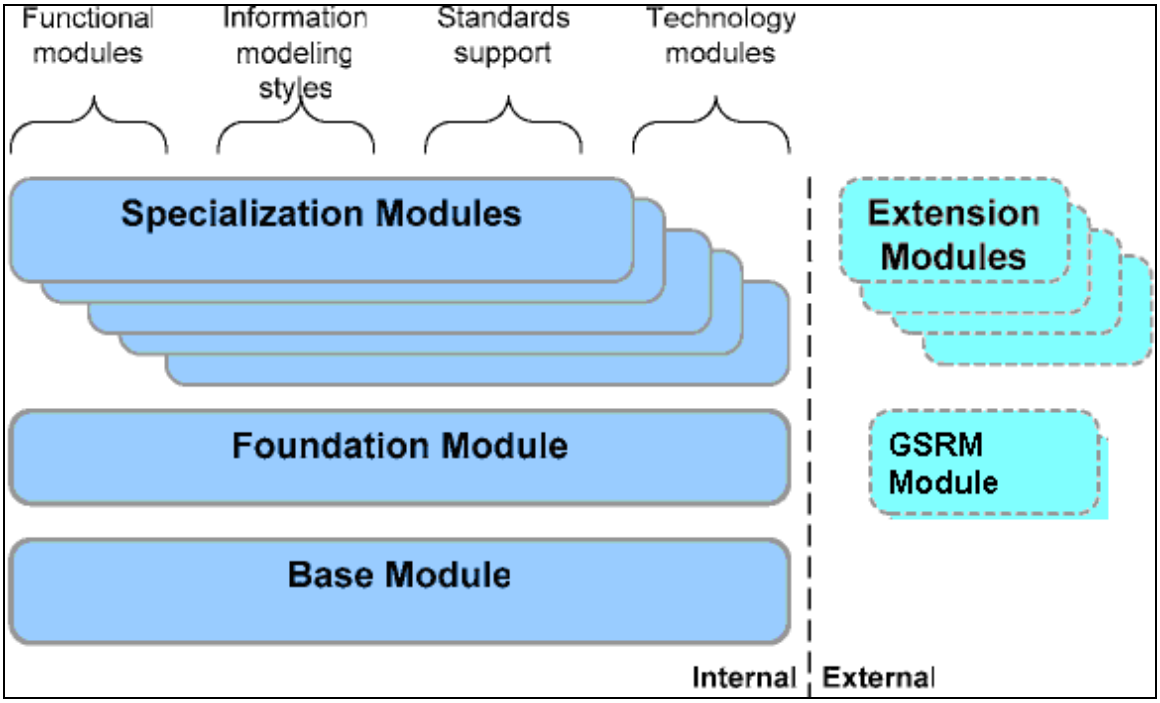

Figure 1 - Module Structure of the UMM Meta model

1.3. Extension Notes

Items that have been modified from the original UMM Foundation Module will be highlighted in Yellow within the Diagram types. Additional tag items will be described within the respective section of the extension module.

1.4. Revision history

Version	Release	Date	Comment
Candidate for 1.0	First Working Draft	2006-01-14	BDV only for review by TBS
Candidate for 1.0	First Working Draft	2006-02-17	Updated BDV plus draft BRV
Candidate for 1.0	First Working Draft	2006-03-06	Revised BDV plus initial version of BRV and BTV
Candidate for 1.0	First Working Draft	2006-03-31	Completed Example diagrams, formatting, etc.
Candidate for 2.0	First Working Draft	2007-03-01	Additional Meta Model items for: • Management, Resources and Results Structure • Instructions for Reports on Prioirities and Plans • Profile of Internaal Services • Program Activity Architecture • Measuring outcomes and outputs, so that performance measurement metrics can be created.
Candidate for 2.0	2nd Working Draft	2007-04-11	Additional examples inserted. Various inconsistencies corrected.
Candidate for 2.0	3rd Working Draft	2007-04-18	OCL constraints inserted
Version 2.0	Final text	2007-04-23	

2. Project Team

2.1. Contact

Name:	Edwin Buchinski
Company:	Treasury Board of Canada Secretariat Government of Canada
Street:	6th Floor, 2745 Iris Street,
City, Province:	Ottawa Ontario
Postal Code:	K1A 0R5
Nation:	Canada
Phone:	+001 613 957-2497

2.2. Disclaimer

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

2.3. Project Team Participants

Project Lead:	Edwin Buchinski	Treasury Board of Canada Secretariat
Editor:	Glenn Miller	Northey, Smith and Associates
Contributors:	Christopher Desmarais	Northey, Smith and Associates
	John-Marc Desmarais	Northey, Smith and Associates
	Debbie Jones	Treasury Board of Canada Secretariat

3. Introduction

3.1. Audience

A reader of this document MUST have a deep understanding of UML 1.4. He or she MUST be able to understand meta-models denoted as UML class diagrams; he or she MUST also be familiar with the United Nations Modeling Methodology (UMM). He or she SHOULD be familiar the UML 1.4 meta-model, at least he or she MUST be able to check back with the UML 1.4 meta-model. The reader SHOULD be familiar with OCL 2.0 in order to understand the OCL constraints of this GSRM profile – those who are not familiar with OCL are provided with plain text descriptions of the constraint. In addition, an understanding of the following items would be helpful:

- Goals of the Government of Canada Business Transformation Enablement Program (BTEP) would be helpful.
- Management Resources Results Structure
- Expenditure Management Information System
- Program Activity Architecture

The information described in this manual is aimed at:

- Advanced business process modelers that check a UML model for GSRM compliance (if not supported by a tool).
- Advanced business process modelers that train other business process modelers and business process analysts.
- Software designers who want to produce compliant UML tools providing support for this GSRM foundation module.
- Software designers who want to produce tools to transform GSRM compliant business collaboration models into specifications of the IT-Layer (ebXML, Web Services, UN/EDIFACT).
- Software designers that want to produce repositories to register GSRM compliant collaboration models.
- Software designers that want to produce GSRM compliant work products, which are created and used by Business Analysts.

3.2. Related Documents

- UN/CEFACT;
  UN/CEAFCT Open Development Process
  http://www.unece.org/cefact/cf_plenary/plenary05/cf_05_05e.pdf
- International Organization for Standardization (ISO)
  Open-edi Reference Model. ISO/IEC 14662
  http://standards.iso.org/ittf/PublicalyAvailableStandards/c037354_ISO_IEC_14662_2004(E).zip
- Object Management Group (OMG)
  Unified Modeling Language Specification (UML), Version 1.4.2
  Http://www.omg.org/docs/formal/04-07-02.pdf

- Object Management Group (OMG)
  Object Language Constraint Language Specification 2.0
  http://www.omg.org/docs/ptc/03-10-14.pdf
- UN/CEFACT Techniques and Methodologies Group (TMG)
  UN/CEFACT's Modeling Methodology (UMM) – UMM Meta Model – Core Module + Foundation Module. Version 1.0
  http://www.untmg.org/...
- Treasury Board of Canada Secretariat
  The UN Modeling Methodology within a Government of Canada Business Architecture.
- Treasury Board of Canada Secretariat
  The Management, Resources and Results Structure Policy: Instructions to Department for Developing a Management, Resources, and Result Structure.
- Treasury Board of Canada Secretariat
  Template Instructions for Reports on Plans and Prioirities
- Treasury Board of Canada Secretariat
  Government of Canada Profile of Internal Services. Version 1.1 (Discussion Draft). November 9, 2006

3.3. Overview

3.3.1. Introduction to UN/CEFACT's Modeling Methodology (UMM)

A primary objective of UN/CEFACT is to develop a modeling methodology that can be used to capture the business knowledge that enables the development of low cost software components by software vendors to help the small and medium size companies, and to enable emerging economies to engage in e-Business practices. Thus, UN/CEFACT's Modeling Methodology focuses on developing business process and information models in a technology neutral manner. This approach provides insurance against 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 Operation View (BOV) and not the Functional Service View (FSV) as defined in ISO/IEC 14662. The BOV is defined as "a perspective of business transaction 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 requirements are reflected in the choreography of the inter-organizational business process and its information exchanges. As such, UMM provides a procedure for specifying (modeling), in a technology-neutral, implementation-independent manner, business processes involving information exchange.

Version 1.0 UMM consists of three views, each covering a set of well defined artifacts:

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

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

Business Requirements View (BRV): the goal of the BRV is to identify collaborative business processes between different business partners and to describe the requirements regarding these collaborative business processes. In order to identify collaborative business processes the static descriptions of the internal business processes discovered in the BDV are described in more detail and are analyzed regarding their dynamic behavior

and their relationship to each other. Based on this analysis the relevant "real-world" concepts in the domain are identified and described as business entities and the requirements on collaboration are identified and described as *business collaboration use cases* and *business transaction use cases.*

Business Transaction View (BTV): The BTV represents the view of the business process analyst who transforms the requirements into choreography of information exchanges. Currently, the overall choreography of business collaboration is defined by an activity graph called a B*usiness Collaboration Protocol*. In future versions, other alternatives might be developed. The business collaboration protocol choreographs the flow among business interactions. This flow depends on the states of business entities.

Currently, a business interaction is always defined by a business transaction, other alternatives might be developed in future versions. A business transaction defines a straight choreography of exchanging business information between two business partners and an optional response. An execution of a business transaction usually results in the change of state of one or more business entities. Thus, the information exchanged in a transaction should be limited to the minimum information needed to change the state of a business entity. Nevertheless, UMM allows the definition of an information exchange in a document-centric approach – even if this is not recommended. A business transaction leads to synchronized states of the business objects at both partners participating in a business transaction. Inasmuch, a business transaction is a unit of work that may or may not be successful, but is allowed to be deemed "failed" under specific conditions, by the initiator.

3.3.2. Introduction to the Government Strategic Reference Model

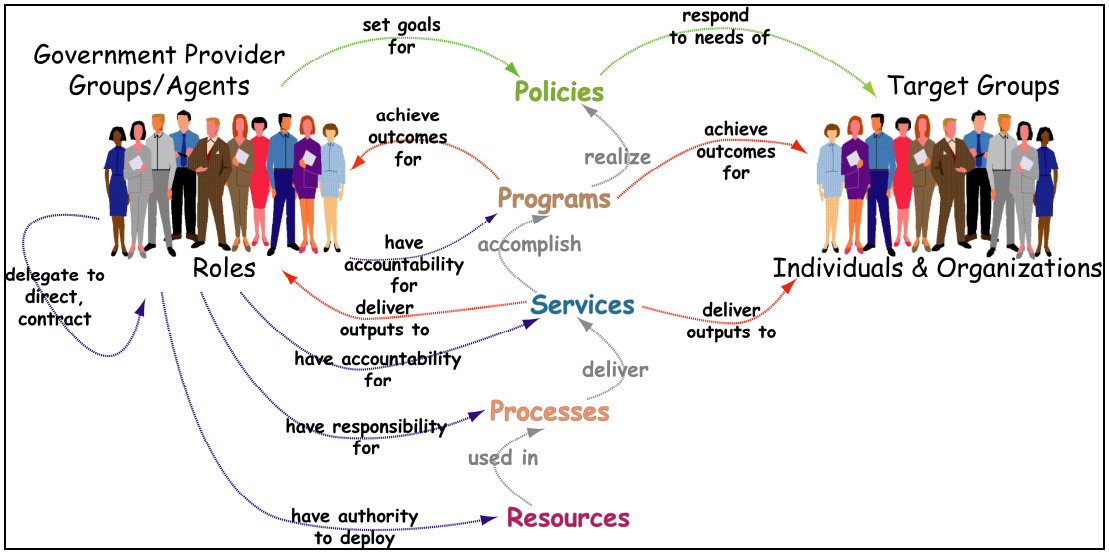

Figure 2 – Terminology of Government

Figure 2 presents some of the terminology of the business of Government. This model, which underlies the GSRM, identifies the fundamental "*Business of Government*" concepts and their interrelationships. As a semantic model, it provides a means of identifying the business concepts that provide context and backdrop to other concepts. For example, by identifying that *PROCESSES* deliver *SERVICES*, the model indicates that

SERVICES (including any models that describe or integrate services) provide a context for *PROCESSES*.

The following defines the concepts presented in the figure:

Policy	A set of regulations that inform and drive the design of an organization as well as governing how it functions. A *POLICY* is realized by *PROGRAMS* achieving their Outcomes.
Program	An accountable mandate to address recognized needs of eligible *TARGET GROUPS*. Each *PROGRAM* aims to achieve specific desirable trends in the level of a target group need (Outcome) for members of a target group. A *PROGRAM* achieves this by designing service outputs that will contribute to the Outcomes desired for the *PROGRAM*. *PROGRAMS* achieving their Outcomes realize *POLICIES*.
Service	A means, administered by a *PROGRAM*, of producing a final valued output (i.e., service output) to address one or more *TARGET GROUP* Needs. The Service Outputs produced by *SERVICES* contribute to a *PROGRAM* achieving its Outcomes. *PROCESSES* create Process Outputs that deliver a *SERVICE*.
Process	The work required to use Process Inputs to transform inputs into a Process Output. When inputs are supplied and a process is performed, a Process Output is produced and delivered. Many processes produce intermediate outputs (e.g. an application form or inspection) on the way to the final valued output (Service Output).
Target Group	A subset of the general population that exhibits a designated Need (lack of something requisite, desirable, or deemed a basic necessity to an individual or a collective). A *TARGET GROUP* is acknowledged and defined in *POLICY* as the intended primary beneficiary of the efforts of a *PROGRAM*. That is, it is a *TARGET GROUP* whose level of Need the *PROGRAM* is intended to change. A *TARGET GROUP* is the ultimate beneficiary of a Service Recipient (the party that directly receives or experiences, either willingly or begrudgingly, the output of a *SERVICE*) receiving or experiencing the outputs generated by a service. In some cases, this beneficiary is different than the recipient.
Organizations	Government employees or organizations (or agents that act on behalf of the government) that have the responsibility to contribute to a *RESOURCEPOOL*, participate in *PROCESSES*, have accountability for the effective and efficient delivery of *SERVICES*, have accountability for achieving the outcomes identified for a *PROGRAM*, or set goals for a *POLICY*.
Service Inputs	Financial and/or Personnel assets that will be consumed by a Process to create a Service Output.
Service Outputs	Tangible results of the performance of a Service.
Service Output Type	The GSRM currently identifies nineteen Service Output Types. A Service Output Type is a grouping of similar Service Outputs that share a similar pattern of Processes to produce and deliver each

	Service Output in the group as well as a distinct pattern of performance metrics.
Resource Pool	A pool of financial and/or Personnel assets that are made available by an Organization; so that those assets can be used to create Service Inputs.
Process Inputs	Assets available to a Process, so that the Process can create a Process Output.
Process Outputs	The expected results of a Process.
Need	A need is defined as a lack of something requisite, desireable, or deemed a basic necessity to an individual or a collective. Target Groups have a Need.

3.4. Objectives

3.4.1. Goals of the Technical Specification

The goal of this specification is:

- To define a set of data types that may be mapped from the UMM Foundation module to the GSRM Extension module.
- To define the semantics of well-formed GSRM Business Collaboration Models
- To define the validation rules for GSRM compliant Business Collaboration Models
- To clarify the basic concepts that a GSRM –compliant Business Collaboration Model is based on.
- To provide an unambiguous definition for GSRM Business Collaboration Models that allows an unambiguous mapping to artifacts for deployment in a Service-Oriented Architecture.
- To define an interoperable profile, based on the GSRM Meta Model, which allows UML tool vendors to customize their tools to be GSRM compliant. Better tool support will lead to a growning GSRM user base.

3.4.2. Requirements

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

- The GSRM Foundation module contains stereotypes that are currently used in the UMM Base Module or UMM Foundation module, as well as additional Stereotypes that have been created or modified to support the GSRM implementation.
- Today, the UML is the most commonly supported modeling language by modeling tools. In order to use a broad range of tools, a UMM Extension module must be based on a valid UML model. Thus, the GSRM module is based on the UML meta-model, as well as the UMM Base and Foundation Modules. In fact, it

provides a UML Profile consisting of stereotypes, tag definitions and OCL constraints.

3.4.3. Refinement and Extensions to UMM 1.0

3.4.3.1. Business Domain View

GSRM is considerably more complex than UMM, but still shares the same general skeleton: a collection of packages that contain a description of processes and actors and their interrelations. However, the BDV in UMM is meant to show the containment and reuse of processes; it is not designed to capture any explanation of why processes are put together. Several modifications were made to allow GSRM to capture the requirements of the model.

The UMM concepts of *BusinessArea* and *ProcessArea* are mapped to the similar concepts in GSRM, represented by two classification schemes named Public Program Fields and Service/Output Types Similar to the UMM, GSRM uses direct aggregation to capture the containment relationship between *BusinessArea* and *ProcessArea.*
In GSRM a *Policy* will identify a need that is met by an *Outcome* of a *Program* that infers the *Program* is part of the *Policy.*

The UMM concept of *BusinessProcess* was mapped to *Service* and *Process* in GSRM. The UMM containment relationship between *ProcessArea* and *Process* was modified in GSRM; instead of a direct containment relationship there is an indirect relationship between *Program* and *Process*. A *Program* has a *DirectOutcome* that has a yield relationship from a *ServiceOutput* of a *Service* and this chain of relationship infers that a *Service* is part of a *Program*.

UMM allows a BusinessProcess to aggregate another BusinessProcess and that relationship is maintained in GSRM. However, a 2nd set of relationships was added to the GSRM to show the relationship between the outputs of processes and services. GSRM uses a set of *yield* relationships to allow it to capture how resources (captured as *ServiceInput*s, *ServiceOutput*s, *ProcessInput*s, and *ProcessOutputs* and associated *YearlyBudget*s) are fed through the model.

UMM also has a simple description of actors in the model. A *Stakeholder* has an interest in the model, and a *BusinessPartner* is a participant in a process. GSRM extends these objects by narrowing down the types of stakeholders and participants. GSRM derives *TargetGroup* and *Organization* from the UMM *Stakeholder*, as two groups that are interested in a *Policy*. Further, the relationship between *TargetGroup* and *Policy* isn't a simple relationship, a *TargetGroup* is indirectly related to a *Policy* because it has a *Need* that is being met by the *Policy*.

GSRM also extends the concept of *BusinessPartner* to *ServiceProvider*, *ServiceRecipient* and *Partner*. *ServiceProvider* and *ServiceRecipient* are special derivations of *BusinessPartner* that is designed to capture special actor relationships between services and their actors. *ServiceProvider* is directly related as actors, but *ServiceRecipient*s are indirectly related because they receive a *ServiceOutput* generated by the *Service*.

The GSRM concept of *Partner* is almost a perfect match with the UMM concept of *BusinessPartnerType*. GSRM contains its own *Partner* definition (rather than simply reusing the UMM *BusinessPartnerType*) to isolate it from any changes in the UMM model; *Partner* should be able to derive from any similar UMM concept without requiring significant changes to GSRM.

3.4.3.2. Business Requirements View

Most of the objects in the *BusinessRequirmentsView* are directly related to similar UMM objects. The concepts of collaboration realization, collaboration use case, and transaction use case are embedded in GSRM in the same way. The majority of objects in this layer are directly transposed into GSRM from UMM; with the understanding that GSRM *Service* is always a collaboration realization; almost all the objects in this layer are related to their UMM counterparts.

There is one significant exception though; UMM provides insufficient support for very complex processes. The only activity graph described by the UMM standard is the *BusinessProcessActivityModel* which lacks reasonable scalability: the *BusinessProcessActivityModel* of a CollaborationUseCase will contain the entire *BusinessProcessActivityModel* information of all of its subordinate processes. If there are several layers of process usage then a *BusinessProcessActivityModel* can contain far too much information to be useful. A new object was introduced that has greater scalability. The *ActivityTransitionGraph* is a scalable solution for describing process flow for collaboration use cases. When used in association with the *CollaborationRequirementsView*, it captures the same information that is assembled in the *BusinessProcessActivityModel*, but without the scalability problems.

3.4.3.3. BusinessTransactionView

The *BusinessTransactionView* underwent only one (albeit large) change; the *BusinessChoreographyView* was removed. The justification for removing this object was that it is only created for collaboration use cases, and every collaboration use case will have an *ActivityTransitionGraph* in the BusinessRequirementsView. In all cases, the *BusinessChoreographyView* would be identical to the *ActivityTransitionGraph*. Since the information captured in the *ActivityTransitionGraph* (i.e. BusinessChoreographyView) is in better alignment with the goal of the *BusinessRequirementsView* (to identify requirements of business processes and interaction of processes and actors), than in the *BusinessTransactionView* (define the choreography of information exchange), it was renamed and moved to the *BusinessRequirementsView.*

3.4.4. Caveats and Assumptions

This specification makes the following assumptions:

- This UML profile is based on the UML meta-model version 1.4.2. This version is the current ISO standard version. Using another UML meta-model as a basis for the development of a UMM and/or GSRM compliant business collaboration model will not deliver correct results.
- As UMM version 2.0 will be restructuring and realigning certain models and artifacts, some of the changes noted above are in anticipation of those changes within UMM 2.0.

3.5. Structure of the GSRM Extension

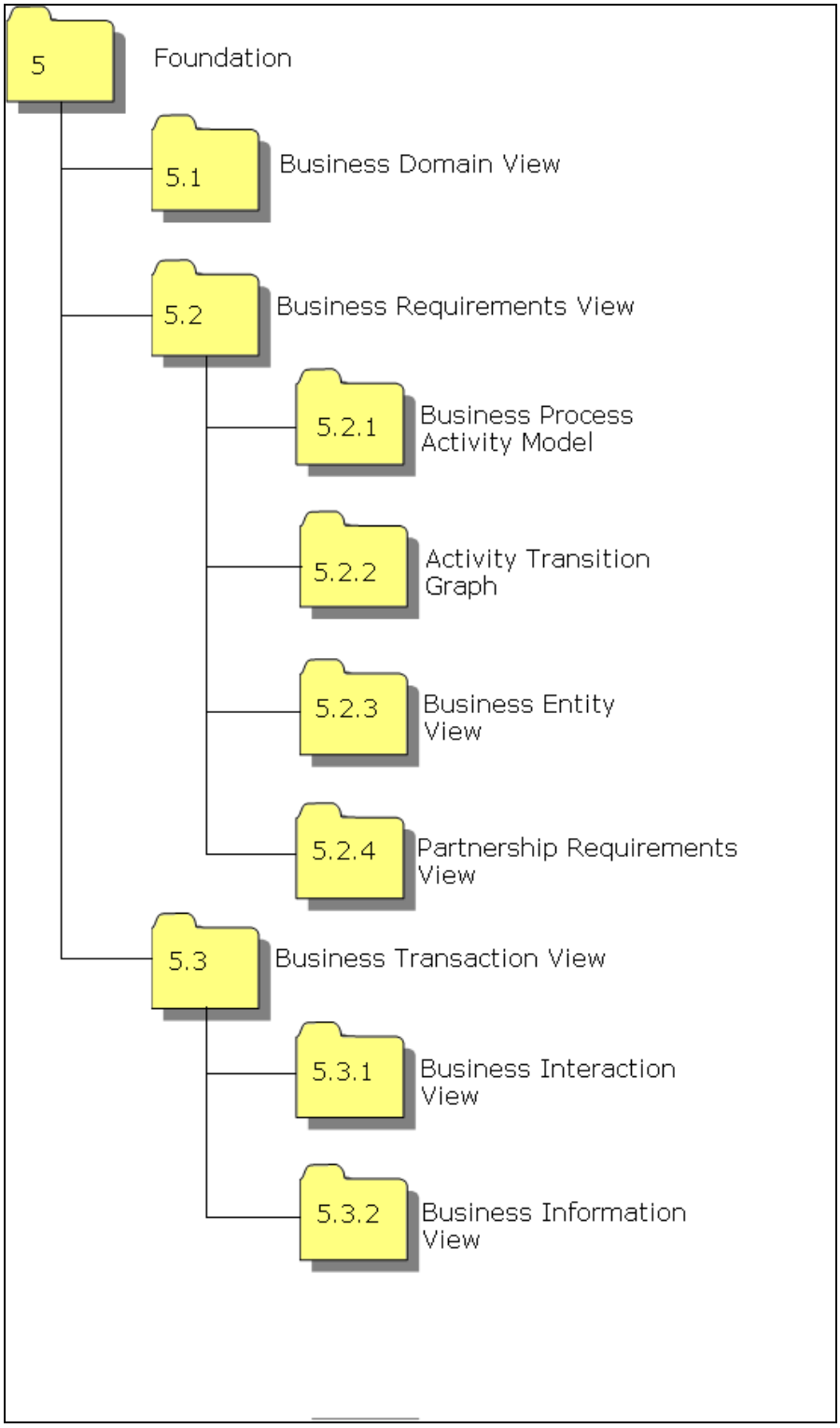

Figure 3 - Overview of the GSRM Extension Module

4. Dependency on other UMM modules (normative)

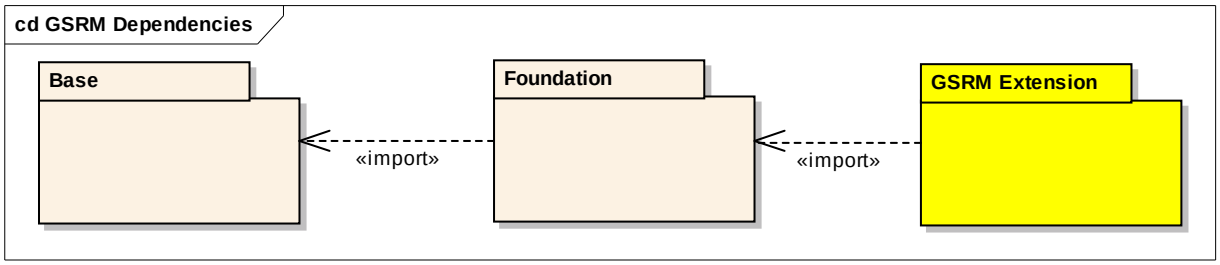

Figure 4 - GSRM Foundation Dependencies

The UMM foundation module 1.0 is built on top of the UMM base module 1.0. All UMM 1.0 stereotypes are shown in beige when these are used in GSRM MM without change. This means that all stereotypes and tag definitions defined in the UMM base module 1.0 are imported into the UMM foundation module 1.0. Figure 5 shows the stereotypes defined in the UMM base module are also used in the foundation module. Note that the stereotypes of the base module are depicted in grey background in all figures of this specification. The formal definition of the stereotypes *RegistryObject* and *BusinessLibrary* is given in the UMM base module 1.0 specification. In the foundation module, packages - that are containers of stereotypes realizing main UMM artifacts - are defined as specializations of the base stereotype *BusinessLibrary*. This means that such packages and their contents are candidates for registration in a registry. The UMM foundation module 1.0 does not define any stereotype that directly inherits from *RegistryObject*. As a consequence, only packages are candidates for registration. The GSRM Extension Module is built on top of the UMM Foundation Module, and thereby inherits the same registry usage as the UMM Base Module 1.0.

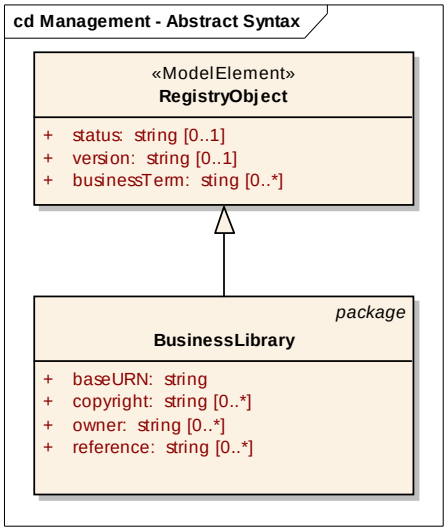

Figure 5 - UMM Base Abstract Syntax

5. GSRM Foundation Module Management

5.0. Foundation Model Management

5.0.1. Conceptual Overview (informative)

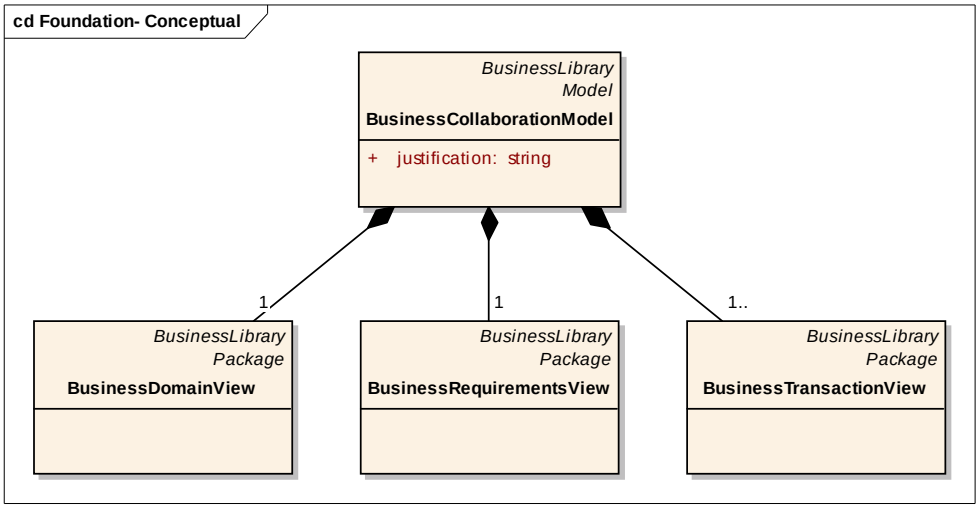

Figure 6 - GSRM Foundation Module Management - Conceptual Overview

A modeling project that follows the UMM approach leads to a BusinessCollaborationModel. A business collaboration model that is UMM compliant is stereotyped as *BusinessCollaborationModel*, which is part of a Government Service Model. As described above the UMM is built by three views. The *BusinessDomainView*, *BusinessRequirementsView*, and the *BusinessTransactionView* are mandatory parts of a business collaboration model (note, while the *BusinessDomainView* is optional in UMM, it is absolutely mandatory for GSRM). Thus a *BusinessCollaborationModel* is composed of exactly one *BusinessDomainView*, one *BusinessRequirementsView*, and one *BusinessTransactionView*.

5.0.2. Stereotypes and Tag Definitions (normative)

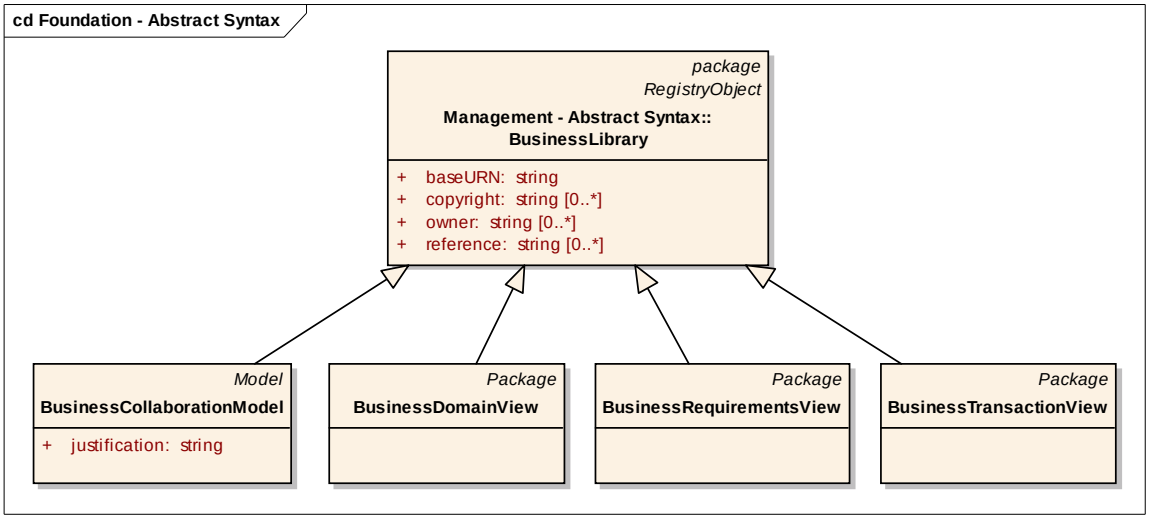

Figure 7 - GSRM Foundation Module Management - Abstract Syntax

Stereotype	RegistryObject	
Base Class	ModelElement	
Parent	**N/A**	
Description	A registry object (or a specialization of it) is a model element, which is a candidate for registration in a registry	
Tag Definition	version	
	Type	String
	Multiplicity	0..1
	Description	Holds the current version of a registered object. Each registered object SHOULD have version information and it SHOULD be set and maintained by the responsible registry
	status	
	Type	String
	Multiplicity	0..1
	Description	An indicator for the current lifecycle status of a registered object. The status MUST be set by the registry
	businessTerm	
	Type	string
	Multiplicity	0..*
	Description	A business term is a synonym, by which a business entity is commonly known

Stereotype	BusinessLibrary	
Base Class	Package	
Parent	**RegistryObject**	
Description	A Business Library is a container for objects, which together build a semantic unit. All objects of this unit SHOULD be registered and retrieved together	
Tag Definition	baseURN	
	Type	String
	Multiplicity	1
	Description	The namespace of a registered business library. Concatenated with the name of the instance (local name), it MUST be a valid uniform Resource name (URN)
	owner	
	Type	String
	Multiplicity	0..*
	Description	The owner of business library, which might be an organization, an institution, or an individual
	copyright	
	Type	string
	Multiplicity	0..*
	Description	Holds information about the copyright of a library
	reference	
	Type	String
	Multiplicity	0..*
	Description	Identifies references to additional resources, where continuative information about the object could be found

Stereotype	BusinessCollaborationModel
Base Class	Model
Parent	**BusinessLibrary**
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 modules, and it MAY be compliant to one or more specialization and/or extension modules
Tag Definition	justification Type: String Multiplicity: 1 Description: Explains the reason from a why the modeled policies are worthy of government attention and specification as a *BusinessCollaborationModel* Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessDomainView
Base Class	Package
Parent	**BusinessLibrary**
Description	A business domain view is a framework for identification and understanding of business processes as well as categorizing them according to a classification scheme. The *BusinessDomainView* is a container capturing the categorization scheme and categorized business processes
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessRequirementsView
Base Class	Package
Parent	**BusinessLibrary**
Description	The business requirements view is a container for all elements needed to identify and describe the requirements on a collaboration between business partners
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessTransactionView
Base Class	Package
Parent	**BusinessLibrary**
Description	The business transaction view is a container for all elements needed to describe the choreography of a business collaboration at various levels and the information exchanged in each step of the choreography
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

5.0.3. Constraints (normative)

A *BusinessCollaborationModel* MUST contain exactly one *BusinessDomainView* package

```
package Model_Management
context Model

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

A *BusinessCollaborationModel* MUST contain exactly one *BusinessRequirementsView* package

```
package Model_Management
context Model

inv oneBusinessRequirementsView:
  self.isBusinessCollaborationModel() implies
  self.ownedElement->select(isBusinessRequirementsView)->size=1
```

A *BusinessCollaborationModel* MUST contain exactly one *BusinessTransactionView* Package

```
package Model_Management
context Model

inv oneBusinessTransactionView:
  self.isBusinessCollaborationModel() implies
  self.ownedElement->select(isBusinessTransactionView)->size=1
```

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

```
package Model_Management
context Model

inv rootLevelPackage:
  (
    self.isBusinessDomainView() or
    self.isBusinessRequirementsView() or
    self.namespace.isBusinessTransactionView()
  ) implies
  self.namespace.isBusinessCollaborationModel()
```

5.0.4. OCL Methods used in the GSRM Foundation Module Management (normative)

OCL-Methods

```
Package  Foundation::Core
Context  ModelElement

--predefined method which evaluates, if the given ModelElement
--has a stereotype equal to the passed name
def:
let hasStereotype (st : String) : Boolean =
  self.stereotype->select(cst | cst.name = st)->notEmpty()

--predefined method which evaluates, if the given elements
--has the stereotype 'BusinessCollaborationModel'
def:
let isBusinessCollaborationModel() : Boolean =
  self.oclIsKindOf(Model) and
  self.hasStereotype('BusinessCollaborationModel')

--predefined method which evaluates, if the given elements
--has the stereotype 'BusinessDomainView'
def:
let isBusinessDomainView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessDomainView')

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

--predefined method which evaluates, if the given elements
--has the stereotype 'BusinessTransactionView'
def:
let isBusinessTransactionView() : Boolean =
  self.oclIsKindOf(Package) and
  self.hasStereotype('BusinessTransactionView')
```

5.1. Business Domain View

5.1.1. Conceptual Overview (informative)

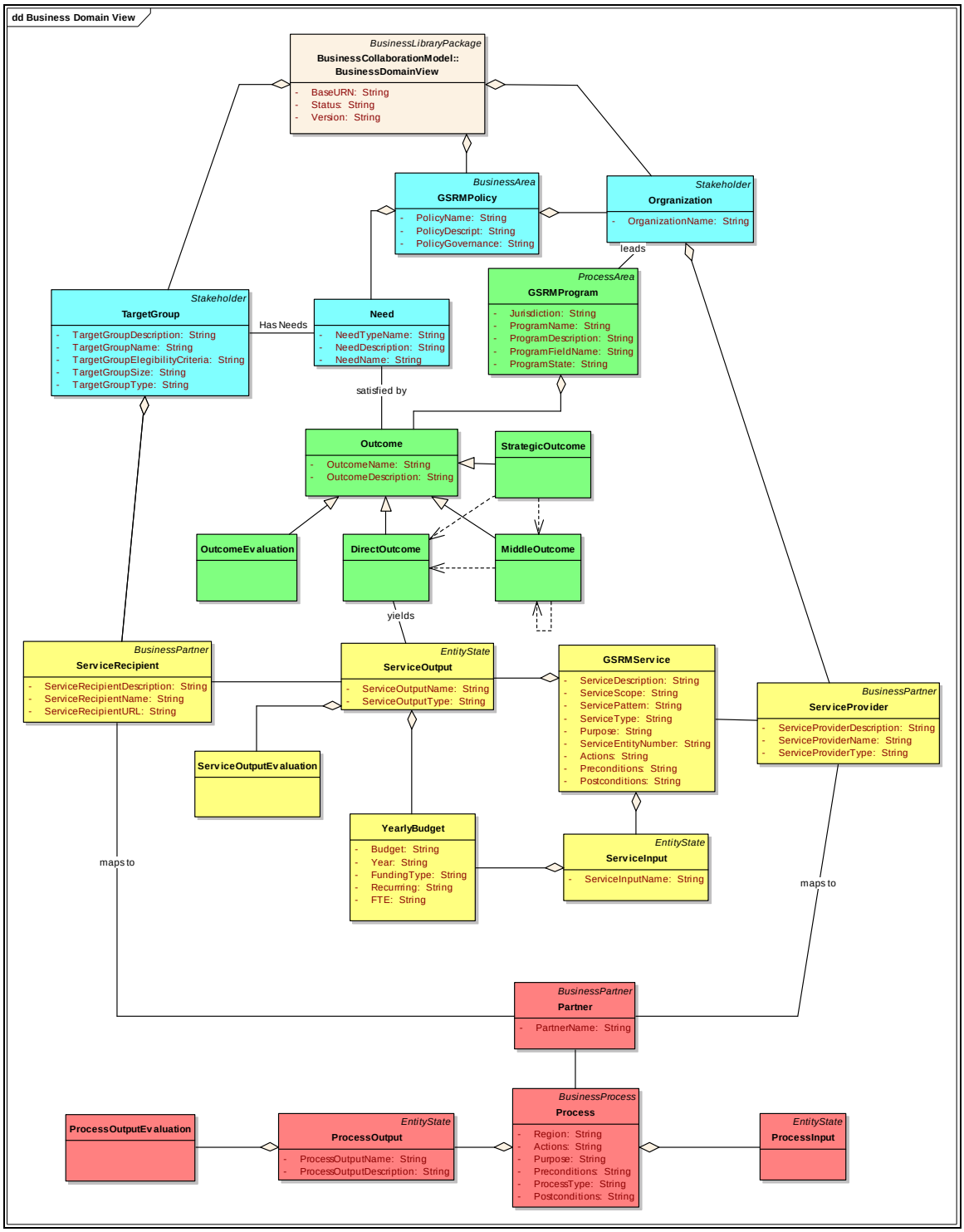

Figure 8 - BusinessDomainView - Conceptual Overview

The *BusinessDomainView* is used to discover business objects that are of relevance in a *BusinessCollaborationModel*. The GSRM extension of the *BusinessDomainView* is based on the UMM *BusinessDomainView*, with significant extensions added to better model concepts and relationships that are unique to government.

The focus in GSRM is on explicitly relating the four concept groupings designated by the four colours. These four logical groupings are associated with each other through *Need*, *Outcomes*, *ServiceOutput* and *ProcessOutput*. The associations are justified by the nature of the relationship between these key concepts.

- The justification for a *Policy* is that it identifies a *Need* of a *TargetGroup*.
- The justification of a *Program* is that it creates *Outcome* that satisfies a *Need*, identified by the parent *Policy*.
- The justification of a *Service* is that it has S*erviceOutput* that contribute to *Outcome* that satisfy *Need*, identified by the parent *Program*.
- The justification of a *Process* is that it has *ProcessOutput* that contribute to a *ServiceOutput* in the parent *Service*.

A *BusinessDomainView* contains complete information about a policy, including the participating *Organization*, the one or more collection of *TargetGroup*, and the *Need* being met by the policy and the *Policy* object itself. The *BusinessDomainView* will contain at least one *Organization*, and that *Organization* could be used in many instances of a *BusinessDomainView* or may be specified without any modeled *BusinessDomainView*; there is a (1..*) to (0..*) relationship between *BusinessDomainView* and *Organization*. Each *BusinessDomainView* will have at least one but possibly more *TargetGroup*, a *TargetGroup* will only exist if it is part of a model but it may be used in many models; there is a (1..*) to (1..*) relationship between *BusinessDomainView* and *TargetGroup*. Each *BusinessDomainView* will also contain exactly one *Policy*; there is a 1 to 1 relationship between *BusinessDomainView* and *Policy*.

An identified *Need* will always be associated to at least one and often many *TargetGroup* that have that *Need*. A *TargetGroup* will have at least one identified *Need* but it may have several instances of *Need*. There is a (1..*) to (1..*) association between *Need* and *TargetGroup.*

An *Organization* can contribute to a policy in three different ways:

- First there will be at least one *Organization* leading and ultimately responsible for each *Program* and an *Organization* may lead more than one *Program*; there is a (1..*) to (1..*) association between *Organization* and *Program*.
- An *Organization* can also be responsible for appointing one or more instances of *ServiceProvider*. Each *ServiceProvider* will be associated with a single *Organization*, and an *Organization* could provide to any number of *ServiceProvider*; there is a 1 to (0..*) relationship between *Organization* and *ServiceProvider*.

- Finally, an *Organization* can contribute the resources (financial or other) that will be used as a *ServiceInput*; these resources are grouped into one or more instances of a *ResourcePool*. An *Organization* can contribute to zero or more instances of a *ResourcePool*, and each *ResourcePool* will only come from one *Organization*; there is a 1 to (0..*) relationship between *Organization* and *ResourcePool*.

A *Policy* contains at least one and possibly several instances of *Program*. GSRM specifies that such a containment relationship only makes sense if the *Program* has an *Outcome* that meets the *Need* associated with the *Policy*. In order to correctly model this, there is an association between a *Need* and an *Outcome*. *Need* may be satisfied by one or many *Outcome*. Inside a *Program, Outcome* is nested into subtypes, and there may be no specific mapping of a modeled *Outcome* to a *Need*. In general there must be at least one *Outcome* mapping to *Need*; no single *Outcome* will necessarily satisfy any *Need*.
There is a (0..*) to (1..*) association between *Need* and *Outcome.*

Outcome is nested into three subtypes: *DirectOutcome, MiddleOutcome,* and *StrategicOutcome*.

- A *DirectOutcome* is a direct transformation of *ServiceOutput*. Each *ServiceOutput* will transform to one and only one *DirectOutcome*. There is a 1 to 1 association between *DirectOutcome* and *ServiceOutput*. Each *DirectOutcome* must be contained by at least one, or possibly more, *MiddleOutcome*.
- *MiddleOutcome* is a transformation of *DirectOutcome* or other *MiddleOutcome*. A *MiddleOutcome* may therefore contain any mix of other *MiddleOutcome* or *DirectOutcome*. They must contain at least one *MiddleOutcome* or one *DirectOutcome*. There is a (1..*) to (0..*) containment relationship between *DirectOutcome* and *MiddleOutcome*. Because a *MiddleOutcome* may be contained by another *MiddleOutcome*, there is also a (0..*) to (0..*) containment relationship between *MiddleOutcome* and *MiddleOutcome.*
- A *StrategicOutcome* is a transformation of *MiddleOutcome*. A particular *MiddleOutcome* may be contained by either a *StrategicOutcome* or another *MiddleOutcome*. There is a (0..*) to (1..*) containment relationship between *MiddleOutcome* and *StrategicOutcome.*

Each Service will require at least one Government *ServiceProvider* to generate the *ServiceOutput*. A *ServiceProvider* is a partner responsibility that a person or agency might assume as part of a *Service*, as the partner is unique to the *Service*. Each *ServiceProvider* Partner maps to a unique *Service*. There is a 1 to (1..*) association between *Service* and *ServiceProvider*.

A *Service* instance will achieve exactly one *ServiceOutput*. A *ServiceOutput* will be generated by a unique GSRM-Service. There is a 1 to 1 containment relationship between *ServiceOutput* and *Service.*

Each *ServiceOutput* is received by one or more instances of a *ServiceRecipient.* A *ServiceRecipient* partner may be eligible to receive one or more instances of a *ServiceOutput*. There is a (1..*) to (1..*) containment relationship between *Service* and *ServiceRecipient*.

Services also require an intstance of a *ServiceInput* (monetary and other) that will be used to create the *ServiceOutput*. Each *ServiceInput* will be associated with one *Service*, and a *Service* can have zero or more instances of a *ServiceInput*. One or more instances of a *ServiceInput* are aggregated into one or more instances of a *ResourcePool* that are then associated with the *Organization*s that will provide the resource to be used as a *ServiceInput*; each *ServiceInput* is associated with a single *ResourcePool* and a *ResourcePool* can aggregate one or more instances of a *ServiceInput*. There is a 1 to (1..*) relationship between *ResourcePool* and *ServiceInput*.

A *ServiceOutput* is a transformation of one or more instances of a *ProcessOutput*. A *ProcessOutput* may be modeled without modeling a direct contribution to a *ServiceOutput* (in the case where there may be unintentional or secondary outputs) and may be mapped to more than one instance of a *ServiceOutput* (typically in the case where the process is being reused). There is a 0..* to 1..* association between *ProcessOutput* and *ServiceOutput*.

A *Process* will generate at least one 1 *ProcessOutput*. All instances of a *ProcessOutput* will be generated by one *Process*. There is a 1..* to 1 containment relationship between *ProcessOutput* and *Process*.

An instance of *Partner* can be a participant in a *Process*. There are always at least 2 instances of *Partner* associated to a *Process*. A *Partner* should always be associated with at least one *Process*; There is a (0..*) to (2..*) association from *Process* to *Partner*.

In the case where a parent *Process* includes a child *Process*, a *Partner* in the parent may map to a *Partner* in the child; a *Partner* in the parent may map to several instances of *Partner* in one or more child instances of *Process*. There is a (0..*) to (0..*) *mapsTo* association from *Partner* to *Partner*.

Every *ProcessOutput* will also be associated with a *Partner* that receives the *ProcessOutput*. Each *ProcessOutput* is received by exactly one *Partner*, but a *Partner* may receive one or many instances of *ProcessOutput*. There is a 1 to (0..*) association between *Partner* and *ProcessOutput*.

Every *Outcome* may have a set of *OutcomeEvaluation* which provides the successcriteria for that *Outcome*. Thus there is a (0..1) relationship between *OutcomeEvaluation* and *Outcome*.

Every *ServiceOuput* may have a set of *ServiceOutputEvaluation* which provides the success criteria for that *ServiceOutput*. Thus there is a (0..1) relationship between *ServiceOuputEvaluation* and *ServiceOuput*.

Every *ProcessOutput* may have a set of *ProcessOutputEvaluation* which provides the success criteria for that *ProcessOutput*. Thus there is a (0..1) relationship between *ProcessOutputEvaluation* and *ProcessOutput*.

5.1.2. Stereotype and Tag Definitions (normative)

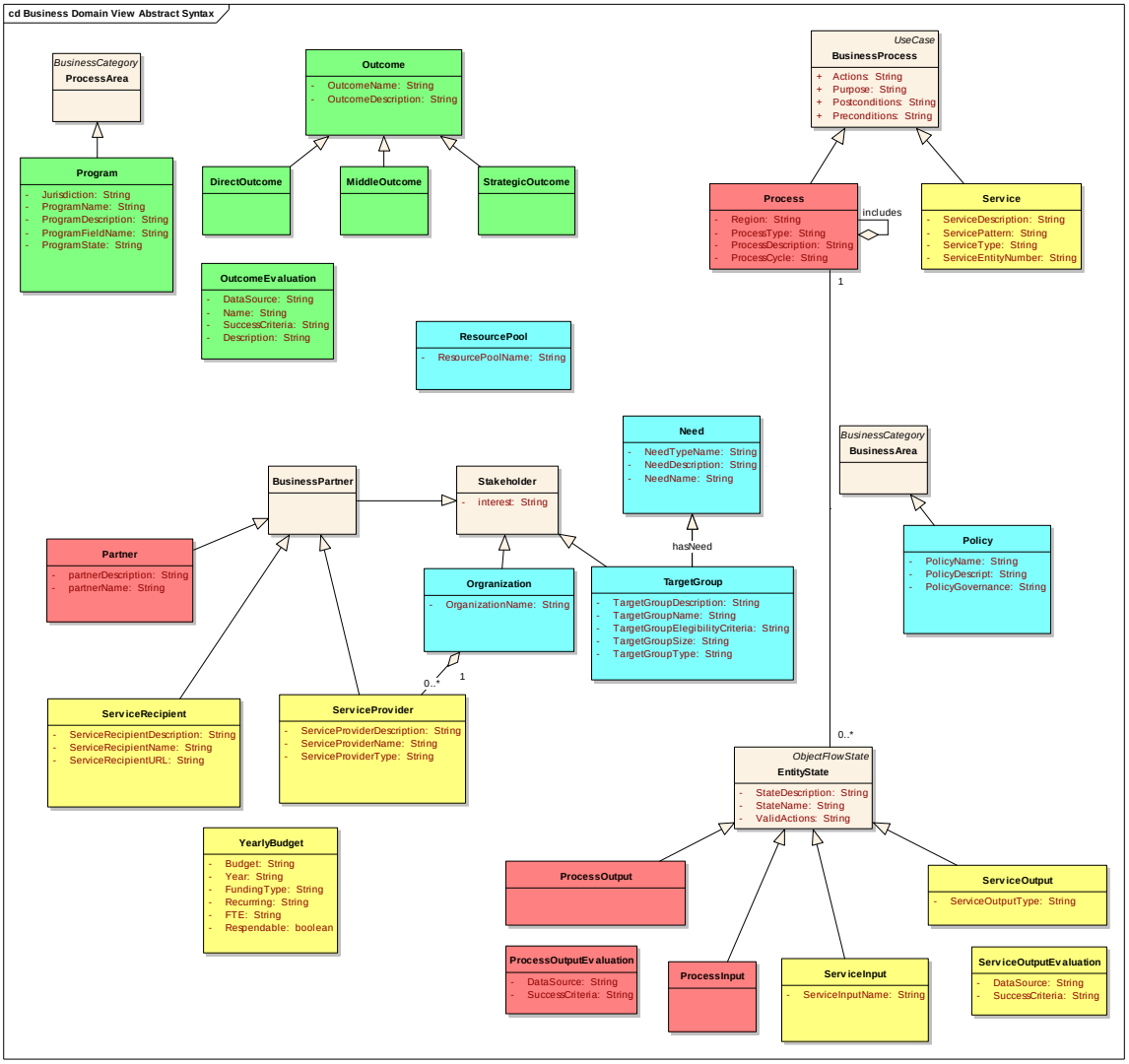

Figure 9 – BusinessDomainView - Abstract Syntax

Stereotype	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 (in case of a recursive structure only a business area on the lowest level) is a category of decomposable business process areas. This means a business area collates either other business areas or process areas. The UMM does not mandate a specific classification schema. A classification schema that might be used is the Porter Value Chain. Based on the Porter Value Chain the UN/CEFACT Common Business Process Catalog recommends a list of eight flat (i.e. non-recursive) categories: Procurement/Sales, Design, Manufacture, Logistics, Recruitment/Training, Financial Services, Regulation, and Health Care. This list of business areas is considered as non exhaustive. The recommended GoC classification scheme for Business Area is the "Program Fields" provided by GSRM top model.
Tag Definition	Inherited tagged values: • objective • scope • businessOpportunity • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	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 (in case of a recursive structure only a process area on the lowest level) is a category of common business processes. This means a process area collates either other process areas or business processes. The UMM does not mandate a specific classification schema. The UN/CEFACT Common Business Process Catalog recommends a list of five flat (i.e. non-recursive) categories that correspond to the five successive phases of business collaborations as defined by the ISO Open-edi model: Planning, Identification, Negotiation, Actualization, Post-Actualization. The recommended GoC classification scheme for Process Area is the "Service Output Type" provided by GSRM top model.
Tag Definition	Inherited tagged values:

	• objective • scope • businessOpportunity • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	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 she or he is defined within, or a business process linked to her or him Inherited tagged values: None

Stereotype	ServiceProvider
Base Class	Actor
Parent	**Partner**
Description	Any Organization responsible for the direct delivery of Service Outputs to the Clients.
Tag Definition	serviceProviderName Type: String Multiplicity: 1 Description: The name given to the Service Provider **serviceProviderDescription** Type: String Multiplicity: 1 Description: A description of the Service Provider **serviceProviderType** Type: Enum Multiplicity: 1 Description: Describes the Provider as either an outside Agent or Government Options: • Agent • Government Representative Inherited tagged values: • Interest

Stereotype	TargetGroup
Base Class	Actor
Parent	**Stakeholder**
Description	A Target Group is a subset of the general population that exhibits a designated need. A Target Group is acknowledged and defined in Policy/Legislation as the intended primary beneficiary of the efforts of the Program. That is, it is a Target Group whose level of need the Program is intended to change. A Target Group is the ultimate beneficiary of a Service Output. In some cases, this beneficiary is different than the recipient.
Tag Definition	**targetGroupName** Type: Type Multiplicity: 1 Description: The Name given to the Target Group **targetGroupDescription** Type: String Multiplicity: 1 Description: A description of the Target Group **targetGroupSize** Type: long Multiplicity: 1 Description: An estimation of the size of the Target Group **targetGroupElegibilityCriteria** Type: String Multiplicity: 1 Description: The rules that determine eligilibity for membership within a Target Group. (OCL - Government Policy and Programs' determine the eligibility criteria). **targetGroupType** Type: enum Multiplicity: 1 Description: Describes whether the members of the Target Group are Individual, Collective, or both Options: • Individual • Collective • Both Inherited tagged values: • interest

Stereotype	BusinessPartner
Base Class	Actor
Parent	**Stakeholder**
Description	A business partner is an organization type, an organizational unit type or a person type that participates in a business process. Business partners typically provide input to and/or receive output from a business process. Due to the fact that a business partner participates in a business process, a business partner has by default, a vested interest in the business process. It follows that a *BusinessPartner* is a special kind of *Stakeholder*.
Tag Definition	partnerName Type: String Multiplicity: 1 Description: A short description of the partner. Inherited tagged values: • interest

Stereotype	ServiceRecipient
Base Class	Actor
Parent	**Partner**
Description	A Service Recipient is the party that directly receives or experiences, either willingly or begrudgingly, the output of a service.
Tag Definition	serviceRecipientName Type: String Multiplicity: 1 Description: The name given to the Service Recipient **serviceRecipientDescription** Type: String Multiplicity: 1 Description: A description of the Service Recipient **serviceRecipientURL** Type: String Multiplicity: 1 Description: There are several GoC reports that allow for the addition of a URL to identify service recipients, when the recipient is an organization. It can be added here. It is expected that other similar contact information might be added to ServiceRecipient at a later stage. Inherited tagged values: • Interest

Stereotype	Policy
Base Class	Package
Parent	**None**
Description	A Policy is a set of regulations that inform and drive the design of an organization as well as governing how it functions
Tag Definition	policyName Type: String Multiplicity: 1 Description: The name of the Policy **policyDescription** Type: String Multiplicity: 1 Description: A description of the Policy **policyGovernance** Type: String Multiplicity: 1 Description: A description of the how the Policy is governed. Inherited tagged values: • status • baseURN • version • businessTerm • owner • copyright • reference

Stereotype	Program
Base Class	Package
Parent	**ProcessArea**
Description	A program is an accountable mandate to address recognized needs of eligible target groups. Each program aims to achieve specified outcomes for members of those target groups. The program achieves this by designing service outputs that will contribute to the outcomes desired for the program.
Tag Definition	jurisdiction Type: String Multiplicity: 1 Description: Jurisdiction of the program. **programName** Type: String Multiplicity: 1 Description: The name of the Program **programDescription** Type: String Multiplicity: 1 Description: A description of the program **programFieldName** Type: Enum

	Multiplicity	1
	Description	The Program Field Name used to classify a Program
	Options	• (Socio-)Economic Development • Science and Knowledge Development • Natural Resources • Environmental Protection • Public Health • Legal, Collective, Democratic & Human Rights • Social Development • Cultural Development • Public Education • Public Safety • Justice • National Security & Defense
	programState	
	Type	String
	Multiplicity	1
	Description	A value representing the current state of the program (ie. active, planned, suspended, etc.)
	Inherited tagged values: • objective (equivalent to GSRM Goal) • scope • business opportunity (equivalent to GSRM Strategy) • status • baseURN • version • businessTerm • owner • copyright • reference	

Stereotype	Service
Base Class	Package
Parent	**BusinessProcess**
Description	A Service is a means, administered by a program, of producing a final valued output (i.e. service output) to address one or more target Group needs. Something is a service when all four of the following criteria are met: • There is a target Group (i.e. individual(s) or organization(s) with needs) • An output that is delivered to a service recipient (client) • There are processes that produce intermediate outputs (associated with the service) Proper authority (Program/Policy) exists to serve the target Group)
Tag Definition	serviceType
	Type: String Multiplicity: 1

Description	A code for the type of service (Program Activty, Sub Program Activity, Sub Sub Program Activity)
serviceDescription	
Type	String
Multiplicity	1
Description	A description of the Service
servicePattern	
Type	string
Multiplicity	1
Description	A value representing the Pattern of the Service as per the Business Transformation Enablement Program Service Reference Pattern Level 2
Options	• Provide Funds • Provide Resources • Provide Transport • Provide Advisory Encounter • Provide Matches, Referrals & Linkages • Provide New Knowledge • Provide Advocacy and Promotional Encounters • Provide Recreational & Cultural Encounters • Provide Educational & Training Encounters • Provide Care & Rehabilitation Encounters • Provide Periods of Agreement • Provide Periods of Permission • Provide Periods of Protection • Provide Findings • Provide Interventions • Provide Rulings and Judgements • Provide Penalties and Periods of Sanction • Provide Rules • Provide Implemented Changes
serviceEntityNumber	
Type	String
Multiplicity	1
Description	The Service Entity Number (GoC) or a similar unique identifier in non-GoC environments.

Inherited tagged values:

- objective (equivalent to GSRM Goal)
- scope
- business opportunity (equivalent to GSRM Strategy)
- purpose
- actions
- preconditions
- postconditions

Stereotype	Process
Base Class	UseCase
Parent	**BusinessProcess**
Description	UMM defines a process as “a set of related activities that together create value”. GSRM defines a process as “a set of related activities that together create a valued output”. A business process might be performed by a single partner or by multiple partners crossing organizational boundaries. In cases where organizations collaborate in a process, the process should create a valued output for all its participants.
Tag Definition	**processName** Type: String Multiplicity: 1 Description: The name of the process. **processCycle** Type: Enum Multiplicity: 1 Description: The Process Cycle is one of four (4) Phases in the lifetime of a process. These are the rough equivalent to the REA Phases within UMM. Options: • Planning Processes • Provisioning Processes • Delivery Processes • Decommissioning/Deregistering Processes **processType** Type: Enum Multiplicity: 1 Description: Process type values come from Service Reference Process Patterns and are from the Level 1 Service Pattern. The process type is a valid value within one of the process cycles. Options: • Planning Processes (cycle description only) o Recognize service Planning cycle o Recognize service contingency event o Forecast service demand o Forecast service risks o Set performance targets for service, processes, resources o Measure performance of service, processes, resources o Estimate service resource requirements o Allocate resources to service processes • Provisioning Processes (cycle description only) o Monitor service resource consumption o Monitor service resource availability o Configure service processes to respond to demand or supply level limits o Configure service processes to respond to contingency event o Source service resources o Register and equip service suppliers o Acquire and register service resources o Pay for service resources o Maintain service resources o Deploy service resources geographically o Set service schedule o Configure service resources o Protect service resources

- Promote services
- Monitor and mitigate service risks
- Process service complaints
- Register and equip service target group members

- Delivery Processes (cycle description only)
  - Register request for service delivery
  - Qualify request for service delivery
  - Set service delivery schedule and notify
  - Open service delivery case
  - Allocate resources to service output
  - Deploy resources for service output
  - Produce service output
  - Deliver service output
  - Collect and account for a service output fee
  - Process service exceptions
  - Register service output
  - Maintain service output
  - Close service delivery case
- Decommissioning/Deregistering Processes (cycle description only)
  - Decommission/deregister service output
  - Decommission/deregister service resources
  - Deregister service suppliers
  - Deregister service target group members

processDescription	
Type	String
Multiplicity	1
Description	A description of the process

Region	
Type	String
Multiplicity	1
Description	The region benefiting from the process

Inherited tagged values:

- purpose
- actions
- preconditions
- postconditions

Stereotype	Need
Base Class	Package
Parent	**None**
Description	A need is defined as a lack of something requisite, desirable, or deemed a basic necessity to an individual or a collective. Target groups have a Need. A program, and the services that are part of the Program, are designed to change the level of a Target Group Need. A Service Recipient receives an output from a service. The level of a Target group Need is changed by the value contribution demonstrated by each Service Recipient receiving that Service Output.
Tag Definition	needsTypeName

needsTypeName	
Type	enum
Multiplicity	1..*
Description	The classification of Target Group Needs
Options	Need categories for individuals

	• Basic Physiological • Safety and Security • Belonging (Social) • Esteem • Self-actualization Need Categories for Collectives • Resources • Risk/Threat Mitigation • Mission Fulfillment **needsName** Type: String Multiplicity: 1 Description: The name of the Need **needsDescription** Type: String Multiplicity: 1 Description: A definition of the Need Inherited tagged values: **None**

Stereotype	Partner
Base Class	Actor
Parent	**BusinessPartner**
Description	A *BusinessPartner* (e.g. a "buyer") is a concept which is more generic than a *Partner* (e.g. a "broker") and allows the reuse of collaborations by mapping a *BusinessPartner* to a *Partner* within a given scenario. Since *BusinessCollaborationUseCase* and *BusinessTransactionUseCase* are defined as occurring between instances of BusinessPartner, they might be reused by different Instances of *Partner* (a "broker" or a "custodian") in different scenarios of the same domain or even in different domains.
Tag Definition	partnerName Type: String Multiplicity: 1 Description: The name of the Partner **partnerDescription** Type: String Multiplicity: 1 Description: A description of the Partner Inherited tagged values: • interest

Stereotype	EntityState
Base Class	State
Parent	**N/A**
Description	A business entity state represents a certain state that a business entity can exist in during its lifecycle (an "order" can exist in the states of "issued", "rejected", "confirmed", etc.)
Tag Definition	StateName Type: String Multiplicity: 1 Description: The name of the State **StateDescription** Type: String

	Multiplicity: 1 Description: A Description of the State **ValidActions** Type: String Multiplicity: 1 Description: Valid Actions in this state

Stereotype	ProcessOutput
Base Class	Class
Parent	**EntityState**
Description	A Process Output is the end product of an individual process
Tag Definition	Inherited tagged values: • StateName • StateDescription • ValidActions

Stereotype	ServiceOutput
Base Class	Class
Parent	**BusinessEntity**
Description	A Service Output is a Service deliverable, the result of a service delivered to a Service Recipient
Tag Definition	**serviceOutputType** Type: Enum Multiplicity: 1 Description: A Service Output Type is a class of service outputs that share similar characteristics, sets of process and performance metrics. Options: • Funds • Resources • New Knowledge • Care and Rehabilitation Encounters • Educational and Training Encounters • Recreational and Cultural Encounters • Movements • Advisory Encounters • Matches, Referral Linkages • Advisory and Promotional Encounters • Periods of Agreement • Periods of Permission • Findings • Rulings and Judgments • Penalties and Periods of Sanction • Interventions • Rules • Implemented Changes Inherited tagged values: • StateName • StateDescription • ValidActions

Stereotype	Outcome

Base Class	Class
Parent	**None**
Description	An outcome is a desirable trend in the level of a target group need. In other words, the definition of a Program will identify the Outcome, one or more, that it is to achieve, by identifying the changes the program will make to the level of Target Group Need that it is mandated to address.
Tag Definition	**outcomeName** Type: String Multiplicity: 1 Description: The name of the Outcome **outcomeDescription** Type: String Multiplicity: 1 Description: A description of the Outcome Inherited tagged values: **None**

Stereotype	DirectOutcome
Base Class	Class
Parent	**Outcome**
Description	A Direct Outcome is a outcome that is solely attributable to the impact of a service output (i.e. from a single service)
Tag Definition	Inherited tagged values: • outcomeName • outcomeDescription

Stereotype	MiddleOutcome
Base Class	Class
Parent	**Outcome**
Description	A Middle outcome is an outcome that appears on a connecting chain between a Direct Outcome and a Strategic Outcome.
Tag Definition	Inherited tagged values: • outcomeName • outcomeDescription

Stereotype	StrategicOutcome
Base Class	Class
Parent	**Outcome**
Description	A Strategic Outcome is the 'last word' expressed by the program in terms of desirable trends in level of Target group Need for the largest possible target Group, the most comprehensive or highest level of Need, etc. For the Federal Government, these are defined in the annual report on Canada's performance.
Tag Definition	**serviceEntityNumber** Type: String Multiplicity: 1 Description: The Service Entity Number (GoC) or a similar unique identifier in non-GoC environments **serviceType** Type: String Multiplicity: 1

	Description	A code for the type of service (Strategic Outcome)

Inherited tagged values:

- outcomeName
- outcomeDescription

Stereotype	Evaluation
Base Class	Class
Parent	**None**
Description	Evaluation is an abstract class that is used as the parent of evaluations that will be used to judge whether an output (*ServiceOutput*, *ProcessOutput*,, or *Outcome*) was successful.
Tag Definition	dataSource Type: String Multiplicity: 1 Description: Describes how to gather the data that will be used to determine success. **successCriteria** Type: String Multiplicity: 1 Description: A description of how success will be judged; where possible this should be a formal specification. **Name** Type: String Multiplicity: 1 Description: The name of the evaluation **Description** Type: String Multiplicity: 1 Description: A description the evaluation

Stereotype	ProcessOutputEvaluation
Base Class	Class
Parent	**Evaluation**
Description	An evaluation that can be applied to judge the success of a ProcessOutput.
Tag Definition	Inherited tagged values: • dataSource • successCriteria • name • description

Stereotype	ServiceOutputEvaluation
Base Class	Class
Parent	**Evaluation**
Description	An evaluation that can be applied to judge the success of a ServiceOutput.
Tag Definition	Inherited tagged values:

	• dataSource • successCriteria • name • description

Stereotype	OutcomeEvaluation
Base Class	Class
Parent	**Evaluation**
Description	An evaluation that can be applied to judge the success of an Outcome.
Tag Definition	Inherited tagged values: • dataSource • successCriteria • name • description

Stereotype	YearlyBudget	
Base Class	Class	
Parent	**None**	
Description	This describes the Financial, personelle, and other resources that would be part of a *ServiceInput* or *ServiceOutput*.	
Tag Definition	budget	
	Type	String
	Multiplicity	1
	Description	An amount of money associated with this budgetary object.
	fte	
	Type	String
	Multiplicity	1
	Description	A description of personel resources associated with this budgetary object, described in Full Time Employees.
	recurring	
	Type	Boolean
	Multiplicity	1
	Description	True if this budget is a recurring item.
	respendable	
	Type	Boolean
	Multiplicity	1
	Description	True if this budget is respendable.
	year	
	Type	String
	Multiplicity	1
	Description	The financial year of this budget.
	fundingType	
	Type	String
	Multiplicity	1
	Description	The type of funding (Grant, Contribution, or other).

Stereotype	ServiceInput
Base Class	Class

Parent	**BusinessEntity**
Description	Describes the needed input for a service; a *ServiceInput* will generally act as a container of *yearlyBudget* objects or be a related either a *ServiceOuptut* or *ProcessOutput* that will actually describe the tangible asset that is required for the service. *ServiceInput* is a description of a service's required inputs; other classes describe the actual asset.
Tag Definition	serviceInputName Type: String Multiplicity: 1 Description: The name of the service input.

Stereotype	ProcessInput
Base Class	Class
Parent	**BusinessEntity**
Description	Describes the needed input for a Process; a *ProcessInput* will generally act as a container of objects or be a *ProcessOutput* that will actually describe the tangible asset that is required for the Process. *ProcessInput* is a description of a processes required inputs; other classes describe the actual asset.
Tag Definition	processInputName Type: String Multiplicity: 1 Description: The name of the service input.

Stereotype	ResourcePool
Base Class	Class
Parent	**None**
Description	A Resource Pool is a collection of one or more instances of *ServiceInput*; is collected so that it can be associated as a complete package to an *Organization*.
Tag Definition	resourcePoolName Type: String Multiplicity: 1 Description: The name of the resource pool.

Stereotype	Organization
Base Class	Class
Parent	**Outcome**
Description	An Organization describes all government and non-government agencies that might contribute to the success of a policy. An Organization can lead, contribute to, or provide the services that enable a policy.
Tag Definition	organizationName Type: String Multiplicity: 1 Description: The name of the organization.

Stereotype	hasNeed
Base Class	Association
Parent	**None**
Description	Describes the association between *TargetGroup* and *Need*
Tag Definition	**No Tagged Values**

Stereotype	leads
Base Class	Association
Parent	**None**
Description	Describes one possible association between *Organization* and *Program*
Tag Definition	**No Tagged Values**

Stereotype	satisfiedBy
Base Class	Association
Parent	**None**
Description	Describes the association between *Need* and *Outcome*
Tag Definition	**No Tagged Values**

Stereotype	yields
Base Class	Association
Parent	**None**
Description	Yields describe several associations where a lower level object can result in a higher level object. In the *BusinessDomainView*, there are yields associations between: • *ProcessOutput* and *ProcessInput* • *ProcessInput* and *ServiceInput* • *ProcessOutput* and *ServiceOutput* • *ServiceOutput* and *DirectOutcome* • *DirectOutcome* and *StrategicOutcome* • *DirectOutcome* and *MiddleOutcome* • *MiddleOutcome* and *MiddleOutcome* • *MiddleOutcome* and *StrategicOutcome*
Tag Definition	**No Tagged Values**

Stereotype	receives
Base Class	Association
Parent	**None**
Description	Describes the association between *ServiceOutput* and *ServiceRecipient*
Tag Definition	**No Tagged Values**

Stereotype	mapsTo
Base Class	Dependency
Parent	**None**
Description	Describes the relationship between Partners where a partner in a high level process (or service) will map to a partner in an included process. In the *BusinessDomainView* there is a *mapsTo* association between: • *ServiceProvider* and *Partner* • *ServiceRecipient* and *Partner* • *Partner* and *Partner*
Tag Definition	**No Tagged Values**

Stereotype	participates
Base Class	Association
Parent	**None**
Description	Describes the association between *Partner* and *Process*

Tag Definition	**No Tagged Values**

Stereotype	isCreatedBy
Base Class	Association
Parent	**None**
Description	Describes the association between *ServiceOutput* and *DirectOutcome*
Tag Definition	**No Tagged Values**

Stereotype	includes
Base Class	Association
Parent	**None**
Description	Describes the association between *Process* and *Process*
Tag Definition	**No Tagged Values**

5.1.3. Constraints (normative)

A *MiddleOutcome* MUST contain a *MiddleOutcome* or a *DirectOutcome*. It may contain both, and it must not contain any *StrategicOutcome*.

```
package Model_Management
context Class

inv contentsMiddleOutcome:
  self.isMiddleOutcome implies
  self.isOutcome() and
  self.contents->notEmpty() and
  (
    self.contents->select(isMiddleOutcome()->size >= 1) or
    self.contents->select(isDirectOutcome()->size >= 1)
  ) and
  self.conents->select(isStrategicOutcome())->size = 0
```

A *DirectOutcome* MUST not contain any *DirectOutcome, MiddleOutcome* or *DirectOutcome.*

```
package Model_Management
context Class

inv contentsDirectOutcome:
  self.isDirectOutcome implies
  self.isOutcome() and
  self.contents->isEmpty()
```

A *Service MUST* contain exactly one *ServiceOutput* and exactly one *Service*

```
package Model_Management
context Class

inv contentsService:
  self.isService() implies
  self.isBusinessCollaborationRealization() and
  self.contents->notEmpty() and
  self.contents->select(isServiceOutput()).size() = 1 and
  self.contents->select(isService()).size() = 1
```

A *Process* MUST contain one or more *ProcessOutput* and at least one of *BusinessCollaborationUseCase* or *BusinessTransactionUseCase*

```
package Model_Management
context Class

inv contentsProcess:
  self.isProcess implies
  self.contents->select(isProcessOutput())->size >= 1 and
  (
     self.contents->select(isBusinessCollaborationUseCase())
      >= 1 or
     self.contents->select(isBusinessTransactionUseCase()) >= 1
  )
```

A *hasNeed* association MUST always connect a *Need* and a *TargetGgroup*

```
package Model_Management
context Association

inv isHasNeedsConnector:
  self.isHasNeed() implies
  self.client->one(isNeed()) and
  self.supplier->one(isTargetGroup())
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

An *isSatisfied* association MUST always connect a *Need* and an *Outcome*

```
package Model_Management
context Association

inv isSatisfiedConnector:
  self.isSatisfied() implies
  self.client->one(isNeed()) and
  self.supplier->one(isOutcome()) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

An *isCreatedBy* association MUST always connect a *DirectOutcome* and a *ServiceOutput*

```
package Model_Management
context Association

inv isCreatedByConnector:
  self.isCreatedBy() implies
  self.client->one(isDirectOutcome()) and
  self.supplier->one(isServiceOutput()) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A *receives* association MUST always connect either:
a *ServiceOutput* and a *ServiceRecipient* or
a *Processoutput* and a *Partner*

```
package Model_Management
context Association

inv receivesConnector:
  self.isReceives() implies
  (
    (
      self.client->one(isServiceOutput()) and
      self.supplier->one(isServiceRecipient())
    ) or (
      self.client->one(isProcessOutput()) and
      self.supplier->one(isPartner())and
    )
  ) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A *provides* association MUST always connect a *Service* and a *ServiceProvider*

```
package Model_Management
context Association

inv providesConnector:
  self.isProvides() implies
  self.client->one(isService()) and
  self.supplier->one(isServiceProvidier()) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A *contributesTo* association MUST always connect a *ServiceOutput* and a *ProcessOutput*

```
package Model_Management
context Association

inv contributesToConnector:
  self.isContributesTo() implies
  self.client->one(isServiceOutput()) and
  self.supplier->one(isProcessOutput()) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A p*articipates* association MUST always connect a *Partner* to one of *Process, BusinessCollaborationUseCase* or *BusinessTransactionUseCase*

```
package Model_Management
context Association

inv participatesConnector:
  self.isParticipates() implies
  self.client->one(isPartner()) and
  (
    self.supplier->one(isProcess()) or
    self.supplier->one(isBusinessCollaborationUseCase) or
    self.supplier->one(isBusinessTransactionUseCase)
  ) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A m*apsTo* dependency MUST always connect a *Partner* to a *Partner*

```
package Model_Management
context Dependency

inv mapsToConnector:
  self.isMapsTo() implies
  self.client->one(isPartner()) and
  self.supplier->one(isPartner()) and
  self.client->size() == 1 and
  self.supplier ->size() == 1
```

A yields association MUST always connect one of the following:
A *ProcessOutput and ProcessInput* or
A *ProcessInput and ServiceInput* or
A *ProcessOutput and ServiceOutput* or
A *ServiceOutput and DirectOutcome* or
A *DirectOutcome and StrategicOutcome* or
A *DirectOutcome and MiddleOutcome* or
A *MiddleOutcome and MiddleOutcome* or
A *MiddleOutcome and StrategicOutcome*

```
package Model_Management
context Association

inv yieldsConnector:
  self.isYields() implies

  (self.client->one(isProcessOutput()) and
  self.supplier->one(isProcessInput())) or
  (self.client->one(isProcessInput()) and
  self.supplier->one(isServiceInput())) or
  (self.client->one(isProcessOutput()) and
  self.supplier->one(isServiceOutput())) or
  (self.client->one(isServiceOutput()) and
  self.supplier->one(isDirectOutcome())) or
  (self.client->one(isDirectOutcome()) and
  self.supplier->one(isStrategicOutcome())) or
  (self.client->one(isDirectOutcome()) and
  self.supplier->one(isMiddleOutcome())) or
  (self.client->one(isMiddleOutcome ()) and
  self.supplier->one(isMiddleOutcome ())) or
  (self.client->one(isMiddleOutcome ()) and
  self.supplier->one(isStrategicOutcome ())) and
  (self.client->size() == 1 and
  self.supplier ->size() == 1)
```

5.1.4. OCL Methods used in Business Domain View (normative)

OCL-Methods

```
Package  Foundation::Core
Context  ModelElement

--predefined method which evaluates, if the given ModelElement
--has a stereotype equal to the passed name
def:
let hasStereotype (st : String) : Boolean =
  self.stereotype->select(cst | cst.name = st)->notEmpty()

--predefined method which evaluates, if the given elements
--has the stereotype 'Needs'
def:
let isNeed() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('Need')

--predefined method which evaluates, if the given elements
--has the stereotype 'Outcome'
def:
let isServiceRecipient() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('Outcome')

--predefined method which evaluates, if the given elements
--has the stereotype 'Policy
def:
let isServiceRecipient() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('Policy')

--predefined method which evaluates, if the given elements
--has the stereotype 'ProcessOutput'
def:
let isServiceRecipient() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('ProcessOutput')

--predefined method which evaluates, if the given elements
--has the stereotype 'Program'
def:
let isProgram() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('Program')

--predefined method which evaluates, if the given elements
--has the stereotype 'Partner'
def:
let isBusinessPartner() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype('Partner')
```

```
--predefined method which evaluates, if the given elements
--has the stereotype ‘Service’
def:
let isService () : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype(‘Service’)

--predefined method which evaluates, if the given elements
--has the stereotype ‘ServiceOutput’
def:
let isServiceOutupt() : Boolean =
  self.oclIsKindOf(Class) and
  self.hasStereotype(‘ServiceOutput’)

--predefined method which evaluates, if the given elements
--has the stereotype ‘ServiceProvicer’
def:
let isServiceProvider() : Boolean =
  self.oclIsKindOf(Actor) and
  self.hasStereotype(‘ServiceProvider’)

--predefined method which evaluates, if the given elements
--has the stereotype ‘ServiceRecipient’
def:
let isServiceRecipient() : Boolean =
  self.oclIsKindOf(Actor) and
  self.hasStereotype(‘ServiceRecipient’)

--predefined method which evaluates, if the given elements
--has the stereotype ‘TargetGroup’
def:
let isTargetGroup() : Boolean =
  self.oclIsKindOf(Actor) and
  self.hasStereotype(‘TargetGroup’)
```

5.1.5. Example – Small Business Startup (informative)

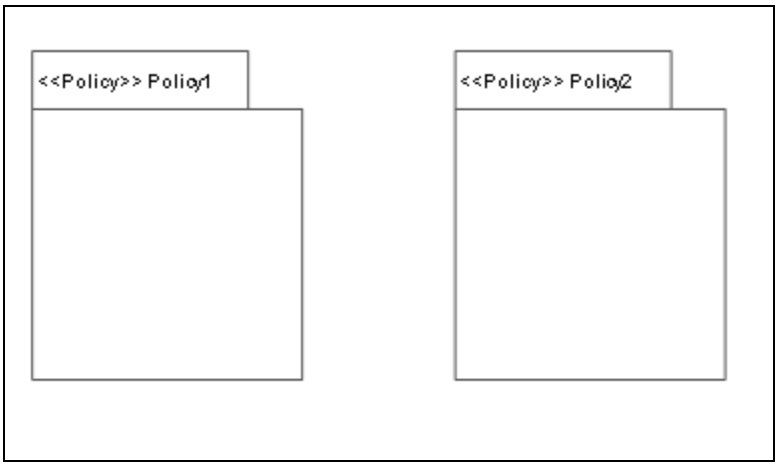

Figure 10 – BusinessDomainView

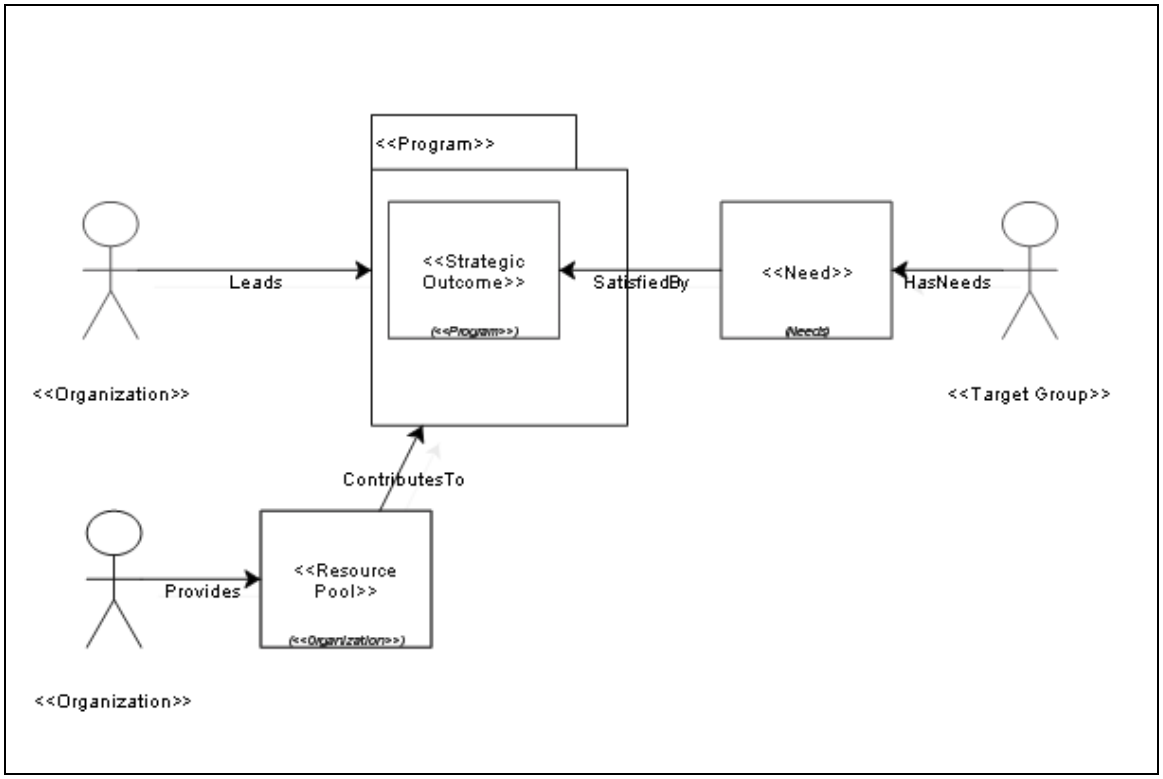

Figure 11 - BusinessDomainView (Policy)

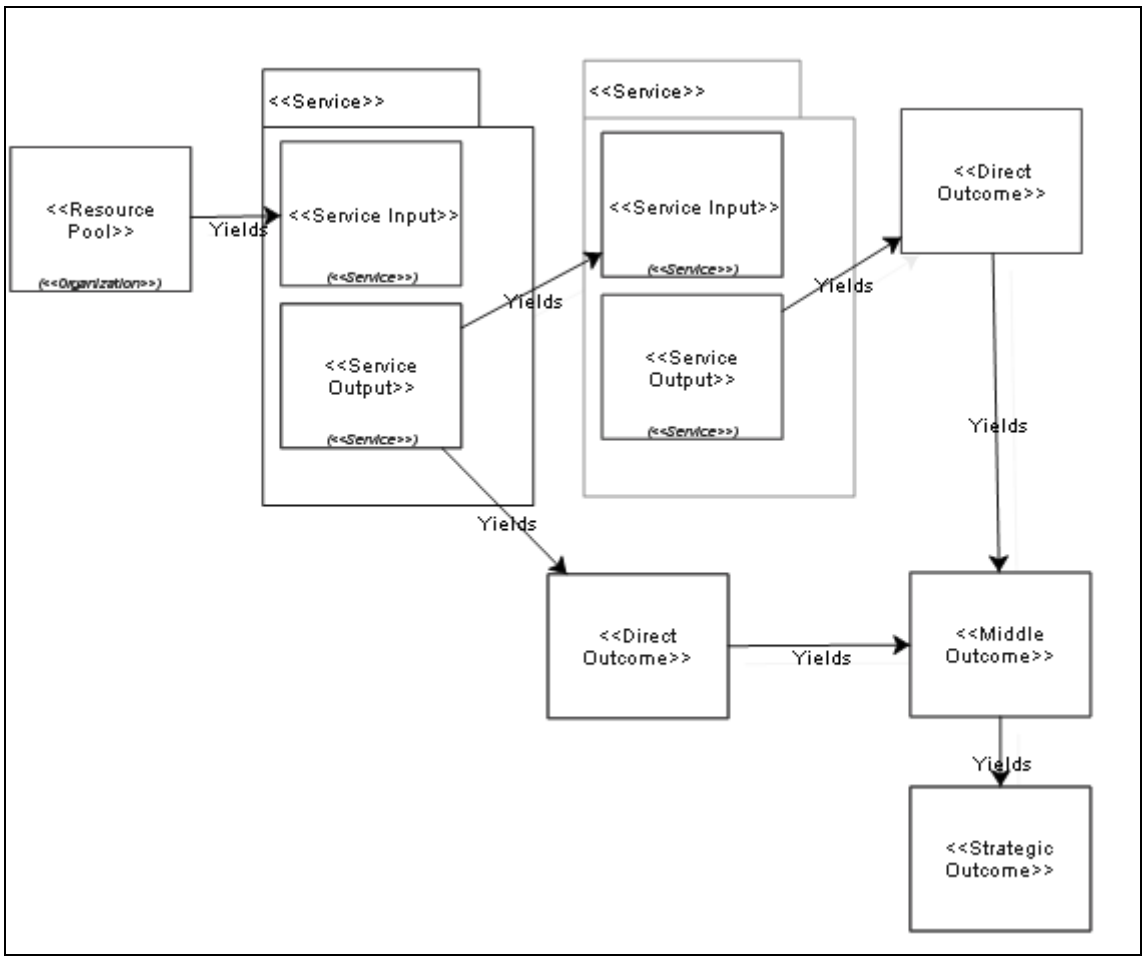

Figure 12 - BusinessDomainView (Program)

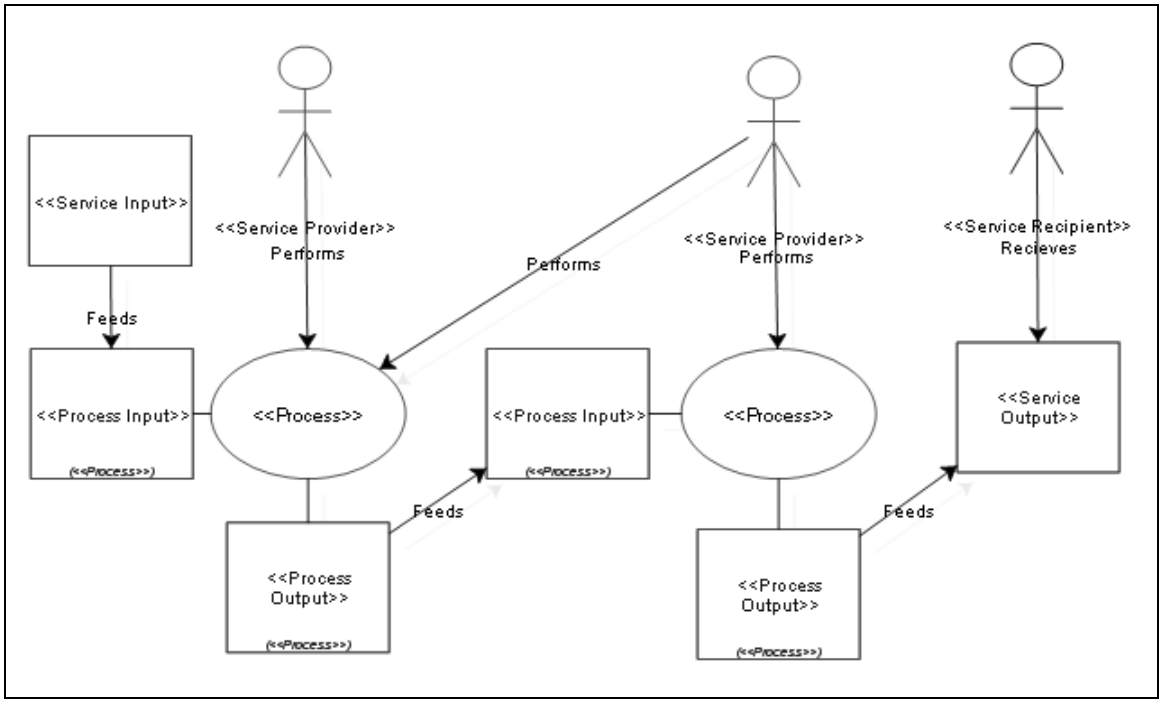

Figure 13 - BusinessDomainView (Service)

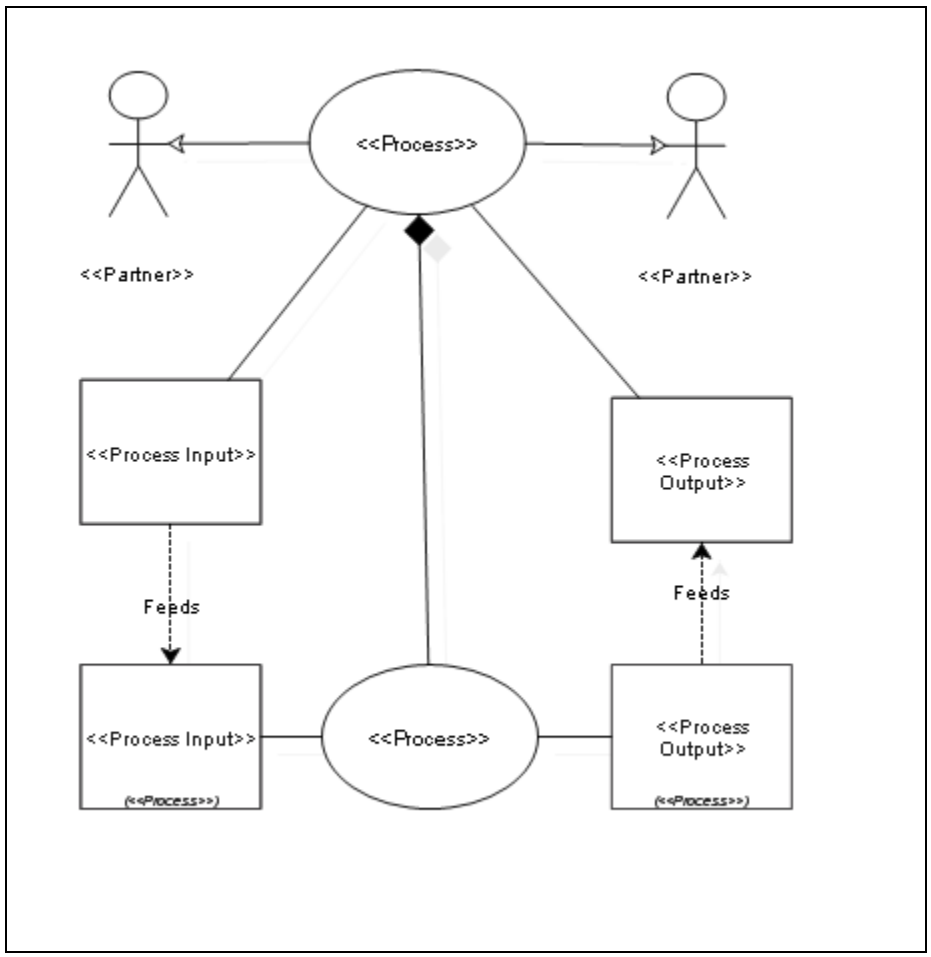

Figure 14 - BusinessDomainView (Collaboration Use Case Process)

Note: For the Collaboration Use Case Process, the Process Input in the above example illustrates an input to a sub process.

5.2. Business Requirements View

5.2.0. General Information

5.2.0.1. Conceptual Overview (informative)

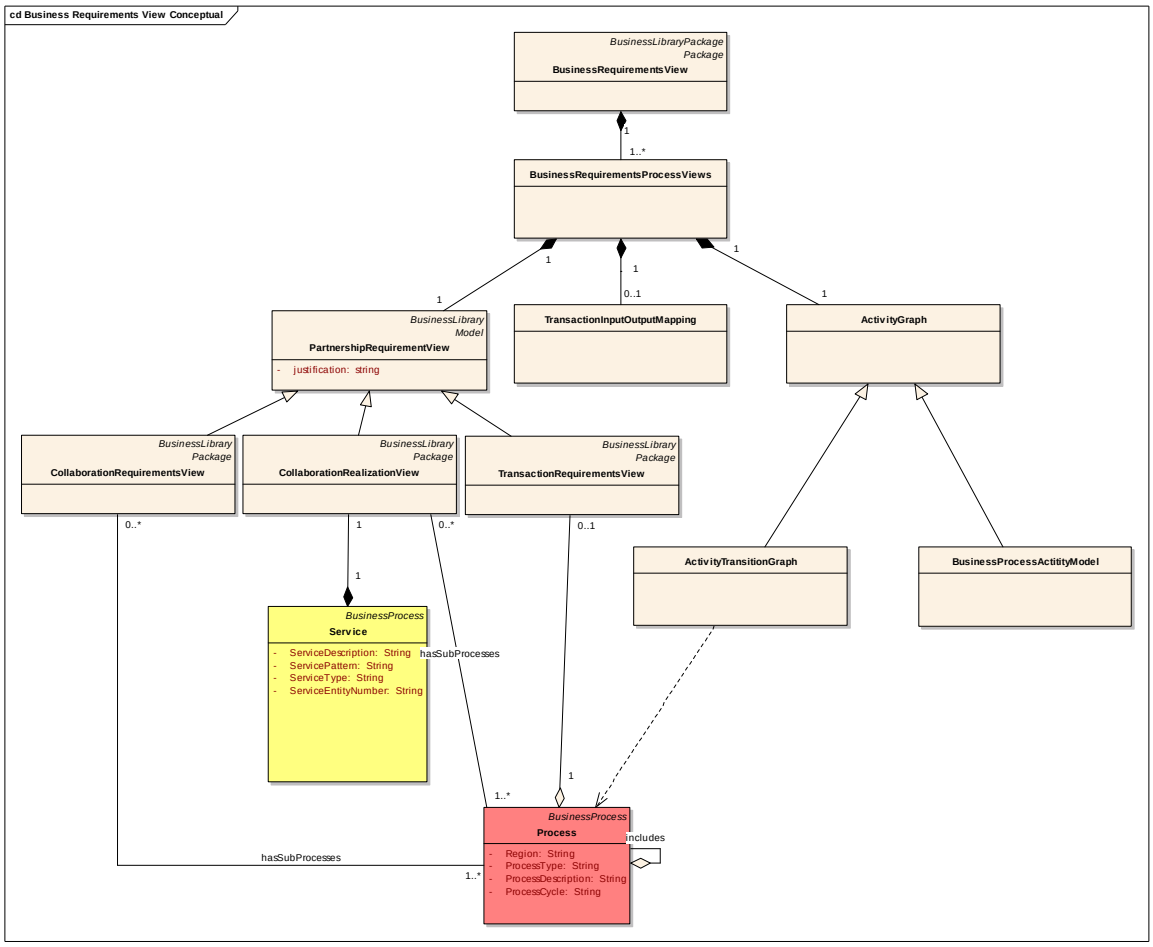

Figure 15 - BusinessRequirementsView - Conceptual Overview

The *BusinessRequirementsView* (BRV) is the second of the 3 views of a UMM complaint *BusinessCollaborationModel*. The goal of the BRV is to identify collaborative business processes between different business partners and to describe the requirements regarding those collaborative business processes. The *BusinessRequirementsView* package serves as a container for artifacts contained within the *BusinessRequirementsProcessView* that helps to capture the requirements of collaborative business processes. To fully understand how the BRV is designed, it is important to understand how processes can be classified by usage. There are three ways a process can be used:

- Transaction use case: The simplest kind of *Process* has no subordinate processes; they will always have at least two *Partner*s (a sender and receiver) and model a single message and (optionally) its immediate response.
- Collaboration use case: Includes a *Process* that has subordinate processes and may include two or more *Partner*. They model complicated process interactions built up from simpler components (either transaction use case or simpler collaboration use case).
- *Service*: Are realizations (*BusinessCollaborationRealization*s in UMM) of complex processes. A *Process* is a reusable abstract object that explains how an activity could occur. A *Service* models how the abstract *Process, Partner* and *ProcessInput* will be bound to *Organization* and *ResourcePool*.

Each of the three process usage types, noted above, will have a related *BusinessRequirementsProcessView* that is bound to it and provides a package of views that describe the *BusinessRequirementsView*. The content of a *BusinessRequirementsProcessView* is dependent on how a *Process* is used (i.e. whether the process is used as a transaction use case, a collaboration use case or a *Service).*

The *BusinessRequirementsProcessView* for a transaction use case *Process* will contain a *BusinessProcessActivityModel,* a *TransactionRequirementsView* and may contain a *TransactionInputOutputMap*:

- A *BusinessProcessActivityModel* is designed to capture a message transmission (and optional response) between a sender and its receiver. Exactly one *BusinessProcessActivityModel* is mandatory for the lowest level process. Thus, the *BusinessRequirementsProcessView* may contain zero or one *BusinessProcessActivityModel.*
- *TransactionRequirementsView* is a simple model that identifies the Partners of the transaction use case. Exactly one *TransactionRequirementsView* is mandatory for each transaction use case, so *BusinessRequirementsProcessView* may contain zero or one *TransactionRequirementsView.*
- *TransactionInputOutputMap* specifies how *ProcessInput* and *ProcessOutput* map to *EntityState* that are described in the related *BusinessProcessActivityModel*. It is possible (but unlikely) that a process is defined without any *ProcessInput* or *ProcessOutput*. Most transaction use cases will have a *ProcessInput, ProcessOutput* or both and then the *TransactionInputOutputMap* will be required to describe their mapping. The *BusinessRequirementsProcessView* may contain zero or one *TransactionInputOutputMap.*

The *BusinessRequirementsProcessView* for collaboration use case *Process* will contain an *ActivityTransitionGraph,* and a *CollaborationRequirementsView*:

- An *ActivityTransitionGraph* is designed to capture the interaction of several processes that comprise a business collaboration use case. Exactly one *ActivityTransitionGraph* is mandatory for the collaboration use case processes. Thus, the *BusinessRequirementsProcessView* may contain zero or one *ActivityTransitionGraph.*
- The *CollaborationRequirementsView* models how each *Partner* of a collaboration use case maps to *Partner* in each process. Exactly one *CollaborationRequirementsView* is mandatory for collaboration use cases, so

BusinessRequirementsProcessView may contain zero or one *CollaborationRequirementsView*.

The *BusinessRequirementsProcessView* for *Service* will contain an *ActivityTransitionGraph*, and a *CollaborationRealizationView*.

- Playing a similar role as for collaboration use cases, the *ActivityTransitionGraph* models the activity flow of an included process. Exactly one *ActivityTransitionGraph* is mandatory for the *Service*s. Thus, the *BusinessRequirementsProcessView* may contain one *ActivityTransitionGraph.*
- The *CollaborationRealizationView* models how the *ServiceRecipient* and *ServiceProvider* of a *Service* map to *Partner* in the included process. Exactly one *CollaborationRealizationView* is mandatory for collaboration use cases, so a *BusinessRequirementsProcessView* may contain zero or one *CollaborationRealizationView*

5.2.0.2. Stereotype and Tag Definitions (normative)

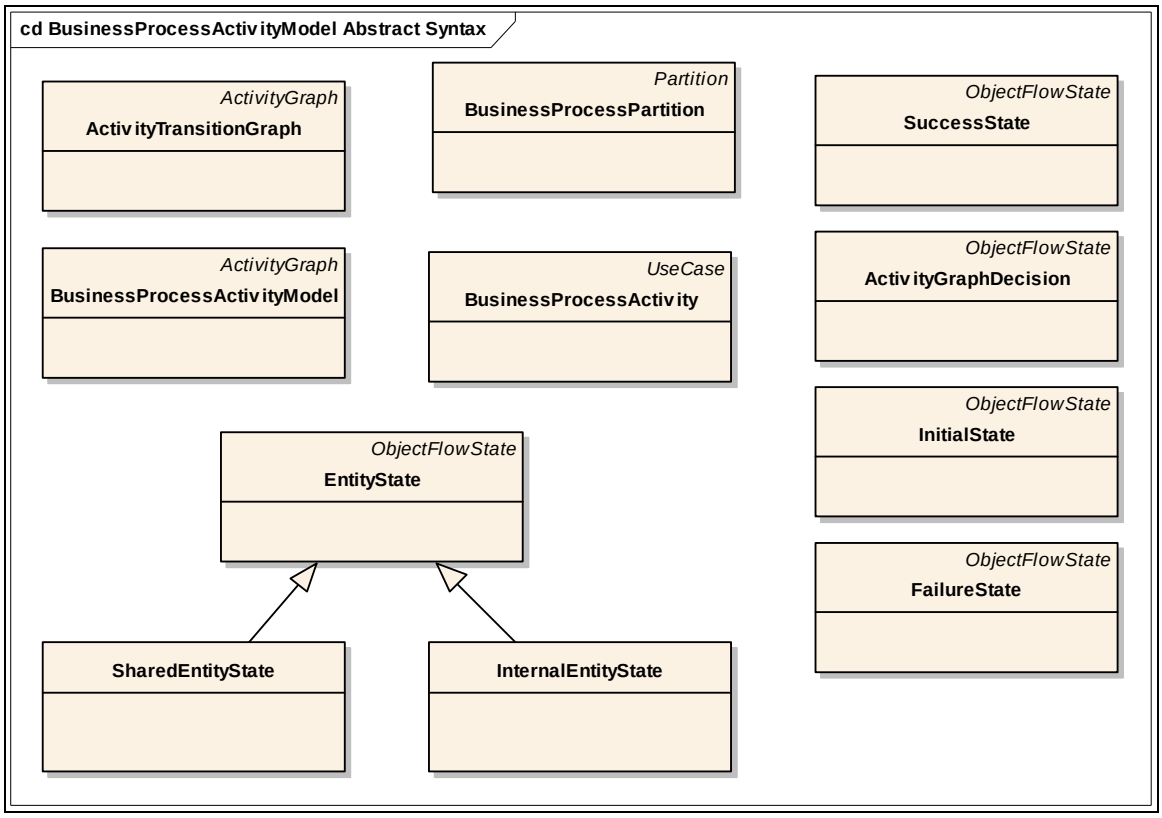

Figure 16 - BusinessRequirementsView - Abstract Syntax

Stereotype	BusinessRequirementsProcessView

Base Class	Package
Parent	**BusinessLibrary (from Base Model)**
Description	The *BusinessRequirementsProcessView* is a container for all the *BusinessRequirementsView* models associated with a specific process.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessProcessActivityModel
Base Class	ActivityGraph
Parent	**BusinessLibrary (from Base Model)**
Description	The *BusinessProcessActivityModel* is a container for elements describing the behavior of an internal transaction use case process.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessEntityView
Base Class	Package
Parent	**BusinessLibrary (from Base Model)**
Description	The *BusinessEntityView* is a container to describe a business entity having significance in the modeled domain, including its business entity lifecycle and business entity states.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	**PartnershipRequirementsView** (abstract)
Base Class	Package
Parent	**BusinessLibrary (from Base Model)**
Description	The *PartnershipRequirementsView* is a container for all elements describing the requirements of a partnership between business partners. These requirements either apply to business collaboration, a business transaction, or the realization of business collaboration. Due to this fact, the *PartnershipRequirementsView* is split into three specializations: the

	CollaborationRequirementsView, the *TransactionRequirementsView*, and the *CollaborationRealizationView*. Since the *PartnershipRequirementsView* is an abstract stereotype, one of its specializations must be used.
Tag Definition	Justification Type: String Multiplicity: 1 Description: Justification of the partnership requirements Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	CollaborationRequirementsView
Base Class	Package
Parent	**PartnershipRequirementsView**
Description	The *CollaborationRequirementsView* is a container for all elements describing the requirements of business collaboration between business partners.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm • justification

Stereotype	TransactionRequirementsView
Base Class	Package
Parent	**PartnershipRequirementsView**
Description	The T*ransactionRequirementsView* is a container for all elements describing the requirements of a business transaction between instances of *Partner*.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm • justification

Stereotype	CollaborationRealizationView
Base Class	Package
Parent	**PartnershipRequirementsView**
Description	The *CollaborationRealizationView* is a container for all elements describing the requirements of a realization of a *BusinessCollaborationUseCase* by business partners.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm • justification

Stereotype	TransactionInputOutputMap
Base Class	None
Parent	**BusinessLibrary (from Base Model)**
Description	The *TransactionInputOutputMap* is a container that describes how *ProcessInput* and *ProcessOutput* map to the entity states associated to the *Process*.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	ActivityTransitionGraph
Base Class	ActivityGraph
Parent	**BusinessLibrary (from Base Model)**
Description	The *ActivityTransitionGraph* is a type of *ActivityGraph* that describes the process flow of a collaboration use case as it transitions from one subordinate *Process* to another.
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

5.2.0.3. Constraints (normative)

A *BusinessRequirementsView* MUST contain at least one *PartnershipRequirementsView* package.
A *BusinessRequirementsView* MUST contain at least one *ActivityGraph* package.
A *BusinessRequirementsView* MAY contain at MOST one *TransactionInputOutputMapping* package.
A *BusinessRequirementsView* MUST contain at least one *CollaborationRequirementsView* package.
A *BusinessRequirementsView* MUST contain at least one *CollaborationRealizationView* package.
A *BusinessRequirementsView* MUST contain at least one *TransactionRequirementsView* package.
A *BusinessRequirementsView* MAY contain at least one *ActivityTransitionGraph* package.
A *BusinessRequirementsView* MAY contain at least one *BusinessProcessActivityModel* package.
A *CollaborationRealizationView MUST* contain at least one *Service* package.
A *CollaborationRealizationView* MAY contain zero or more *Process*
A *TransactionRequirementsView* MAY contain zero or more *Process*
A *CollaborationRequirementsView* MAY contain zero or more *Process*

```
package Model_Management
context Package

inv packagesAllowedInBRV:
  self.isBusinessRequirementsView() implies
  self.contents->forAll
  (
    isPartnershipRequirementsView() or
    isActivityGraph() or
    isTransactionInputOutputMapping() or
    isCollaborationRequirementsView() or
    isCollaborationRealizationView() or
    isTransactionRequirementsView() or
    isActivityTransitionGraph() or
    isBusinessProcessActivityModel() or
    isService() or
    isProcess()
  ) and
  self.contents->exists(isPartnershipRequirementsView) and
  self.contents->exists(isActivityGraph) and
  self.contents->exists(isTransactionInputOutputMapping) and
  self.contents->exists(isCollaborationRealizationView) and
  self.contents->exists(isService)
```

5.2.1. Business Process Activity Model

5.2.1.1. Conceptual Overview (informative)

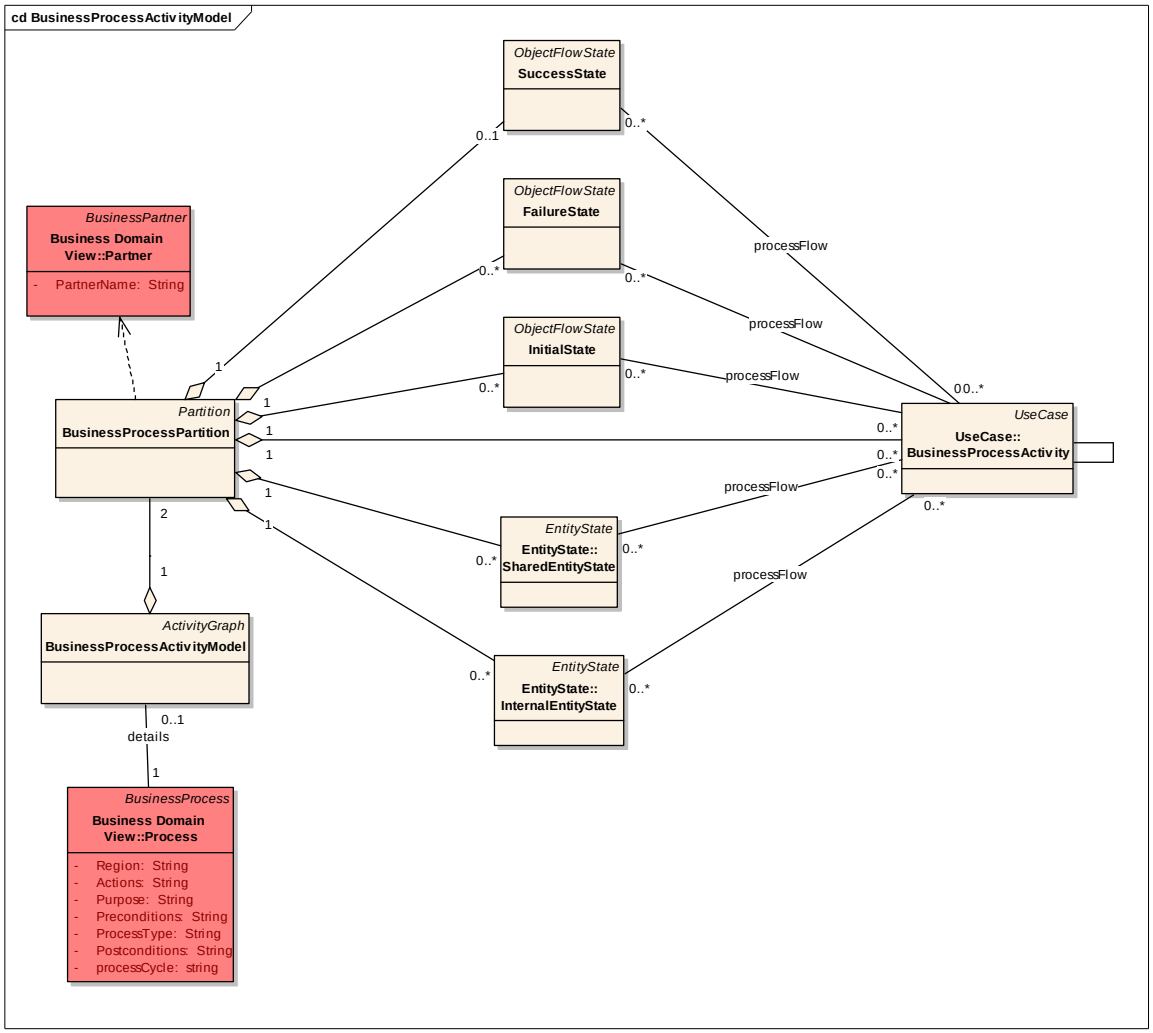

Figure 17 - BusinessProcessActivityModel (BPAM) - Conceptual Overview

The *BusinessProcessActivityModel* gives an overview about the lowest level business processes, their activities and the business partners that execute these activities. A *BusinessProcessActivityModel* gives an overview about the business process and how the *Partner*s perform the process. Each transaction use case *Process* will be associated to exactly one *BusinessProcessActivityModel*, but a *Process* that is not a transaction use case will not have a *BusinessProcessActivityModel*. There is a 0..1 to 1 relationship between *BusinessProcessActivityModel* and *Process*.

A *BusinessProcessActivityModel* is described as a flow of business process activities. A business process activity is assigned to the partition of the business partner executing the activity. The need for a collaborative business process is identified whenever a transition connecting two business process activities crosses between partitions. It follows, that either a *BusinessProcessActivityModel* is composed of one or more

BusinessProcessActivity or a Partition (which is part of a *BusinessProcessActivityModel*) is composed of one or more BusinessProcessActivity. A *BusinessProcessActivity* might be refined by another *BusinessProcessActivityModel*. Thus a *BusinessProcessActivity* is composed of zero or one *BusinessProcessActivityModel* which in turn is a composite of zero or one *BusinessProcessActivity*.

A *BusinessProcessActivityModel* may also denote important states of business entities that are manipulated during the execution of a business process. A business entity state is the output from one business activity and input to another business activity. There is a transition from a business process activity to a business entity state signaling an output as well as a transition from a business entity state to a business process activity signaling an input. Some business entity states are meaningful to one business partner 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 activity model may include both internal and shared business entity states. Hence, a *BusinessProcessActivityModel* is composed of zero to many *InternalBusinessEntityState* and of zero to many *SharedBusinessEntityStates*. A *SharedBusinessEntityState* signals the need for a collaborative business process.

Transaction use case processes always have exactly 2 instances of *Partner*; those instances of *Partner* are represented by a *BusinessProcessPartition* in a *BusinessProcessActivityModel*; there is a 1 to 2 relationship between *BusinessProcessActivityModel* and *BusinessProcessPartition*. A *BusinessProcessPartition* can then contain any number of internal states (of all types including *InitialState*, *SuccessState*, *FailureState* and *InternalEntityState*) and *BusinessProcessActivity*.

- There is a 1 to 0..1 relationship between *BusinessProcessPartition* and *InitialState*.
- There is a 1 to 0..* relationship between *BusinessProcessPartition* and *SuccessState*.
- There is a 1 to 0..* relationship between *BusinessProcessPartition* and *FailureState*.
- There is a 1 to 0..* relationship between *BusinessProcessPartition* and *InternalEntityState*.
- There is a 1 to 0..* relationship between *BusinessProcessPartition* and *BusinessProcessActivity*.

Almost all states exist inside the boundaries of partitions; the exception is the *SharedEntityState* that represent the data transmitted from one *Partner* to another. Those are contained in the *BusinessProcessActivityModel* itself. There must be at least one *SharedEntityState* in each model, but there may be several (representing different initial messages or a choice of responses). There is a 1 to 1..* relationship between *BusinessProcessActivityModel* and *SharedEntityState*.

The expected activity flow is that the *Process* will have one *InitialState*… after the *InitialState* there will be a series of *BusinessProcessActivity* that cause a change of state change in one of the business entities (or to a final *SuccessState* or *FailureState*). It is

expected that each state (except final states) will be followed by a *BusinessProcessActivity*, which is then followed by the final state.

5.2.1.2. Stereotype and Tag Definitions (normative)

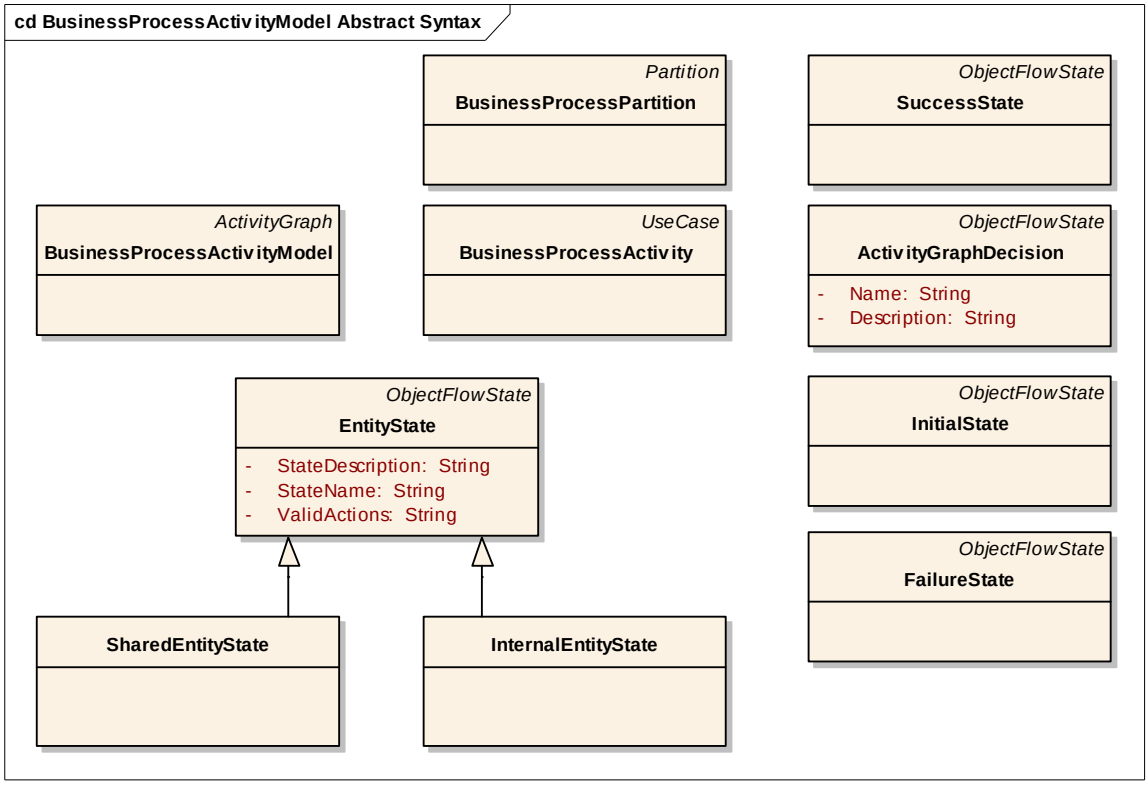

Figure 18 - BusinessProcessActivityModel (BRV) Abstract Syntax

Stereotype	BusinessProcessActivity
Base Class	ActivityGraph
Parent	**None**
Description	A *BusinessProcessActivity* corresponds to a step in the execution of a *BusinessProcessActivityModel*. A Business Activity might be refined by another *BusinessProcessActivityModel*. Thus, the UML base class for *BusinessProcessActivity* is not an atomic action state, but a state – which is a generalization of action and composite state.
Tag Definition	**No tagged values**

Stereotype	SharedEntityState
Base Class	ObjectFlowState
Parent	**None**
Description	The *SharedEntityState* represents a state of a *BusinessEntity* that is shared between the business processes of two involved *Partner*s.
Tag Definition	**No tagged values**

Stereotype	InternalEntityState
Base Class	ObjectFlowState
Parent	**None**

Description	The *InternalEntityState* represents a state of a *BusinessEntity* that is internal to a single *Partner*.
Tag Definition	**No tagged values**

Stereotype	SuccessState
Base Class	ObjectFlowState
Parent	**None**
Description	The *SuccessState* represents the state after successful execution of the *Process*.
Tag Definition	**No tagged values**

Stereotype	FailureState
Base Class	ObjectFlowState
Parent	**None**
Description	*FailureState* represents the state after unsuccessful execution of the *Process*.
Tag Definition	**No tagged values**

Stereotype	InitialState
Base Class	ObjectFlowState
Parent	**None**
Description	*InitialState* represents the state at the immediate start of the *Process*.
Tag Definition	**No tagged values**

Stereotype	ActivityGraphDecision
Base Class	ObjectFlowState
Parent	**None**
Description	*ActivityGraphDecision* represents a decision state where a partner will make a decision about which path the process flow will continue on.
Tag Definition	**PartnershipRequirementsView**

Stereotype	BusinessProcessPartition
Base Class	Partition
Parent	**None**
Description	An instance of a *BusinessProcessPartition* is a partition associated with the instance of *Partner* to a *Process* and contains the *BusinessProcessActivity* and instances of *EntityState* that are internal to a *Partner*.
Tag Definition	**No tagged values**

Stereotype	details
Base Class	Association
Parent	**None**
Description	Describes the association between *Service* and *BusinessProcessActivityModel*
Tag Definition	**No tagged values**

Stereotype	processFlow
Base Class	Association
Parent	**None**
Description	Describes the process flow where a state transitions to a

	BusinessProcessActivity, or where a *BusinessProcessActivity* transitions to a state. There is a *processFlow* association between: • *BusinessProcessActivity* and *InitialState* • *BusinessProcessActivity* and *SuccessState* • *BusinessProcessActivity* and *FailureState* • *BusinessProcessActivity* and *InternalEntityState* • *BusinessProcessActivity* and *SharedEntityState*
Tag Definition	**No tagged values**

5.2.1.3. Constraints (normative)

The *BusinessProcessActivityModel* MUST contain nothing else, but *BusinessProcessActivityModels,* instances of *Partner* or instances of *BusinessProcess* and it must NOT be empty

```
package Model_Management
context Package

inv AllowedElementsInBusinessProcessActivityModel:
  self.isBusinessProcessActivityModel() implies
  self.contents->forAll
  (
    isBusinessProcessActivityModel() or
    isPartner() or
    isBusinessProcess()
  ) and
  self.contents->notEmpty()
```

A *BusinessProcessActivityModel* MUST have partitions.

```
package Behavioral_Elements::State_Machines
context CompositeState

inv BusinessProcessActivityModeslHavePartitions:
  self.stateMachine.isBusinessProcessActivityModel() implies
  self.stateMachine.oclAsType(ActivityGraph).partition-> notEmpty())
```

A partition in a *BusinessProcessActivityModel* MUST contain one or more instances of *BusinessProcessActivity* and MAY contain *InternalBusinessEntityState, PseudoState, FinalState* and *Transition*

```
package Behavioral_Elements::Activity_Graphs
context Partition

inv AllowedModelElementsInBusinessProcessActivityModelPartition:
  self.isPartition() implies
  self.contents->forAll
  (
    isBusinessProcessActivity()
    or isInternalBusinessEntityState()
    or isPseudoStateOrFinalStateOrTransition()
    or isResourceState()
  ) and
```

```
  self.contents->exists(isBusinessProcessActivity())
```

5.2.1.4. OCL Methods used in Business Process Activity Model (normative)

OCL-Methods

```
package Foundation::Core
context ModelElement

--Predefined method which evaluates, if the given Modelelement
--has a stereotype equal to the passed name
def:
let hasStereotype (st : String) : Boolean =
  self.stereotype->select(cst | cst.name = st)->notEmpty()

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

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

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

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

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

-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is fork
```

```
def:
let isFork() : Boolean =
  self.oclAsType(Pseudostate).kind = PseudostateKind::fork and
  self.oclIsKindOf(Pseudostate)

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

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

-- Returns true if the type of the element 'Transition'
def:
let isTransition() : Boolean =
  self.oclIsKindOf(Transition)

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

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

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

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

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

--return true if the given element is a partition
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-- checks if a Actor is stereotyped a Partner
def :
let isPartner() : Boolean =
  self.oclIsTypeOf(Partner) and
  self.hasStereotype(‘Partner’)
```

```
-- checks if a Class is stereotyped a Resource
def :
let isResource() : Boolean =
  self.oclIsTypeOf(Class) and
  self.hasStereotype('Resource')
```

5.2.1.5. Example (informative)

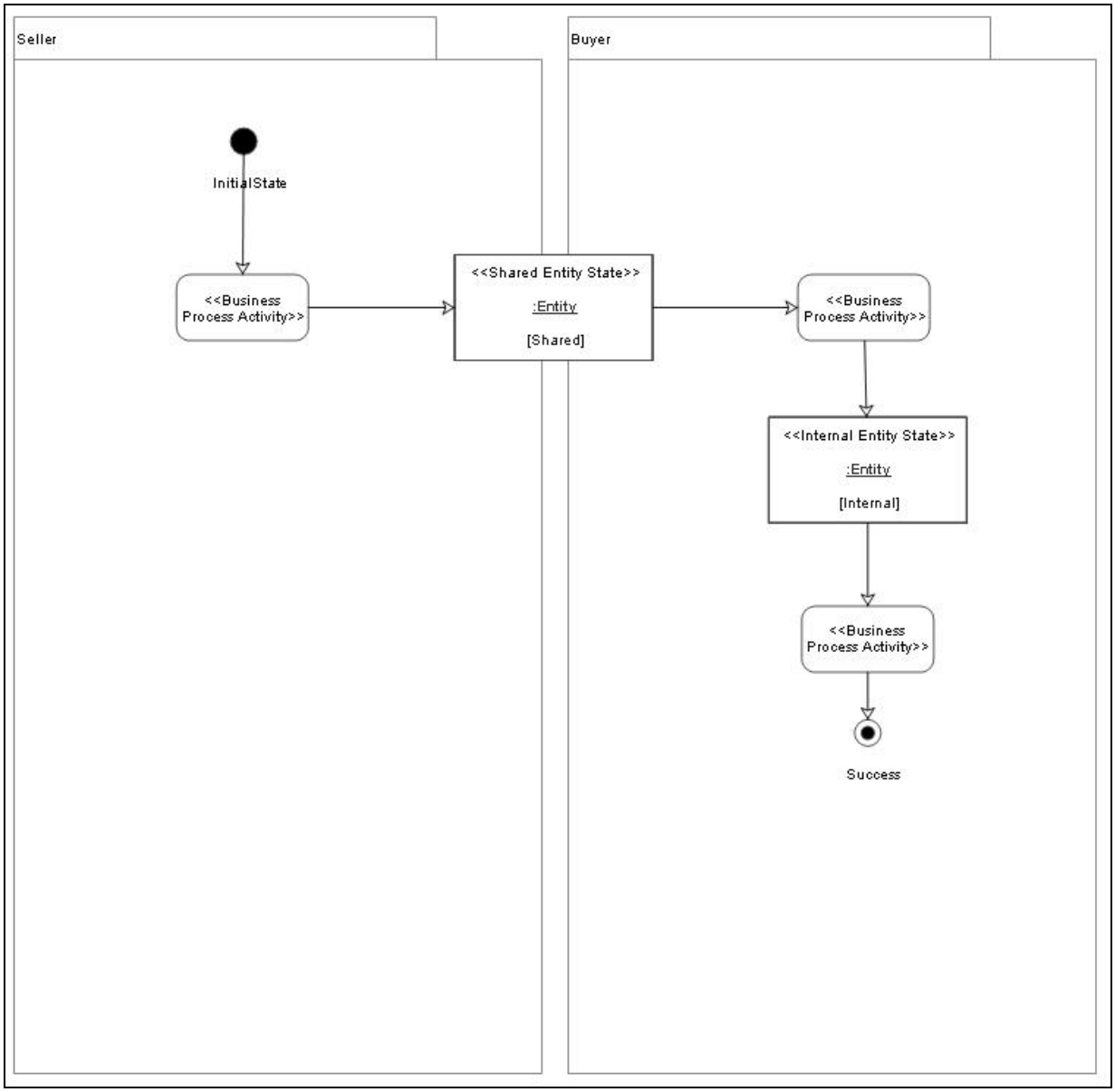

Figure 19 - Example Business Process Activity Model

5.2.2. Activity Transition Graph

5.2.2.1. Conceptual Overview (Informative)

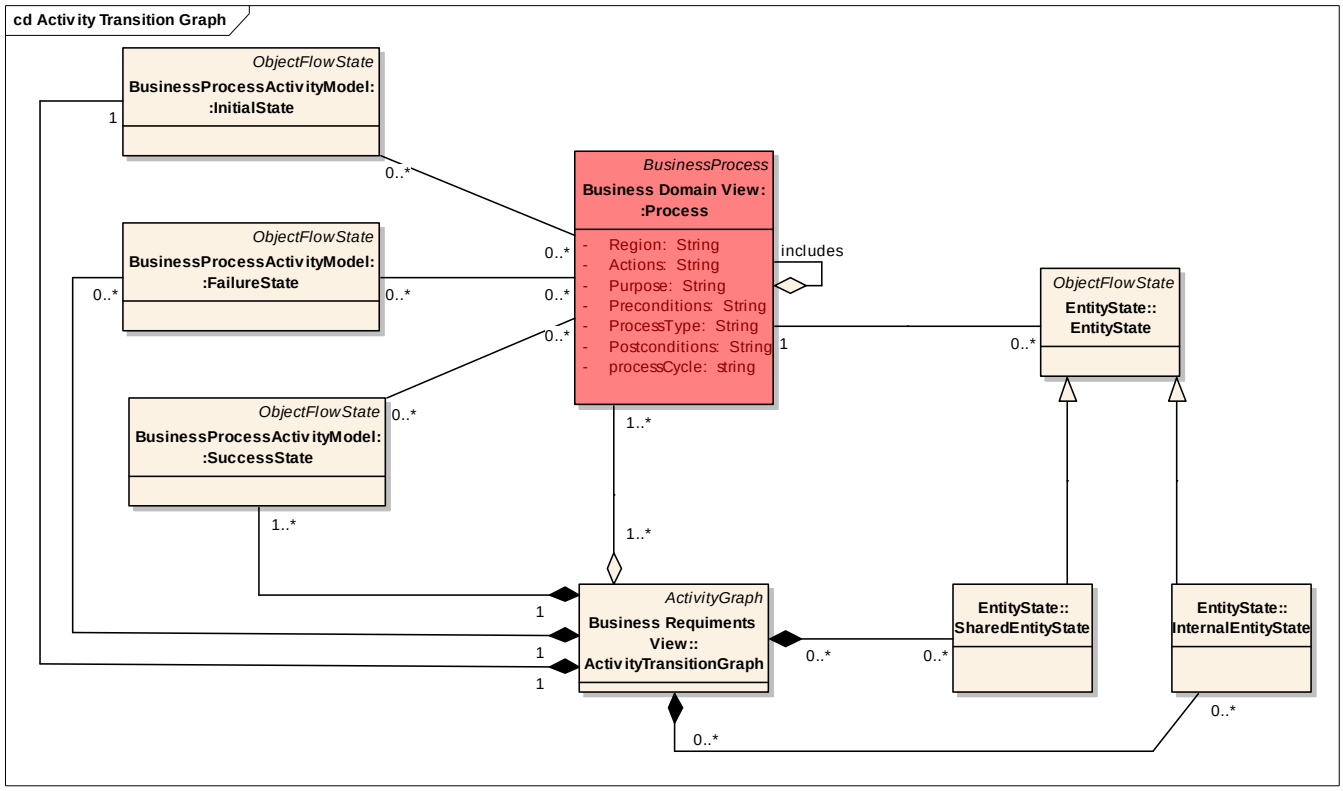

Figure 20 - Activity Transition Graph - Conceptual Overview

While a *BusinessProcessActivityModel* provides a lot of detail for transaction use cases, collaboration use cases are often far too complicated to be comprehensible when presented in that type of model. Instead, collaboration use cases (and instances of *Service*) require a more abstract model. An *ActivityTransitionGraph* allows for modeling most of the data required to show how a *Process* navigates through its included *Process*, but it does fail to capture who is performing the roles of different instances of *Partner*. That information is captured for processes in the mandatory *CollaborationRequirementsView*, and for the instance of *Service* it is captured in the mandatory *CollaborationRealizationView*.

Instead of a complete flow diagram, the *ActivityTransitionGraph* focuses on showing the flow between the *BusinessPraocessActivity*. Each *ActivityTransitionGraph* is dependant on one *Process* that it models. The *ActivityTransitionGraph* then contains all the *BusinessPraocessActivity* of the *Process* it is dependant on. There may be several such instances of *Process* and each *Process* can be part of one or more *ActivityTransitionGraph*; there is a 0..* to 0..* relationship between *ActivityTransitionGraph* and *Process*.

Each *ActivityTransitionGraph* can have a variety of states. It will have one *InitialState*, and will have one or more final states (*FailureState*, *SuccessState*). It will also have to capture the states where transitions occur from one included process to another, so it will contain a variety of *SharedEntityState* and/or *InternalEntityState*.

- There is a 1 to 1 relationship between *ActivityTransitionGraph* and *InitialState*.
- There is a 1 to 0..* relationship between *ActivityTransitionGraph* and *FailureState*.
- There is a 1 to 1..* relationship between *ActivityTransitionGraph* and *SuccessState*.
- There is a 0..* to 0..* relationship between *ActivityTransitionGraph* and *SharedEntityState*.
- There is a 0..* to 0..* relationship between *ActivityTransitionGraph* and *InternalEntityState*.

The expected flow is that the *ActivityTransitionGraph* will have one *InitialState* that will be related to one of the included processes, that included process will then run until a specific *EntityState* is reached. When that *EntityState* is reached another *BusinessPraocessActivity* can be started. Each transition from one *BusinessPraocessActivity* to another happens when the first *BusinessPraocessActivity* reaches a specific *EntityState* (either an *InternalEntityState* or a *SharedEntityState*). The *EntityState* that fires the transition from one *BusinessPraocessActivity* to another are captured in the *ActivityTransitionGraph* model, and are related to the *Process*. Each *Process* can have zero or more transitioning instances of *EntityState*, each transition *EntityState* will belong to one process; there is a 1 to 0..* relationship between *Process* and *EntityState* (which will be either a *SharedEntityState* or an *InternalEntityState*).

Note: The UMM standard contains a *BusinessChoreographyView*; in GSRM this View was slightly modified and moved to this layer as an *ActivityTransitionGraph*.

5.2.2.2. Stereotype and Tag Definitions (normative)

Stereotype	Process	
Base Class	UseCase	
Parent	**BusinessProcess**	
Description	UMM defines a process as “a set of related activities that together create value”. GSRM defines a process as “a set of related activities that together create a valued output”. A business process might be performed by a single partner or by multiple partners crossing organizational boundaries. In cases where organizations collaborate in a process, the process should create a valued output for all its participants.	
Tag Definition	processName	
	Type	String
	Multiplicity	1
	Description	The name of the process.
	processCycle	
	Type	Enum
	Multiplicity	1
	Description	The Process Cycle is one of four (4) Phases in the lifetime of a process. These are the rough equivalent to the REA Phases within UMM.
	Options	• Planning Processes • Provisioning Processes • Delivery Processes • Decommissioning/Deregistering Processes
	processType	
	Type	Enum
	Multiplicity	1
	Description	Process type values come from Service Reference Process Patterns, and are from the Level 1 Service Pattern. The process type is a valid value within one of the process cycles.
	Options	• Planning Processes (cycle description only) o Recognize service Planning cycle o Recognize service contingency event o Forecast service demand o Forecast service risks o Set performance targets for service, processes, resources o Measure performance of service, processes, resources o Estimate service resource requirements o Allocate resources to service processes • Provisioning Processes (cycle description only) o Monitor service resource consumption o Monitor service resource availability o Configure service processes to respond to demand or supply level limits o Configure service processes to respond to contingency event o Source service resources o Register and equip service suppliers

- Acquire and register service resources
- Pay for service resources
- Maintain service resources
- Deploy service resources geographically
- Set service schedule
- Configure service resources
- Protect service resources
- Promote services
- Monitor and mitigate service risks
- Process service complaints
- Register and equip service target group members

- Delivery Processes (cycle description only)
  - Register request for service delivery
  - Qualify request for service delivery
  - Set service delivery schedule and notify
  - Open service delivery case
  - Allocate resources to service output
  - Deploy resources for service output
  - Produce service output
  - Deliver service output
  - Collect and account for a service output fee
  - Process service exceptions
  - Register service output
  - Maintain service output
  - Close service delivery case
- Decommissioning/Deregistering Processes (cycle description only)
  - Decommission/deregister service output
  - Decommission/deregister service resources
  - Deregister service suppliers
  - Deregister service target group members

processDescription	
Type	String
Multiplicity	1
Description	A description of the process

Region	
Type	String
Multiplicity	1
Description	The region benefiting from the process

Inherited tagged values:

- purpose
- actions
- preconditions
- postconditions

Stereotype	InitialState
Base Class	ObjectFlowState
Parent	**None**
Description	The InitialState is used to identify the start point of the sequence of events which make up an *ActivityTransitionGraph*
Tag Definition	**No tagged values**

Stereotype	FailureState
Base Class	ObjectFlowState
Parent	**None**
Description	A failure state is used to denote that the business process was unsuccessful
Tag Definition	**No tagged values**

Stereotype	SuccessState
Base Class	ObjectFlowState
Parent	**None**
Description	A success state is used to denote that the activity between the business partners was successful from the business perspective, not from the technology perspective
Tag Definition	**No tagged values**

Stereotype	EntityState
Base Class	ObjectFlowState
Parent	**None**
Description	An EntityState represents the state of a class of information
Tag Definition	**No tagged values**

Stereotype	SharedEntityState
Base Class	EntityState
Parent	**None**
Description	A *SharedEntityState* represents the state of a class of business information which is exchanged between the parties, as part of the activity which makes up an *ActivityTransitionGraph*
Tag Definition	**No tagged values**

Stereotype	InternalEntityState
Base Class	Entity
Parent	**None**
Description	An *InternalEntityState* represents the state of a class of business information which is used/consumed/operated on, by only one business partner as part of the activity which makes up an *ActivityTransitionGraph*
Tag Definition	**No tagged values**

5.2.2.3. Constraints (normative)

An *ActivityTransitionGraph* MUST **have 1 *InitialState***
An *ActivityTransitionGraph* MAY **have 1 or more *FailureState***
An *ActivityTransitionGraph* MUST **have at least 1 *SuccessState***
An *ActivityTransitionGraph* MAY **have 1 or more *SharedEntityState***
An *ActivityTransitionGraph* MAY **have 1 or more *InternalEntityState***

```
package Model_Management
context ActivityTransitionGraph

inv AllowedElementsInActivityTransitionGraph:
  self.isBusinessProcessActivityModel() implies
  self.contents->forAll
  (
    isInitialState() or
    isFailureState() or
    isSuccessState() or
    isSharedEntityState() or
    isInternalEntityState() or
    isProcess()
  ) and
  self.contents->notEmpty()
```

5.2.2.4. OCL Methods used in Activity Transition Graph (normative)

OCL-Methods

```
package Foundation::Core
context ModelElement


--Predefined method which evaluates, if the given Modelelement
--has a stereotype equal to the passed name
def:
let hasStereotype (st : String) : Boolean =
self.stereotype->select(cst | cst.name = st)->notEmpty()
--Predefined method which evaluates, if the given element
--has the stereotype 'InternalEntityState'
def:
let isInternalEntityState() : Boolean =
self.oclIsKindOf(ObjectFlowState) and
self.hasStereotype('InternalEntityState')
--Predefined method which evaluates, if the given element
--has the stereotype 'SharedEntityState'
def:
let isSharedEntityState() : Boolean =
self.oclIsKindOf(ObjectFlowState) and
self.hasStereotype('SharedEntityState')
--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessProcess'
def:
let isBusinessProcess() : Boolean =
self.oclIsKindOf(ObjectFlowState) and
self.hasStereotype('BusinessProcess')
-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is initial
def:
let isInitialState() : Boolean =
self.oclAsType(Pseudostate).kind = PseudostateKind::initial and
self.oclIsKindOf(Pseudostate)
-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is choice
def:
let isChoice() : Boolean =
self.oclAsType(Pseudostate).kind = PseudostateKind::choice and
self.oclIsKindOf(Pseudostate)
-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
-- is fork
def:
let isFork() : Boolean =
self.oclAsType(Pseudostate).kind = PseudostateKind::fork and
self.oclIsKindOf(Pseudostate)
-- Returns true if the type of the element or one of the
-- supertypes is 'PseudoKindState' and its Pseudostatekind
```

```
-- is join
def:
let isJoin() : Boolean =
self.oclAsType(Pseudostate).kind = PseudostateKind::join and
self.oclIsKindOf(Pseudostate)
-- Returns true if the type of the element or is 'FinalState'
def:
let isFinalState() : Boolean =
self.oclIsKindOf(FinalState)
-- Returns true if the type of the element 'Transition'
def:
let isTransition() : Boolean =
self.oclIsKindOf(Transition)
--Returns true if the element is a standard-element of an ActivityGraph
def:
let isPseudoStateOrFinalStateOrTransition() : Boolean =
isInitialState() or isChoice() or isFork() or isJoin() or
isTransition()
or isFinalState()
--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessProcessActivityModel'
def:
let isBusinessProcessActivityModel() : Boolean =
self.oclIsKindOf(ActivityGraph) and
self.hasStereotype('BusinessProcessActivityModel')
--return true if the given element is a partition
def:
let isPartition() : Boolean =
self.oclIsKindOf(Partition)
--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntity'
def :
let isBusinessEntity() : Boolean =
self.oclIsKindOf(Class) and
self.hasStereotype('BusinessEntity')
--Predefined method which evaluates, if the given element
--has the stereotype 'BusinessEntityState'
def :
let isEntityState() : Boolean =
self.oclIsKindOf(State) and
self.hasStereotype('EntityState')
```

5.2.2.5. Example (Informative)

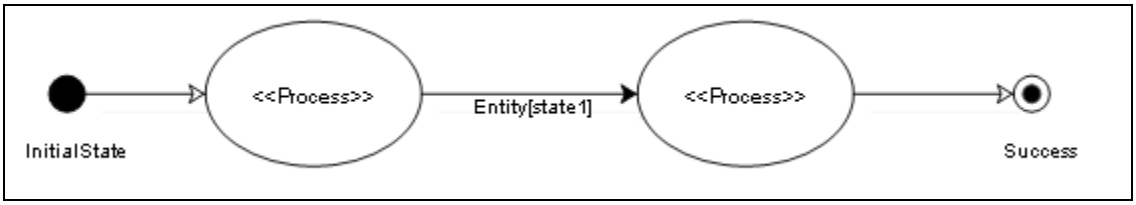

Figure 21 - Activity Transition Graph

5.2.3. Business Entity View

5.2.3.1. Conceptual Overview (informative)

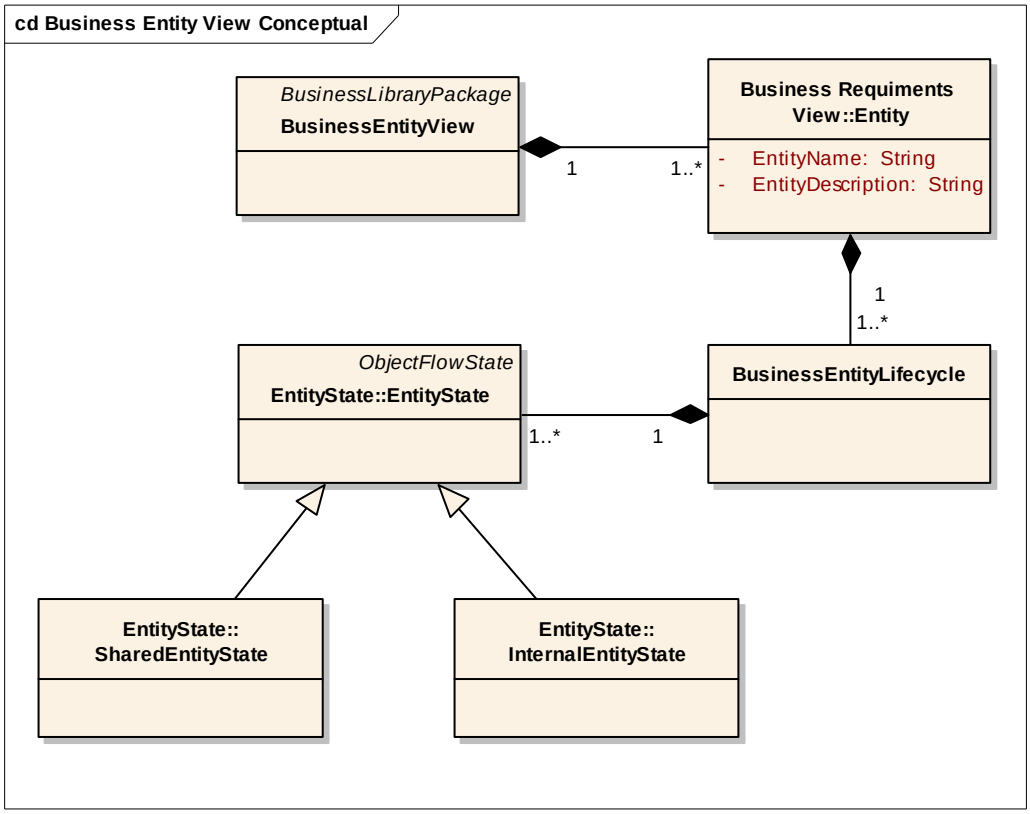

Figure 22 - BusinessEntityView (BRV) - Conceptual Overview

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

5.2.3.2. Stereotype and Tag Definitions (normative)

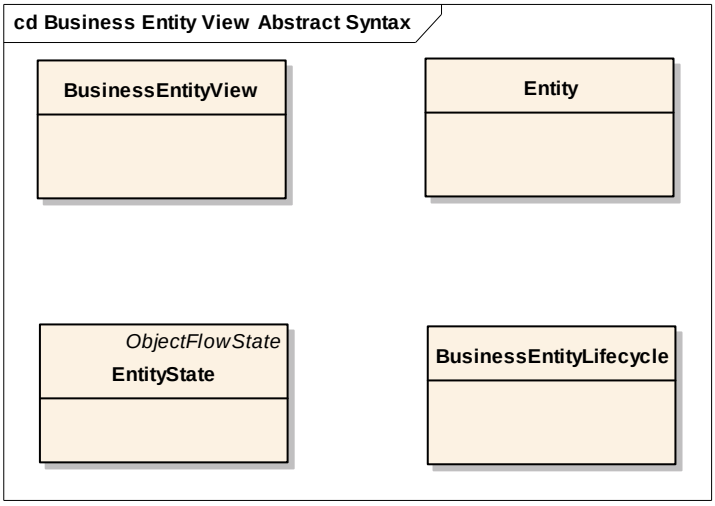

Figure 23 - BusinessEntityView (BRV) - Abstract Syntax

Stereotype	Entity
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 partners in a collaborative business process (e.g. order, account, etc.)
Tag Definition	**No tagged values**

Stereotype	BusinessEntityLifecycle
Base Class	StateMachine
Parent	**N/A**
Description	A business entity lifecycle represents the different business entity states in which a business entity can exist, and the events and transitions that lead from one business entity state to another business entity state of that business entity
Tag Definition	**No tagged values**

Stereotype	EntityState
Base Class	State
Parent	**N/A**
Description	An entity state represents a certain state a business entity can exist in during its lifecycle (an “order” can exist in the states “issued”, “rejected”, “confirmed”, etc.)
Tag Definition	**No tagged values**

5.2.3.3. Constraints (normative)

The *BusinessEntityView* MUST contain nothing else than instances of *Entity*

```
package Model_Management
context Package

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

An *Entity* has zero or one *BusinessEntityLifecycle* that expresses its behavior

```
package Foundation::Core
context Class

inv LifecyclesOfEntity:
  self.isEntity() implies
  self.behavior->select(isBusinessEntityLifecycle())->size()<=1
```

A *BusinessEntityLifecycle* MUST only contain *EntityState*, *PseudoState*, *FinalState* or *Transition*

```
package Behavioral_Elements::State_Machines
context CompositeState

inv ContainsOnlyEntityStates:
  self.stateMachine.isBusinessEntityLifecycle() implies
  self.subvertex->forAll
  (
    isEntityState() or
    isPseudoStateOrFinalStateOrTransition()
  ) and
  self.subvertex->exists(isBusinessEntityState())
```

5.2.3.4. Example (informative)

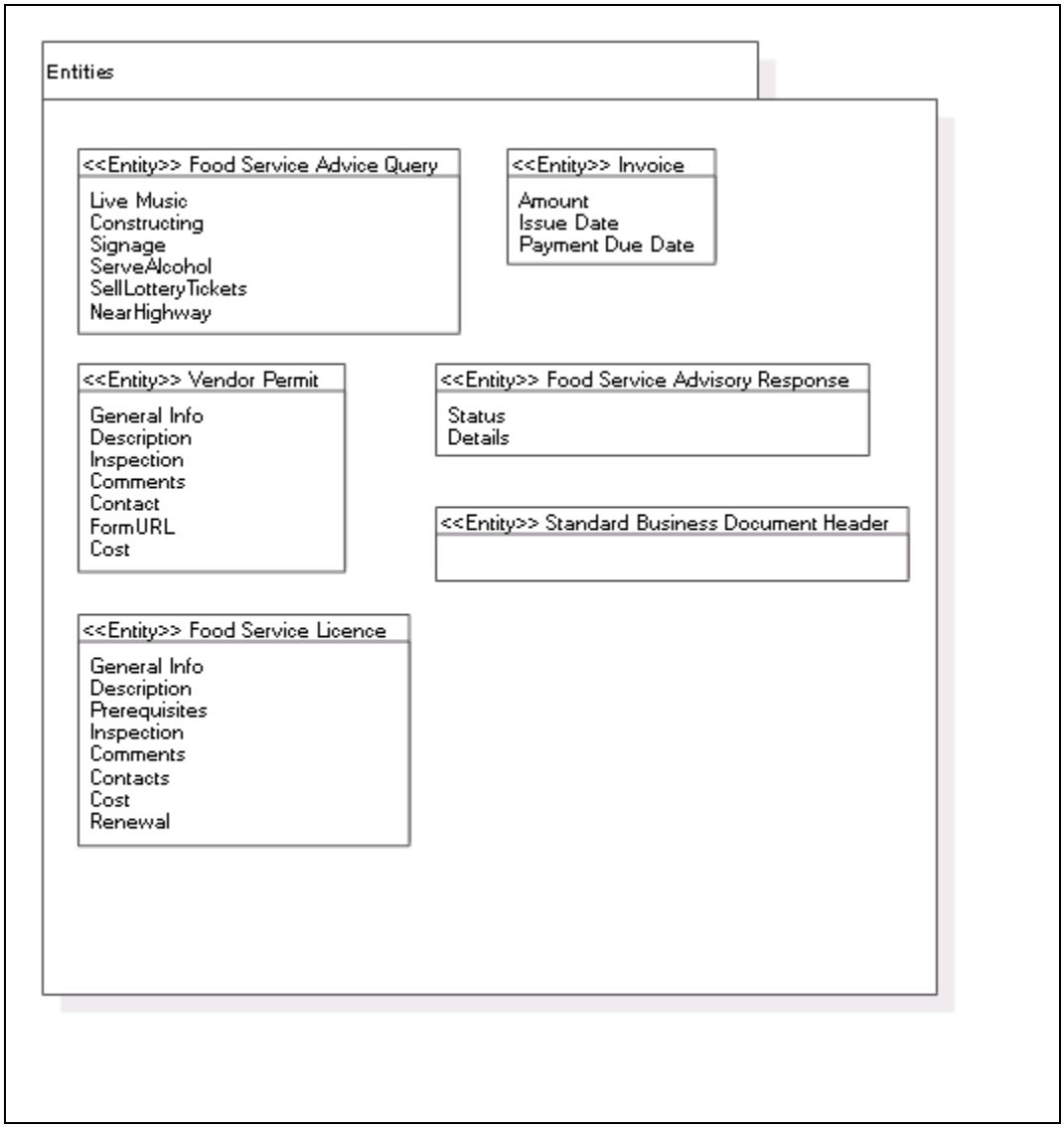

Figure 24 - Example Business Entities View - Entities

Note: The example of the entities is informative not definitive.

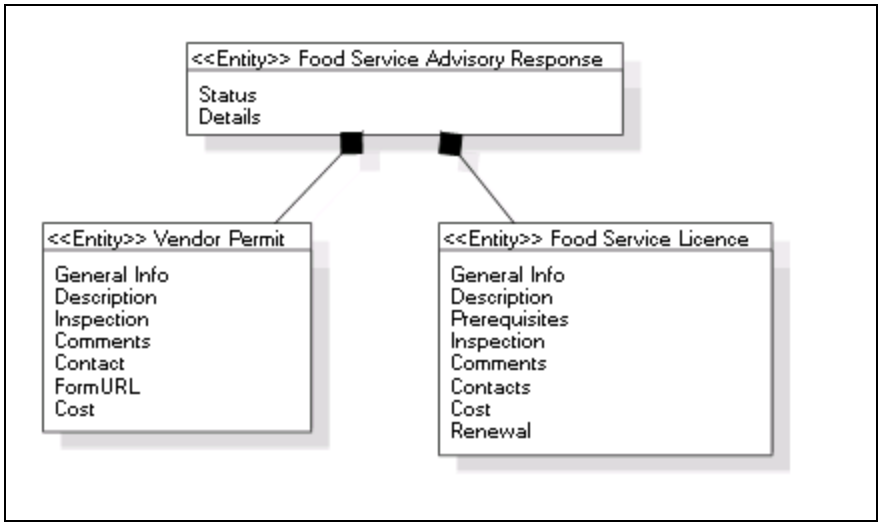

Figure 25 - Example Business Entity View - Entity

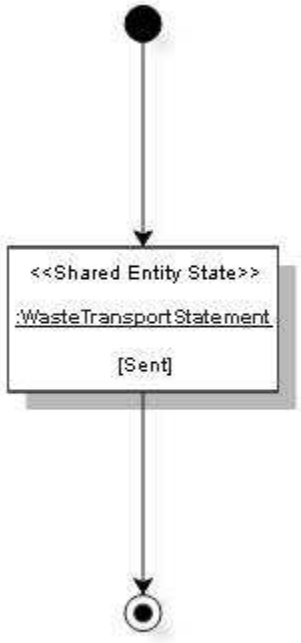

Figure 26 - BusinessEntityView (Entity Lifecycle)

5.2.4. Partnership Requirements View

5.2.4.1. Conceptual Overview (informative)

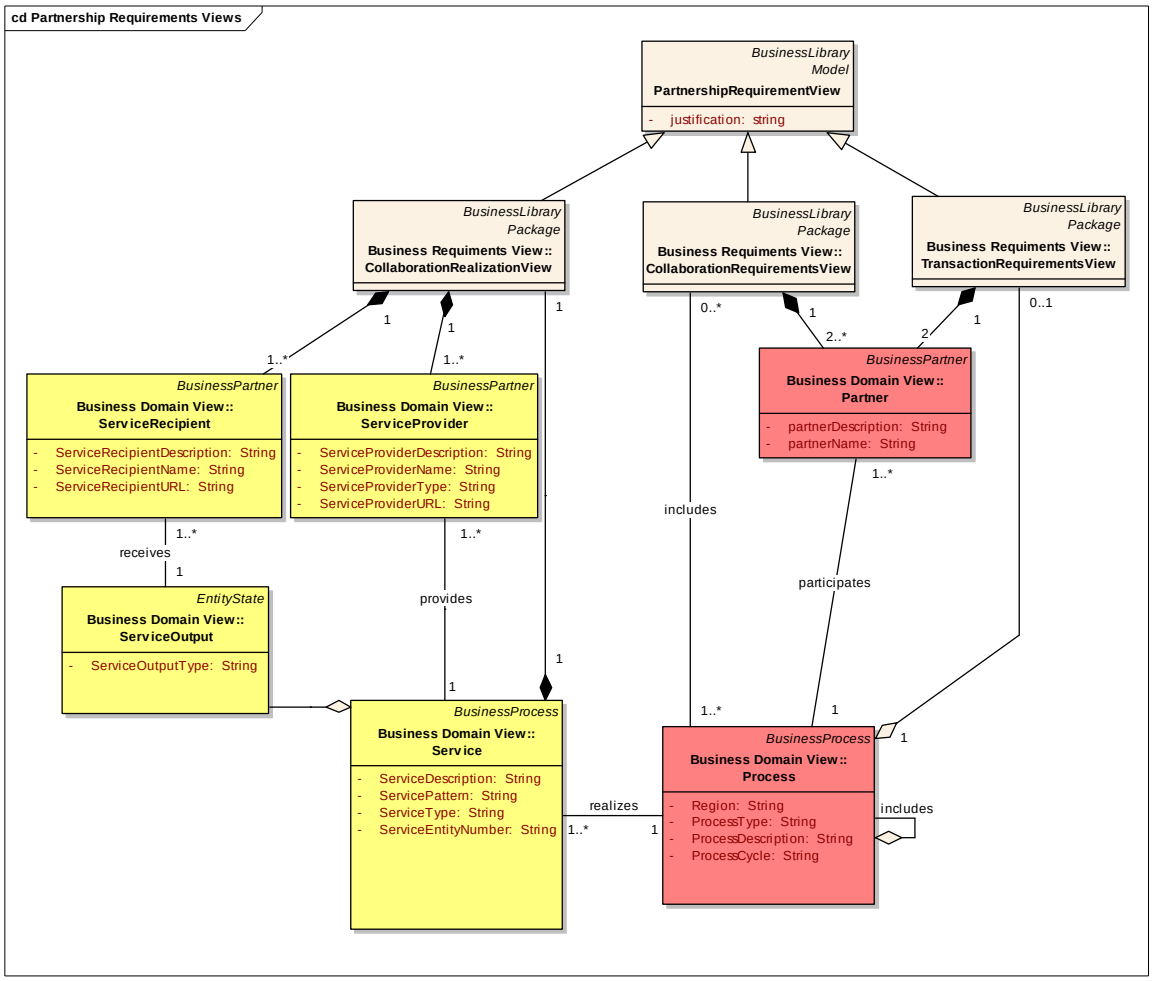

Figure 27 - PartnershipRequirementsView (BRV) - Conceptual Overview

While the previous views (*BusinessProcessActivityModel* and *ActivityTransitionGraph*) were primarily designed to model process flow, they fail to properly capture the relationships between instances of *Partner* and instances of *Process*. In particular how a *Partner* in a *Process* maps to the *Partner* in that *Process*'s included instances of *Process*. A *PartnershipRequirementsView* is designed to capture these *Partner* relations.

In the previous section (see *BusinessProcessActivityModel*) instances of *Process* were categorized on the basis of how they are used. Each category (transaction usecase, collaboration usecase, and Service) has its own *PartnershipRequirementsView* subtype.

Transaction use cases have their partnerships modeled by a *TransactionRequirementsView*. The *TransactionRequirementsView* simply shows the

instances of *Partner* involved in the *Process*. If the *Process* is a transaction use case, then it will have exactly one *TransactionRequirementsView* uniquely associated with it; if it is not a transaction use case then it will not have a *TransactionRequirementsView*. There is a 1 to 0..1 relationship between *Process* and *TransactionRequirementsView*.

A transaction use case has exactly two instances of *Partner* which will be captured in the *TransactionRequirementsView*; there is a 2 to 1 relationship between *Partner* and *TransactionRequirementsView*.

Collaboration use cases have their partnerships modeled by a *CollaborationRequirementsView*. The *CollaborationRequirementsView* shows the *Process* and the instances of *Partner*, as well as its included instances of *Process* and *Partner*.

A *CollaborationRequirementsView* will always have at least two instances of *Partner*, but could have many more; each *Partner* could be included in several *CollaborationRequirementsView* (the one for its *Process* AND for all *Process* that include its *Process*). There is a 2..* to 1..* relationship between *Partner* and *CollaborationRequirementsView*.

A *CollaborationRequirementsView* will also contain included *Process*, there will always be at least one included *Process* (in order to be a collaboration) but there may be several; Any *Process* that is not linked by an *include* association will not be included in any *CollaborationRequirementsView* but its' own. There is a 1..* to 0..* relationship between *CollaborationRequirementsView* and *Process*.

Instances of *Service* have their partnerships modeled by a *CollaborationRealizationView* (the GSRM concept of *Service* is related to the UMM concept of *CollaborationRealization*). The *CollaborationRealizationView* shows the *ServiceProvider* and instances of *ServiceRecipient* associated with the *Service* and indicate how they relate to the *Partner*s of the included instances of *Process*. Each *Service* will have a unique *CollaborationRealizationView;* there is a 1 to 1 relationship between *Service* and *CollaborationRealizationView*.

A *Service* will have at least one *ServiceProvider*, and possibly many more. There is a 1 to 1..* relationship between *CollaborationRealizationView* and *ServiceProvider*. A *Service* will have at least one *ServiceRecipient*, and possibly many more. There is a 1 to 1..* relationship between *CollaborationRealizationView* and *ServiceRecipient*.

A *Service* requires at least one and possibly several instances of *Process*; these included instances of *Process* are contained in the *CollaborationRealizationView*. There is a 0..* to 1..* relationship between *Process* and *CollaborationRealizationView*.

Included instances of *Process* will have instances of *Partner* that also need to be represented in the *CollaborationRealizationView*, so that the relationship between instances of *ServiceProvider*, *ServiceRecipient* and *Partner* can be modeled. Each *Partner* in a *Process* subordinate to the *Service* will be included, so there will be at least 2 instances of *Partner* and at least one *Process* which will have at least 2 instances of

Partner) and possibly many more. There is a 0..* to 2..* relationship between *Partner* and *CollaborationRealizationView*.

5.2.4.2. Stereotype and Tag Definitions (normative)

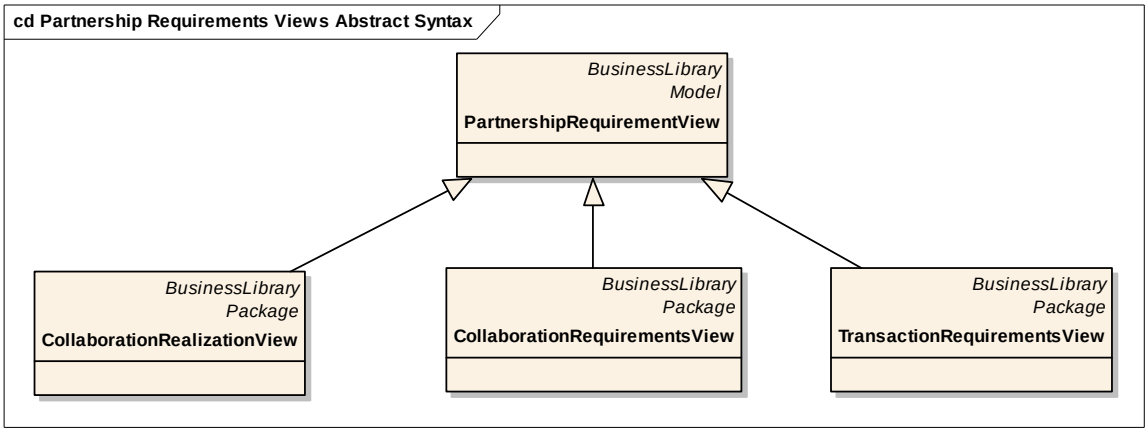

Figure 28 - PartnershipRequirementsView (BRV) - Abstract Syntax

Stereotype	PartnershipRequirementsView
Base Class	Package
Parent	**BusinessLibrary**
Description	A *PartnernshipRequirementsView* is an abstract model that captures the relationship between instances of *Partner* and instances of *Process*, instances of *Process* and instances of subordinate *Process*, and the instances of *Partner* in the parent *Process* relationships to *Partern* in instances of subordinate *Process*.
Tag Definition	**No tagged values**

Stereotype	CollaborationRealizationView
Base Class	Package
Parent	**BusinessLibrary**
Description	*CollaborationRealizationView* is a type of *PartnernshipRequirementsView* that captures the *Partner* and *Process* relationships for a *Service* (which is related to the UMM concept of *CollaborationRealization*).
Tag Definition	**No tagged values**

Stereotype	CollaborationRequirementsView
Base Class	Package
Parent	**BusinessLibrary**
Description	*CollabarationRequirementsView* is a type of *PartnernshipRequirementsView* that captures the *Partner* and *Process* relationships for a collaboration use case *Process* (a *Process* that has subordinate processes)
Tag Definition	**No tagged values**

Stereotype	TransactionRequirementsView
Base Class	Package
Parent	**BusinessLibrary**
Description	*TransactionRequirementsView* is a type of *PartnernshipRequirementsView* that captures the *Partner* and *Process* relationships for a transaction use case *Process* (a *Process* that has **no** subordinate processes)
Tag Definition	**No tagged values**

Stereotype	includes
Base Class	Association
Parent	N/A
Description	Describes the relationship between *CollaborationRequirementsView* or *CollaborationRealizationView* and *Process*, where a *CollaborationRequirementsView* will model all subordinate instances of *Process* of its associated *Process*.
Tag Definition	**No tagged Values**

5.2.4.3. Constraints (normative)

The *CollaborationRequirementView* MUST contain exactly one *BusinessCollaborationUseCase,* at least two instances of *Partner,* and at least two *participates* associations.

```
package Model_Management
context Package

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

The *TransactionRequirementsView* MUST contain exactly one *BusinessTransactionUseCase* , exactly two instances of *Partner,* and exactly two *participates* associations

```
package Model_Management
context Package

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

The *CollaborationRealizationView* MUST contain exactly one *BusinessCollaborationRealization,* at least two instances of *Partner*, and at least two *participates* associations

```
package Model_Management
context Package

inv AllowedElementsInRealizationView:
  self.isCollaborationRealizationView() implies
  self.contents->notEmpty() and
  self.contents->select(isPartner())->size()>=2 and
  self.contents->one(isBusinessCollaborationRealization()) and
  self.contents->select(isParticipates())->size()>=2 and
  self.contents->forAll
  (
    isBusinessCollaborationRealization() or
    isParticipates() or
    isPartner()
  )
```

A *BusinessCollaborationUseCase* MUST be associated with two or more instances of *Partner* via stereotyped binary *participate* associations

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BusinessCollaborationUCAssociatedWith2Partners:
  self.isBusinessCollaborationUseCase() implies
  self.associations->size() >= 2 and
  self.associations->forAll
  (
    a | a.isParticipates() and
    a.allConnections->exists(isPartner()) and
    a.connection->size=2
  )
```

A *BusinessTransactionUseCase* MUST be associated with exactly two instances of *Partner* via stereotyped binary *participate* associations

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BusinessTransactionUCAssociatedWith2Partners:
  self.isBusinessTransactionUseCase() implies
  self.associations->size() = 2 and
  self.associations->forAll
  (
    a | a.isParticipates() and
    a.allConnections->exists(isPartner()) and
    a.connection->size=2
  )
```

A *BusinessCollaborationRealization* MUST be associated with two or more instances of *Partner* via stereotyped binary *participate* associations

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BusinessCollaborationRealizationAssociatedWith2Partners:
  self.isBusinessCollaborationRealization() implies
  self.associations->size() >= 2 and
  self.associations->forAll
  (
    a | a.isParticipates() and
    a.allConnections->exists(isPartner())
    and a.connection->size=2
  )
```

A *BusinessCollaborationRealization* MUST be the client of exactly one realization dependency to a *BusinessCollaborationUseCase*

```
package Behavioral_Elements::Use_Cases
context UseCase

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

A *BusinessCollaborationUseCase* MUST include one or more other *BusinessCollaborationUseCase* or one or more *BusinessTransactionUseCase*, but at least one of them.

```
package Behavioral_Elements::Use_Cases
context UseCase

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

A *BusinessTransactionUseCase* MUST not include further *UseCase*.

```
package Behavioral_Elements::Use_Cases
context UseCase

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

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

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BTUCIncludedAtLeastOnce:
  self.isBusinessTransactionUseCase() implies
  self.include->forAll(base.isBusinesscollaborationusecase()) and
  self.include->collect(base)->notEmpty()
```

A *BusinessCollaborationUseCase* and a *BusinessTransactionUseCase* MUST not be source or target of an *extend* association

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BTUC_BCUC_IsNoExtendTarget:
  (
    self.isBusinessTransactionUseCase() or
    self.isBusinessCollaborationUseCase()
  ) implies
  self.extend->isEmpty()
```

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

```
package Behavioral_Elements::Use_Cases
context UseCase

inv BusinessCollaborationRealizationNoIncludesAndExtends:
  self.isBusinessCollaborationRealization() implies
  self.extend->isEmpty() and
  self.include->isEmpty()
```

All dependencies from/to an instance of *Partner* must be *mapsTo* dependencies.

```
Package Behavioral_Elements::Use_Cases
context Actor

inv AllDependenciesToAndFromPartnerMustBeMapsTo:
  self.isPartner() implies
  self.clientDependency->forAll
  (
    d | d.isMapsToDependency()) and
    self.supplierDependency->forAll(s | s.isMapsToDependency()
  )
```

5.2.4.4. Example (informative)

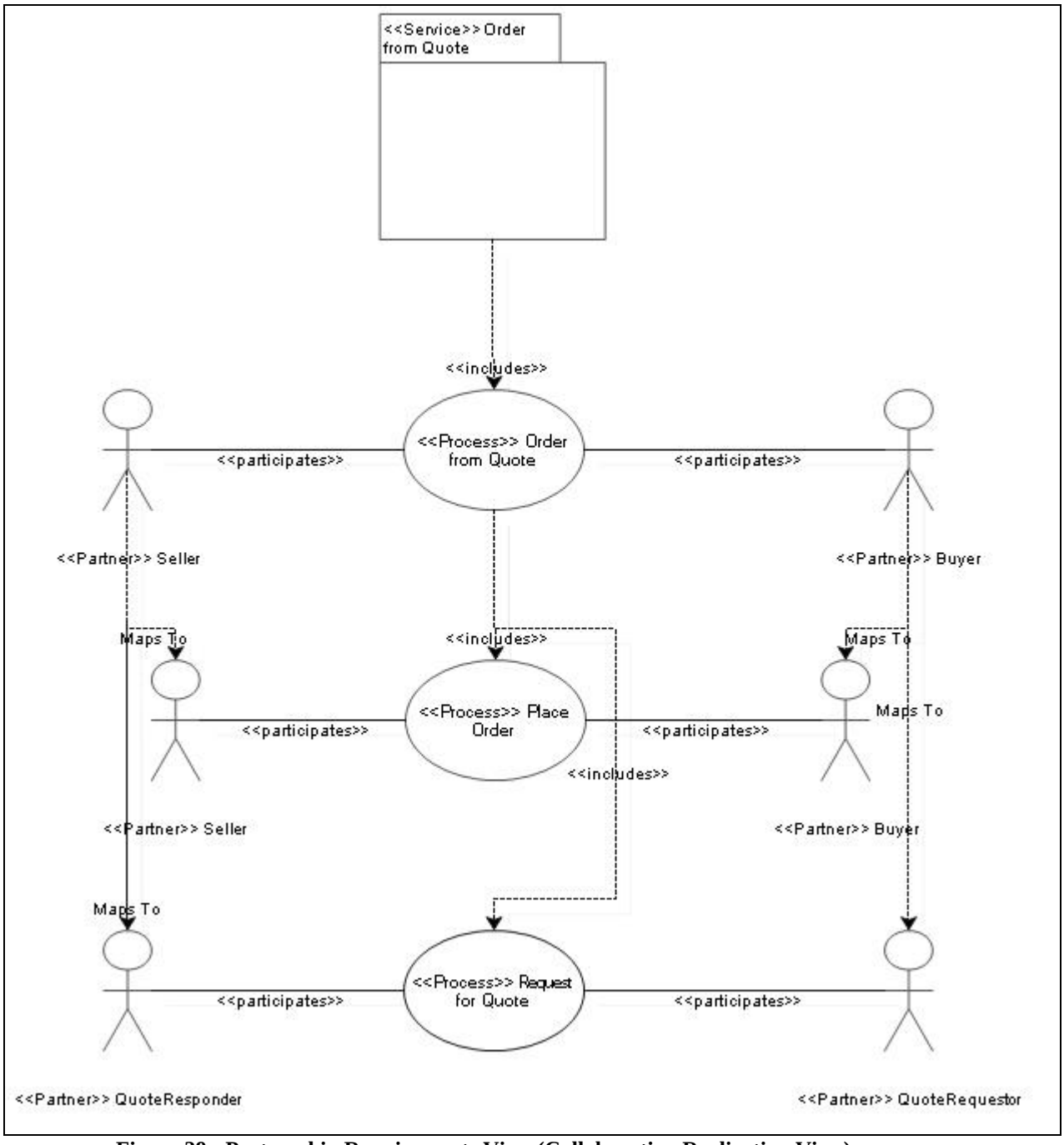

Figure 29 - Partnership Requirements View (Collaboration Realization View)

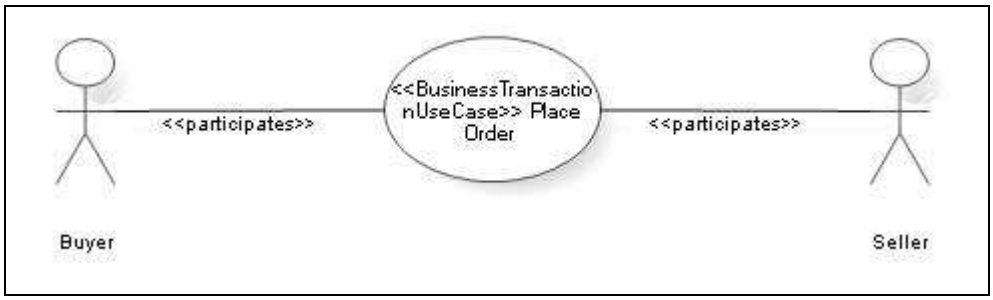

Figure 30 - Partnership Requirements View (Transaction Requirements View)

5.3. Business Transaction View

5.3.0. General information

5.3.0.1. Transaction Pattern Types

The following diagram shows the "truth tree" which breaks down the requirements for a business transaction, finishing with the appropriate transaction type:

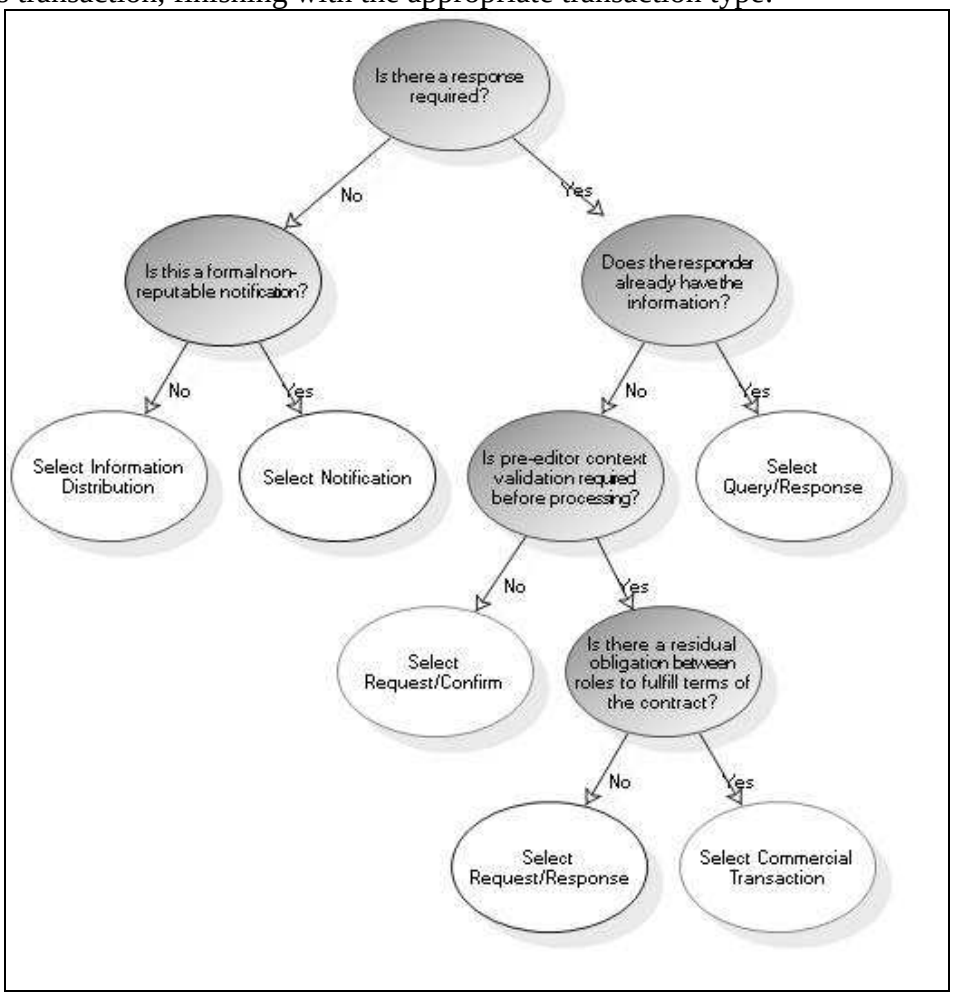

Figure 31 - Business Transaction View - Tranasction Types

5.3.0.2. Conceptual Information (informative)

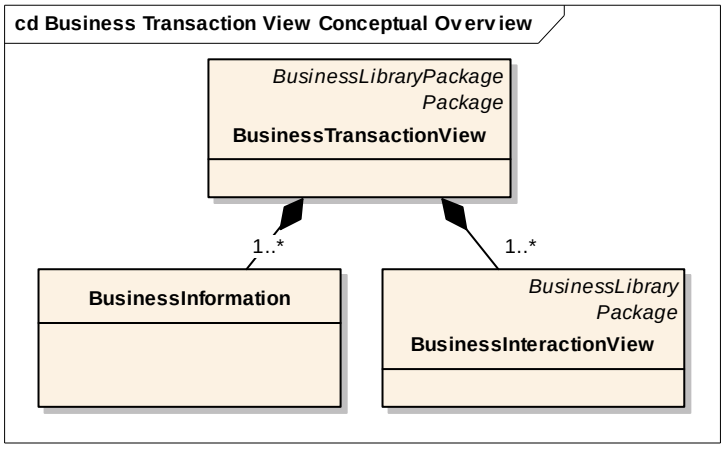

Figure 32 - BusinessTransactionView - Conceptual Overview

The *BusinessTransactionView* (BTV) is an elaboration on the *BusinessRequirementsView* from a business analyst's perspective. A BTV defines choreography of information exchanges. The *BusinessTransactionView* package is a container for two different artifacts that together describe the overall choreography of information exchanges. A *BusinessInteractionView* 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 *BusinessInteractionView* captures artifacts that define a flow in accordance to the requirements of a corresponding *TransactionRequirementsView* of the BRV. A *BusinessInformationView* is a container of artifacts that describe the information exchanged in an interaction. The *BusinessInteractionView* deals with artifacts describing the dynamic aspects of collaboration. The *BusinessInformationView* deals with artifacts describing the structural aspects of the processes. Each of the two views must occur at least once in the *BusinessTransactionView*. Thus the *BusinessTransactionView* is composed of one to many instances of *BusinessInteractionView*, and of one to many instances of *BusinessInformationView*.

Note: The UMM standard also contains a *BusinessChoreographyView*; in GSRM this View was slightly modified and moved to the BRV layer as an ActivityTransactionGraph.

5.3.0.3. Stereotype and Tag Definitions (normative)

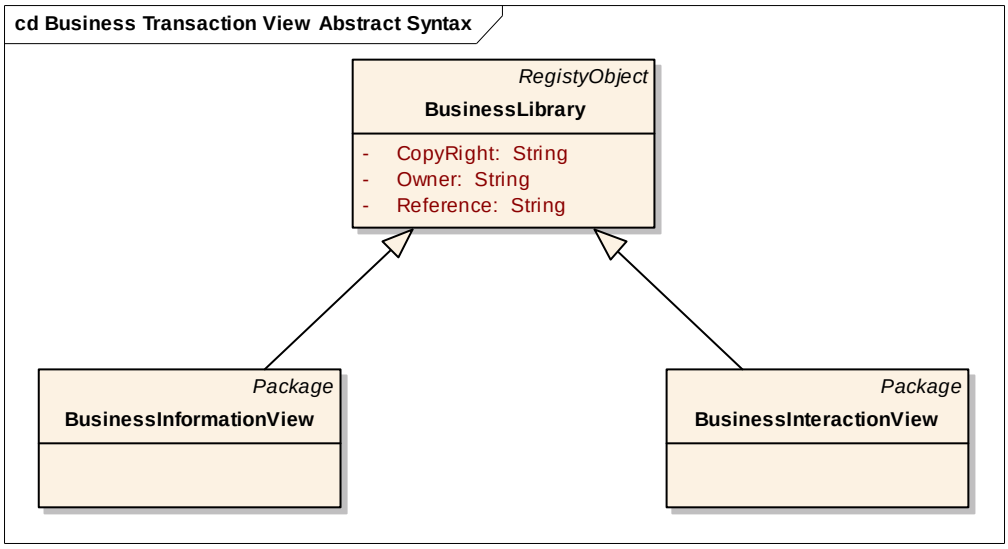

Figure 33 - BusinessTransactionView - Abstract Syntax

Stereotype	BusinessInteractionView
Base Class	Package
Parent	**BusinessLibrary (from BaseModule)**
Description	A *BusinessInteractionView* is a container for artifacts that define a choreography leading to synchronized states of business entities at both sides of the interaction
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

Stereotype	BusinessInformationView
Base Class	Package
Parent	**BusinessLibrary (from BaseModule)**
Description	A *BusinessInformationView* is a container for artifacts that describe the information exchanged in an interaction
Tag Definition	Inherited tagged values: • baseURN • owner • copyright • reference • version • status • businessTerm

5.3.0.4. Constraints (normative)

A BusinessTransaction*View* MUST contain at least one *BusinessInteractionView* package, and at least one *BusinessInformationView* package.

```
package Model_Management
context Package

inv packagesAllowedInBTV:
self.isBusinessTransactionView() implies
self.contents->exists(isBusinessInteractionView()) and
self.contents->exists(isBusinessInformationView())
```

5.3.0.5. OCL Methods used in Business Transaction View (normative)

OCL-Methods

```
package Foundation::Core
context ModelElement

--Predefined method whichs evaluates, if the given Modelelement
--has a stereotype equal to the passed name
def :
let hasStereotype (st : String) : Boolean =
self.stereotype->select(self.name = st)->notEmpty()

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

--Predefined method whichs evaluates, if the given element
--is a subtype of 'BusinessInteractionBehavior'
def :
let isBusinessInteractionBehavior() : Boolean =
self.oclIsKindOf(ActivityGraph) and
self.hasStereotype('BusinessTransaction')

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

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

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

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

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

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

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

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

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

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

```
or
self.hasTaggedValue('BusinessTransactionType','InformationDistributi
onActivity')

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

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

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

-- Returns true if the type of the element is Transition
def:
let isTransition() : Boolean =
self.oclIsKindOf(Transition)


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

--Returns true if a package is stereotyped as
BusinessTransactionView
def:
let isBusinessTransactionView() : Boolean =
```

```
self.hasStereotype('BusinessTransactionView') and
oclIsKindOf(Package)

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

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

-- Returns true if the stereotype of the given element is
'InformationEntitiy'
-- and if the type of the element is Class
def :
let isInformationEntity() : Boolean =
self.hasStereotype('InformationEntity') and
self.oclIsKindOf(Class)

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

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

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

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

--Returns true if the stereotype of the element
-- is 'RequestingInformationEnvelope'
def :
```

```
let isRequestingInformationEnvelope() : Boolean =
self.hasStereotype('RequestingInformationEnvelope') and
oclIsKindOf(ObjectFlowState)

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

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

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

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


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

5.3.1. Business Interaction View

5.3.1.1. Conceptual Overview (informative)

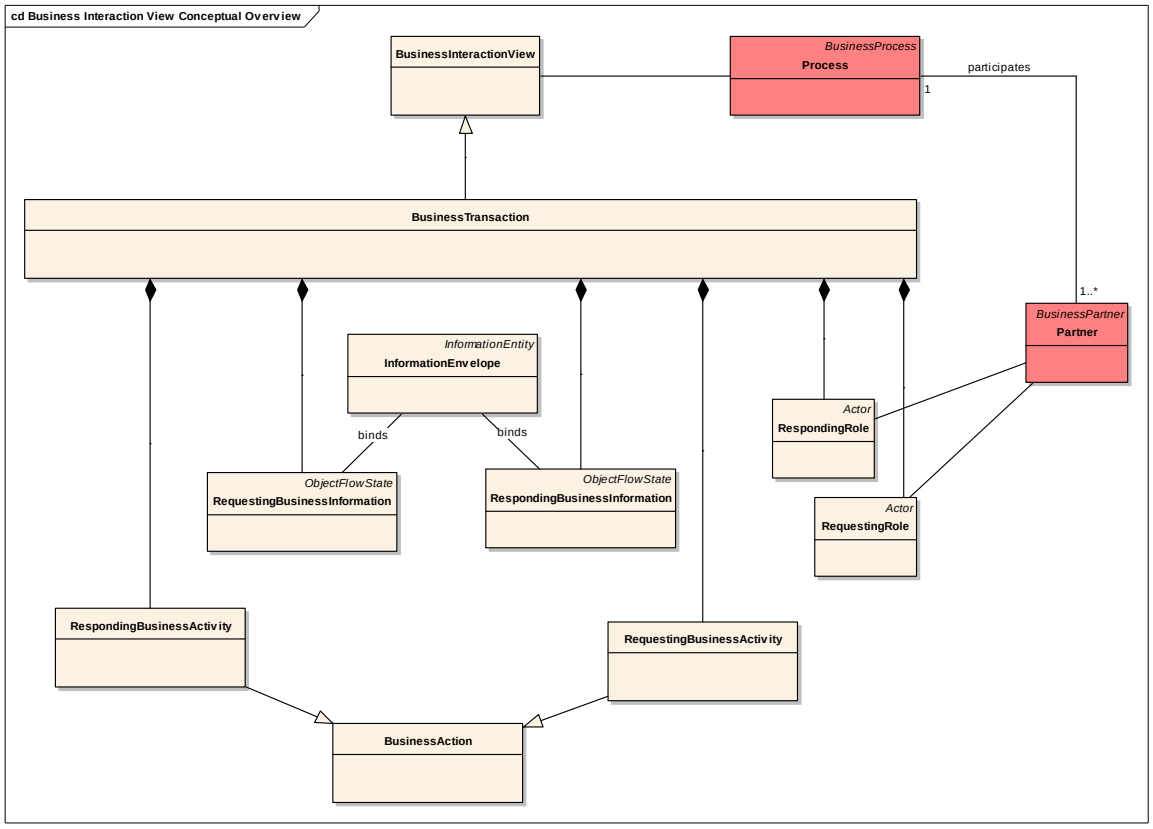

Figure 34 - BusinessInteractionView (BTV) - Conceptual Overview

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

BusinessInteractionBehavior is an abstract concept. In a future version there may exist different approaches to describe the choreography and information exchanges in a business interaction. In this version, the only valid specialization of a *BusinessInteractionBehavior* is the *BusinessTransaction*. A business transaction is an

atomic business process between two partners, which involves sending business information from one partner to the other and an optional reply. The business transaction is composed of two partitions - one for each partner. Hence, a *BusinessTransaction* is composed of exactly two instances of *BusinessTransactionPartition*. Each *BusinessTransactionPartition* relates to one *Partner*. A *Partner* might be assigned to multiple instances of *BusinessTransactionPartition* in different business transactions – however to only one *BusinessTransactionPartition* within any single business transaction. This means, that the two partitions of a business transaction must be assigned to different partners.

Within a business transaction each partner performs exactly one business action – the requesting partner performs a requesting business activity and the responding partner performs a responding business activity. Each business action – no matter whether requesting or responding business activity – is assigned to a partition, and each partition comprises exactly one business action. Note that each partner might perform multiple business actions, but only in different business transactions. It follows that a *BusinessTransaction* is composed of exactly one *RequestingBusinessActivity* and exactly one *RespondingBusinessActivity*. Both *RequestingBusinessActivity* and *RespondingBusinessActivity* are specializations of *BusinessAction*. A *BusinessAction* is assigned to one *BusinessTransactionPartition*, and a *BusinessTransactionPartition* comprises one *BusinessAction*.

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

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

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

5.3.1.2. Stereotype and Tag Definitions (normative)

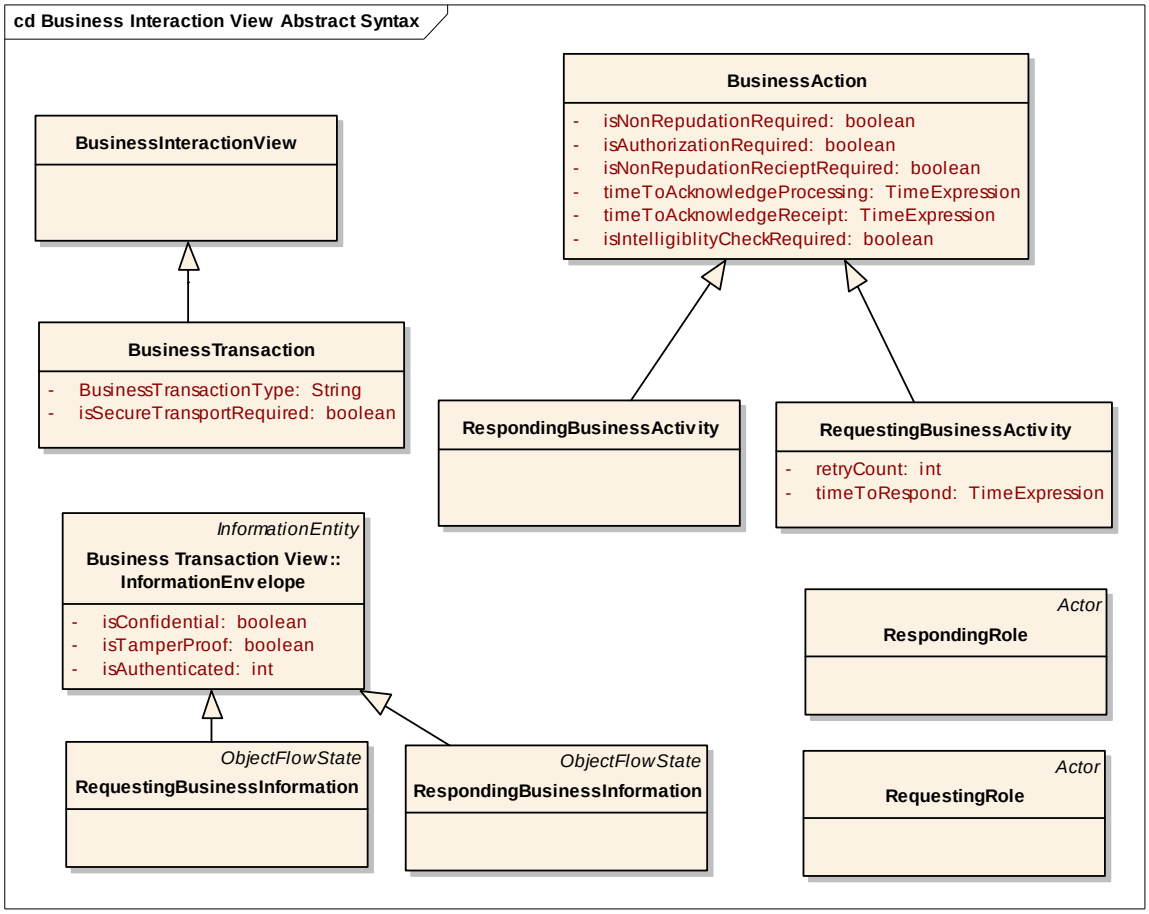

Figure 35 - BusinessInteractionView (BTV) - Abstract Syntax

Stereotype	BusinessInteractionView
Base Class	Class
Parent	N/A
Description	A business interaction is a persistent representation of a synchronization of business states between partners. It is a unit of work that allows roll-back.
Tag Definition	**No tagged values**

Stereotype	RespondingRole
Base Class	Partition
Parent	N/A
Description	A business transaction partition is used to define an area of responsibility for the responder. A partner is appointed to the partition of a business responding role. This *Partner* takes on the responsibility for the business action that is allocated within that area.
Tag Definition	**No tagged values**

Stereotype	RequestingRole
Base Class	Partition
Parent	N/A
Description	A business transaction partition is used to define an area of responsibility for the requestor. A partner is appointed to the partition of a business requesting role. This *Partner* takes on the responsibility for the business action that is allocated within that area.
Tag Definition	**No tagged values**

Stereotype	BusinessAction (abstract)
Base Class	ActivityGraph
Parent	**BusinessInteractionBehavior**
Description	The business action is executed by a partner during a business transaction. Business action is an abstract stereotype. This means a business action is either a requesting business activity or a responding business activity.
Tag Definition	**isAuthorizationRequired** Type: Boolean Multiplicity: 1 Description: If a Partner needs authorization to request a business action or to respond to a business action then the sending Partner must sign the business document exchanged and the receiving Partner must validate this business control and approve the authorizer. A responding partner must signal an authorization exception if the sending Partner is not authorized to perform the business activity. A sending partner must send notification of failed authorization if a responding partner 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 an information envelope to send a signed receipt. The *isNonRepudiationOfReceiptRequired* tag indicates that an involved party must not be able to repudiate the execution of sending the signed receipt. **timeToAcknowledgeReceipt** Type: TimeExpression Multiplicity: 1

	Description	Both partners may agree to mutually verify receipt of business information within 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 partner as well as the responding partner – it depends on whether requesting or responding business information is acknowledged. Similarly, the affirmant may be the requesting partner as well as the responding partner – 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). 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 business information might be the requesting partner as well as the responding partner – depending whether a requesting or responding business information is acknowledged. Also the affirmant may be one of the two partners. 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). 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 is*IntelligibleCheckRequired* 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.

Inherited tagged values: **None**

Stereotype	RequestingBusinessActivity
Base Class	ActionState
Parent	**BusinessAction**
Description	A requesting business activity is a business action that is performed by a partner requesting business service from another partner.
Tag Definition	timeToRespond Type: TimeExpression Multiplicity: 1 Description: Both partners may agree in case of a two-way business transaction that the responding partner must return the responding information business information within a specific duration. A responding partner must exit the transaction if it is not able to return the responding business information within the agreed timeout period. A requesting partner must retry a business transaction if necessary or must send notification of failed business control (possibly revoking a contractual offer) if a responding partner 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 partner until the time a responding business information is "properly received" by the requesting partner in return. **retryCount** Type: Integer Multiplicity: 1 Description: The requesting partner must re-initiate the business transaction as 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. Inherited tagged values: • isAuthorizationRequired • isNonRepudiationRequired • isNonRepudiationReceiptRequired • timeToAcknowledgeReceipt • timeToAcknowledgeAcceptance • isIntelligibleCheckRequired

Stereotype	RespondingBusinessActivity
Base Class	ActionState

Parent	**BusinessAction**
Description	A responding business activity is a business action that is performed by an partner responding to another business partner Partner's request for business service
Tag Definition	Inherited tagged values: • isAuthorizationRequired • isNonRepudiationRequired • isNonRepudiationReceiptRequired • timeToAcknowledgeReceipt • timeToAcknowledgeAcceptance • isIntelligibleCheckRequired

Stereotype	InformationEnvelope
Base Class	InformationEntity
Parent	N/A
Description	An envelope that contains business information.
Tag Definition	**isConfidential** Type: Boolean Multiplicity: 1 Description: True if the contents of the *InformationEnvelope* should be treated as confidential. **isTamperProof** Type: Boolean Multiplicity: 1 Description: True if the contents of the *InformationEnvelope* must be made tamper proof. **isAuthenticated** Type: Boolean Multiplicity: 1 Description: True if the contents of the *InformationEnvelope* must be autheticateable.

Stereotype	RequestingInformationEnvelope
Base Class	ObjectFlowState
Parent	N/A
Description	A type of *InformationEnvelope*, the requesting information envelope is a container of business information that is sent from the requesting partner to the responding partner to indicate a state change in one or more business entities. This business state change might be irreversible in the case of a one-way business transaction or an interim state of a two-way business transaction. It is important to note that the term requesting information envelope does not mean that the business information refers to a request in a business sense. The term requesting information envelope indicates that the execution of a transaction is requested by the requesting Partner to the responding Partner – no matter whether this is an information distribution, a notification, a request, or the offer in a commercial transaction.
Tag Definition	Inheritied tagged values: • isConfidential • isTamperProof • isAuthenticated

Stereotype	RespondingInformationEnvelope
Base Class	ObjectFlowState
Parent	N/A
Description	A type of InformationEnvelope, the responding information envelope is a container of business information that is sent in case of a two-way business transaction from the responding partner to the requesting partner in order to set one or more business entities in a final state (which were in an interim state before).
Tag Definition	Inheritied tagged values: • isConfidential • isTamperProof • isAuthenticated

Stereotype	BusinessTransaction	
Base Class	Model	
Parent	**BusinessInteractionView**	
Description	The type of interaction view associated with a transaction use case *Process*.	
Tag Definition	BusinessTransactionType	
	Type	String
	Multiplicity	1
	Description	The transaction type pattern that the transaction is based on: • Information Distribution • Notification • Request/Confirm • Request/Response • Commercial Transaction • Query/Response
	isSecureTransportRequired	
	Type	Boolean
	Multiplicity	1
	Description	True if the *InformationEnvelope* being transmitted in this transaction must be transmitted on a secure line.

5.3.1.3. Constraints (normative)

A *BusinessInteractionView* package MUST contain exactly one *BusinessTransaction* and no other elements

```
package Model_Management
context Package

inv BIVcontainsExactlyOneBT:
    self.isBusinessInteractionView() implies
    self.contents->one(isBusinessTransaction())
    and self.contents->size()=1
```

A *BusinessTransaction* MUST be connected with exactly one *BusinessInteractionBehavior* via a dependency with the stereotype *mapsTo*

```
package Behavioral_Elements::Activity_Graphs
context ActivityGraph

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

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

```
package Behavioral_Elements::Activity_Graphs
context ActivityGraph

inv BusinessTransactionHasExactlyTwoBTPartitions:
    self.isBusinessTransaction() implies
    self.oclAsType(ActivityGraph).partition->size() = 2
    and self.oclAsType(ActivityGraph).partition->forAll(part |
    part.isUMMTransactionPartition()
    and (part.contents->one(isRequestingBusinessActivity()) xor
    part.contents
    ->one(isRespondingBusinessActivity())))
    and self.oclAsType(ActivityGraph).partition->collect(part |
    part.contents)->one(isRequestingBusinessActivity())
    and self.oclAsType(ActivityGraph).partition->collect(part |
    part.contents)->one(isRespondingBusinessActivity())
```

A *BusinessTransactionPartition* MUST have a classifier, which MUST be one of the associated *Partners* of the corresponding *BusinessTransactionUseCase*

```
package Behavioral_Elements::Activity_Graphs
context Partition

inv BusinessTransactionPartitionClassifier:
    self.isUMMTransactionPartition() implies
    self.classifierPartner.base->size()=1 and
    self.activityGraph.clientDependency->
    collect(s | s.supplier)->collect(a |
    a.oclAsType(UseCase).associations)->
    collect(allConnections)
    ->select(isPartner())->one(x | x = (self.classifierPartner.base-
    >
    asSequence->first()))
```

The partition of the requesting *Partner* must contain exactly one *RequestingBusinessActivity*, one *RequestingInformationEnveleope* and one *InitialState*. Furthermore there MUST be at least two *FinalStates* in this *BusinessTransactionPartition*

```
package Behavioral_Elements::Activity_Graphs
context Partition

inv ContentsOfRequestingPartition:
    self.isUMMTransactionPartition() implies
    self.contents->one(isRequestingBusinessActivity()) implies
    self.contents->forAll(isRequestingBusinessActivity()
    or isRequestingInformationEnvelope()
    or isInitialState()
    or isFinalState()
    or isTransition()
    )
    and
    self.contents->one(isRequestingInformationEnvelope()) and
    self.contents->select(isFinalState())->size()>1 and
       self.contents->one(isInitialState())
```

The partition of the responding *Partner* MUST exactly contain one *RespondingBusinessActivity*. Furthermore if the transaction is a two way business transaction, then the partition must contain a *RespondingInformationEnvelope* as well. If the transaction is a one way business transaction, then the responder partition must not contain a *RespondingInformationEnvelope*.

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

Exactly one *Transition* MUST lead from the *InitialState* to the *RequestingBusinessActivity*

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

Exactly one *Transition* MUST lead from a *RequestingBusinessActivity* to the *equestingInformationEnvelope*

```
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrRequestingBusinessActivity2RequInfEnvelope:
    self.isUMMTransactionPartition() implies
    self.contents->one(isRequestingBusinessActivity()) implies
```

```
    self.contents->select(isRequestingBusinessActivity())->
    forAll(oclAsType(ActionState).outgoing->size()=1 and
    oclAsType(ActionState).outgoing->asSequence()
    ->first().target.isRequestingInformationEnvelope())
```

Exactly one *Transition* MUST lead from the *RequestingInformationEnvelope* to the *RespondingBusinessActivity*

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

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

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

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

```
package Behavioral_Elements::Activity_Graphs
context Partition

inv TrRespondingInformationEnvelope2RequestingBusinessActivity:
    self.activityGraph.isTwoWayTransaction() implies
    self.contents->one(isRespondingBusinessActivity()) implies
    self.contents->select(isRespondingInformationEnvelope())->
    forAll(oclAsType(ObjectFlowState).outgoing->size()=1 and
    oclAsType(ObjectFlowState).outgoing->asSequence
    ->first().target.isRequestingBusinessActivity())
```

There MAY be a *Transition* from *RespondingBusinessActivity* to *RequestingBusinessActivity* (only for one-way business transactions)

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

One *Transition* MUST lead from the *RequestingBusinessActivity* to each *FinalState*.

```
package Behavioral_Elements::Activity_Graphs
context Partition

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

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

```
package Behavioral_Elements::Activity_Graphs
context ObjectFlowState

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

5.3.1.4. Example (informative)

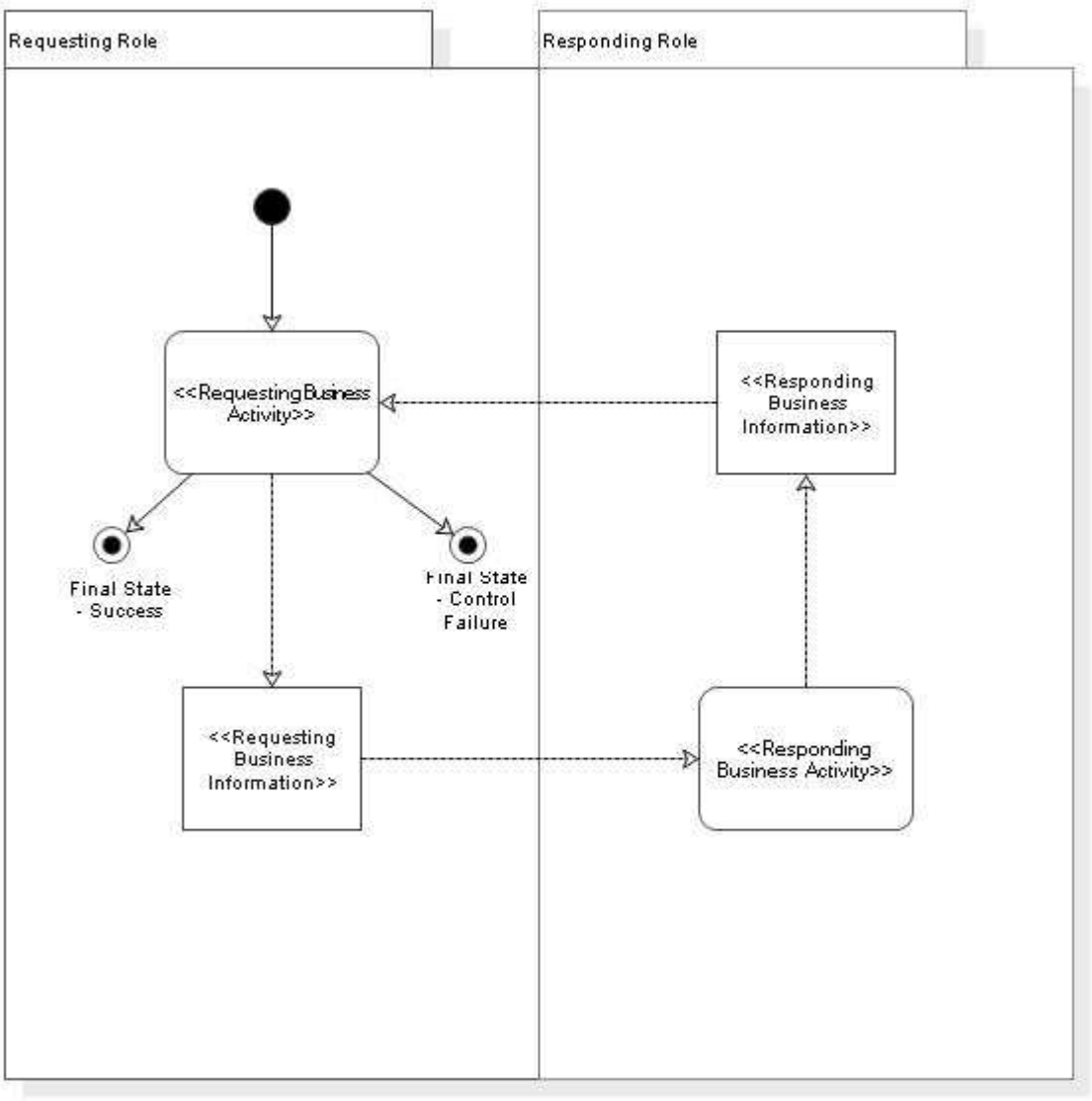

Figure 36 - BusinessInteractionView (BusinessTransactionView)

5.3.2. Business Information View

5.3.2.1. Conceptual Overview (informative)

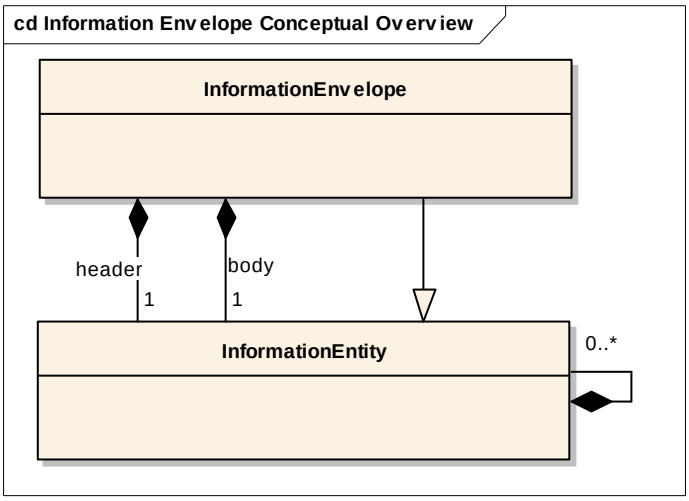

Figure 37 - BusinessInformationView (BTV) - Conceptual Overview

A *BusinessInformationView* is a container of artifacts that describe the information exchanged in an interaction. As mentioned earilier, a *RequestingInformationEnvelope* and *RespondingInformationEnvelope* are of type *InformationEnvelope*. An information envelope serves as a cover for all the information exchanged between the *RequestingBusinessActivity* and the *RespondingBusinessActivity* or vice versa. The information included in the envelope is structure by classes that are stereotyped as *InformationEntity*. Information entities might be recursively nested. Thus there is a unary composition hierarchy added to *InformationEntity*. An information envelope consists of a header and one or more bodies. Both header and body are modeled as information entities. It follows, that an *InformationEnvelope* is composed of exactly one *InformationEntity* with the PartnerName *header* and of one or more *InformationEntities* with the PartnerName *body*. An *InformationEnvelope* is a specialization of an *InformationEntity* that fulfills all the rules mentioned for the information envelope as well.

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

5.3.2.2. Stereotype and Tag Definitions (normative)

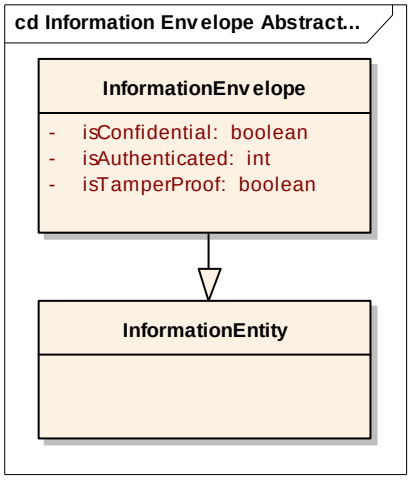

Figure 38 - BusinessInformationView (BTV) - Abstract Syntax

Stereotype	InformationEntity	
Base Class	Class	
Parent	**N/A**	
Description	An information entity realizes structured business information that is exchanged by Partners performing activities in a business transaction. Information entities include or reference other information entities through associations	
Tag Definition	isConfidential	
	Type	Boolean
	Multiplicity	1
	Description	If the flag is set, the information entity is encrypted so that unauthorized parties cannot view the information
	isTamperProof	
	Type	Boolean
	Multiplicity	1
	Description	If the flag is set, the information entity has an encrypted message digest that can be used to check if the message has been tampered with. This requires a digital signature (sender's digital certificate and encrypted message digest) associated with the document entity
	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.
	Inherited tagged values: **None**	

Stereotype	InformationEnvelope
Base Class	Class
Parent	**InformationEntity**
Description	An information envelope is a container for information entities. The information envelope is a specialization of the information entity. It extends the concept of the information entity by the fact that it includes exactly one information entity that takes on the Partner of a header and at least one information entity that takes on the Partner of a body. Furthermore, the information exchanged in a business transaction i.e. requesting business information and responding business information is always of type information envelope.
Tag Definition	Inherited tagged values: • isConfidential • isTamperProof • isAuthenticated

5.3.2.3. Constraints (normative)

An *InformationEnvelope* MUST have one association to an *InformationEntity* with *Partner* name header

```
package Foundation::Core
context Class

inv InformationEnvelopeHasHeader:
    self.isInformationEnvelope() implies
    self.associations->forAll(a | a.connection->size() = 2 and
    a.allConnections->one(participant.isInformationEntity() and
    AssociationEndPartner.name = 'header'))
```

An *InformationEnvelope* MUST have at least one associated *InformationEntity* with *Partner* name body

```
package Foundation::Core
context Class

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

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

```
package Foundation::Core
context Class

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

5.3.2.4. OCL Contraints used in all packages of the Business Transaction View (BTV) (normative)

OCL-Methods

```
package Foundation::Core
context ModelElement

--Predefined method whichs evaluates, if the given Modelelement
--has a stereotype equal to the passed name
def :
let hasStereotype (st : String) : Boolean =
self.stereotype->select(self.name = st)->notEmpty()

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

--Predefined method whichs evaluates, if the given element
--is a subtype of 'BusinessInteractionBehavior'
def :
let isBusinessInteractionBehavior() : Boolean =
self.oclIsKindOf(ActivityGraph) and
self.hasStereotype('BusinessTransaction')

--Predefined method whichs evaluates, if the given element
--is a 'BusinessChoreography'
def :
let isBusinessChoreography() : Boolean =
self.oclIsKindOf(Class) and
self.hasStereotype('BusinessChoreography')

--Predefined method which evaluates, if the
--ActivityGraph is a BusinessCollaborationProtocol
def:
let isBusinessCollaborationProtocol() : Boolean =
self.oclIsKindOf(ActivityGraph) and
self.hasStereotype('BusinessCollaborationProtocol')

--Predefined method which evaluates, if the
--ActivityGraph is a subtype of
--BusinessChoreographyBehavior
def:
let isBusinessChoreographyBehavior() : Boolean =
self.oclIsKindOf(ActivityGraph) and
self.hasStereotype('BusinessCollaborationProtocol')

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

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

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

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

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

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

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

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

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

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

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

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

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

let isPseudoStateOrFinalStateOrTransition() : Boolean =
isInitialState() or
isFinalState()

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

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

-- Returns true if the stereotype of the given element is
--'BusinessInteractionView'
-- and if the type of the element is Package
```

```
def :
let isBusinessInteractionView() : Boolean =
self.hasStereotype('BusinessInteractionView') and
self.oclIsKindOf(Package)

-- Returns true if the stereotype of the given element is
'InformationEntitiy'
-- and if the type of the element is Class
def :
let isInformationEntity() : Boolean =
self.hasStereotype('InformationEntity') and
self.oclIsKindOf(Class)

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

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

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

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

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

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

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

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

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


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

5.3.2.5. Example (informative)

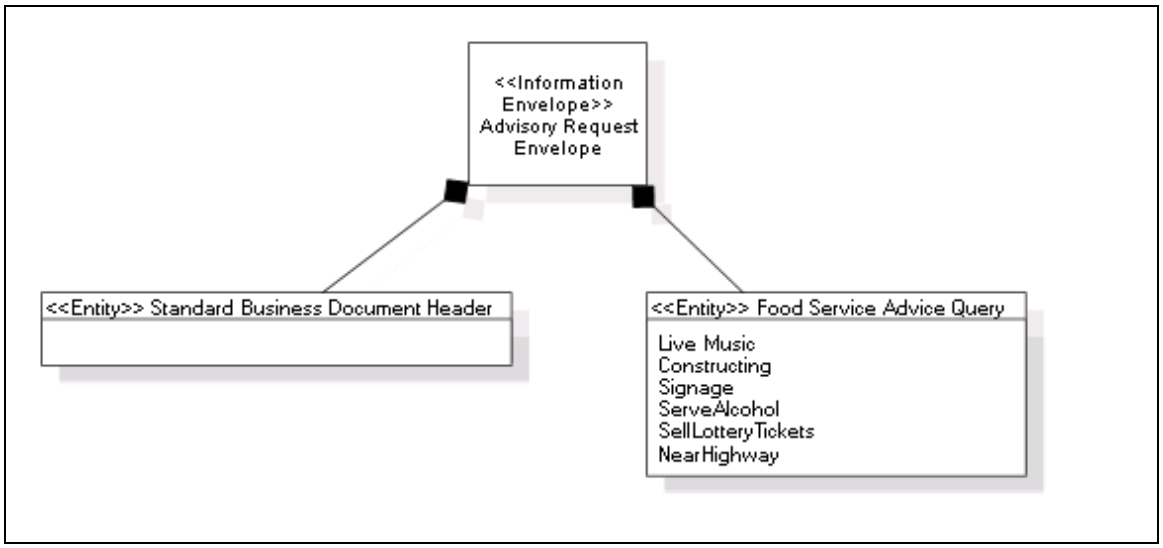

Figure 39 - BusinessInformationView (BusinessTransactionView)

6. Copyright / Permission to Reproduce

This document is covered by the provisions of the *Copyright Act*, by Canadian laws, policies, regulations and international agreements. Such provisions serve to identify the information source and, in specific instances, to prohibit reproduction of materials without written permission.

6.1. Non-commercial Reproduction

Information in this document is intended to be readily available for personal and public non-commercial use and may be reproduced, in part or in whole and by any means, without charge or further permission from the Treasury Board of Canada Secretariat. We ask only that:

- Users exercise due diligence in ensuring the accuracy of the materials reproduced;
- The Treasury Board of Canada Secretariat be identified as the source department; and,
- The reproduction is not represented as an official version of the materials reproduced, nor as having been made, in affiliation with or with the endorsement of the Treasury Board of Canada Secretariat.

6.2. Reproduction of Government Symbols

The official symbols of the Government of Canada, including the 'Canada' wordmark, may not be reproduced, whether for commercial or non-commercial purposes, without written authorization. Request for authorization from the Treasury Board Secretariat may be addressed to:

Federal Identity Program,
Treasury Board of Canada Secretariat,
300 Laurier Avenue West,
Ottawa, Canada K1A 0R5
Telephone: 613-957-2533,
Facsimile: 613-946-5187,
E-mail: information@fip-pcim.gc.ca

6.3. Commercial Reproduction