PUC

ISSN 0103-9741

Monografias em Ciência da Computação

n° 06/18

CM-OPL: Configuration Management Ontology Pattern Language Specification

Ana Carolina Brito de Almeida

Maria Luiza Machado Campos

Daniel Schwabe

Sérgio Lifschitz

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 06/18 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena July, 2018

CM-OPL: Configuration Management Ontology Pattern Language Specification

Ana Carolina Brito de Almeida[1], Maria Luiza Machado Campos[2]
Daniel Schwabe, Sérgio Lifschitz

[1]Department of Computer Science – State University of Rio de Janeiro (UERJ)
[2]Department of Computer Science - Federal University of Rio de Janeiro (UFRJ)

ana.almeida@ime.uerj.br, mluiza@ppgi.ufrj.br, {dschwabe, sergio}@inf.puc-rio.br

Abstract. This document presents the Configuration Management Ontology Pattern Language (CM-OPL). It is the first version of the CM-OPL, represented by using OPL-ML (Ontology Pattern Language Modeling Language). Therefore, we used a structural model to represent the CM-OPL patterns and structural relationships between them. Also, we present a general process model to provide a general view of the CM-OPL process, and detailed process models expand the process general view.

Keywords: Ontology; Pattern; OPL; Configuration Management.

Resumo. Este documento apresenta a linguagem de padrão de ontologia para gerência de configuração (CM-OPL). É a primeira versão da CM-OPL, representada pelo uso da OPL-ML (Linguagem de Modelagem para Linguagem de padrão de ontologia). Além disso, utilizamos um modelo estrutural para representar os padrões da CM-OPL e os relacionamentos entre eles. Adicionalmente, nós apresentamos o modelo do processo geral para viabilizar uma visão geral do processo CM-OPL e detalhamos os modelos do processo, expandindo a visão geral do processo.

Palavras-chave: Ontologia; Padrão; OPL; Gerência de configuração.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

We have written this document based on the S-OPL specification written by NEMO group [Quirino et al, 2018]. An Ontology Pattern Language (OPL) is a network of interconnected Domain-Related Ontology Patterns (DROPs) that provides holistic support for solving ontology development problems for a specific domain [Ruy et al, 2017]. We used the OPL-ML [Quirino et al, 2017] to represent the CM-OPL.

The Configuration Management Ontology Pattern Language (CM-OPL) is an OPL that addresses the core conceptualization about the configuration management problem. We have extracted CM-OPL patterns from the Configuration Management Task Ontology (CMTO) used for semantic integration [Calhau et al, 2012][Calhau, 2011]. We have chosen this ontology because it is generic and well-founded using UFO-A [Guizzardi, 2005]. The CMTO focuses on the three main activities of the Configuration Management process: Configuration Identification, Version Control, and Change Control. Thus, we may organize the patterns of CM-OPL in these three groups: *Configuration Identification, Version Control, and Change Control*.

We briefly present the patterns that compose CM-OPL in Section 2. Then, we give the CM-OPL structural model in Section 3, explaining the CM-OPL process model in Section 4. Finally, in Section 5, each CM-OPL pattern is fully described.

2 CM-OPL Domain-Related Ontology Patterns

We organize CM-OPL into three groups, namely: (i) *Configuration Identification*, (ii) *Version Control*, and (iii) *Change Control*.

According to CMTO (Configuration Management Task Ontology) [Calhau et al, 2012], the Configuration Identification refers to identifying product items to be controlled (Configuration Items - CIs), defining criteria for selecting CIs and their versions, establishing standards for numbering, and defining tools and techniques to be used to control the items. Item can be any element that composes a product and can have its configuration managed. The item can be a tool or an artifact because both are subject to change and we could manage their changes. The Configuration Item is an element from the product that we may configure and manage. This is an item that has a configuration selection done by a configuration manager.

We describe in Table 1 the intent of the patterns of the Configuration Identification group.

Table 1 – Patterns of the *Configuration Identification* group

Id	Name	Intent
P-Manager	Person Configuration Manager	Represents persons as configuration managers.
A-Manager	Agent Configuration Manager	Represents agents or machines as configuration managers.
PA-Manager	Person / Agent Configuration Manager	Represents persons and agents or machines as configuration managers.
ISelection	Item Selection	Allows selecting the configuration that is necessary, which items are managed and who is responsible for it. Represents an object that formalizes which items of a product/item that are managed.
IType	Item Type	Defines the item type that is the subject of the configuration. Any item which can have change.
IArtifact	Item Artifact	Represents an item that is the subject of the configuration as an artifact.
ITool	Item Tool	Represents an item that is the subject of the configuration as a tool.
IAtomic	Item Atomic	Represents an atomic configuration item of the product/item which could be configured and managed.
IComposite	Item Composite	Represents a composite configuration item of the product/item which could be configured and managed.
IBaseline	Item Baseline	Defines a configuration snapshot to the configured item at any given time.

Version control combines procedures and tools to manage different versions of the CIs. The item evolves over time. So, the CI has one or more versions which represent the evolution of the item. The version is related to the item and can be atomic or composite. A composite CI has others versions, and they are called configuration. The version of an atomic CI is an atomic version. When a configuration has a markup, it practices the role of baseline done by Configuration Manager. Additionally, it has control of the version before and after the modification. Before the modification, the CI needs to have the version checked out. Then, s/he does the modification and checks-in the modified version.

We describe in Table 2 the intent of the patterns of the Version Control group.

Table 2 - Patterns of the *Version Control* group

Id	Name	Intent
IVersion	Item Version	Represents the version of the item that has configuration changed.
IVariant	Item Variant	Represents the variation of the item – parallel versions.
IRevision	Item Revision	Represents the revision of the item – when versions overwrite others versions.
IVAtomic	Item Version Atomic	Represents an atomic version of the CI.
IVConfigura-tion	Item Version Configura-tion	Represents the configuration with a composite CI.
IRepository	Item Repository	Represents the location that holds all ver-sions of CI, including the project, reposi-tory and its ramifications composed of some versions of CI.
ICheckout	Item Check-out	Represents the last version of the item that will be changed.
ICheckin	Item Check-in	Represents the register of the version of the modified item.

Change Control deals with change management during the product life cycle. The Requester requires a change of an item of the product based on a version. This version is submitted to the change. The change can be a problem to solve or customization of the item. An Evaluator evaluates the possibility to implement the change and decides if the change can be implemented or not. When the request is approved, the Executor can execute the change of the version checked-out and submitted to the validation (check-in). The Verifier validates the changes made, verifying if it is in accordance with what was specified.

We describe in Table 3 the intent of the patterns of the Version Control group.

Table 3 - Patterns of the *Change Control* group

Id	Name	Intent
P-Requester	Person Requester	Represents persons as requesters.
A-Requester	Agent Requester	Represents agents/machines as requesters.
PA-Requester	Person / Agent Requester	Represents persons and agents or machines as requesters.
IRequestV	Item Version Request	Represents the change request mediated by a Requester and a version, when submitted for change.
IRequest	Configuration Item Request	Represents the change request mediated by a Requester and a configuration item, which submitted for change.
P-Evaluator	Person Evaluator	Represents persons as evaluators.
A-Evaluator	Agent Evaluator	Represents agents/machines as evaluators.
PA-Evaluator	Person / Agent Evaluator	Represents persons and agents or machines as evaluators.
IEvChRequest	Configuration Item Evaluation Change Request	Represents the evaluation if the item can have the CI applied.
P-Executor	Person Executor	Represents persons as executors.
A-Executor	Agent Executor	Represents agents/machines as executors.
PA-Executor	Person / Agent Executor	Represents persons and agents or machines as executors.
IExecRequestV	Item Version Execution Request	Represents the execution of the change in a version of the configuration item.
IExecRequest	Configuration Item Execution Request	Represents the execution of the change in a configuration item.
P-Verifier	Person Verifier	Represents persons as verifiers.
A- Verifier	Agent Verifier	Represents agents/machines as verifiers.
PA- Verifier	Person / Agent Verifier	Represents persons and agents or machines as verifiers.
IVerRequest	Configuration Item Verification Request	Represents the verification of the item with the CI applied through a specification.

3 CM-OPL Structural Model

We present in Figure 1 the CM-OPL structural model. In the model, *patterns* are represented by rectangles with underlined labels. *Regions delimited by blue straight lines represent pattern group*s. *Rectangles delimit groups of variant pattern*s with red dotted edges. *Variant patterns* are patterns that solve the same problem but in different ways. Thus, from a set of variant patterns, when developing an ontology, only one can be used to solve the problem. Pattern dependency relations are represented by directed arrows, meaning that the source pattern (or pattern group) requires the target pattern to be applied first. Finally, dotted arrows are used to indicate that a pattern requires one of the patterns of a variant group. In the structural model, different colors are used to identify application actions patterns from different groups.

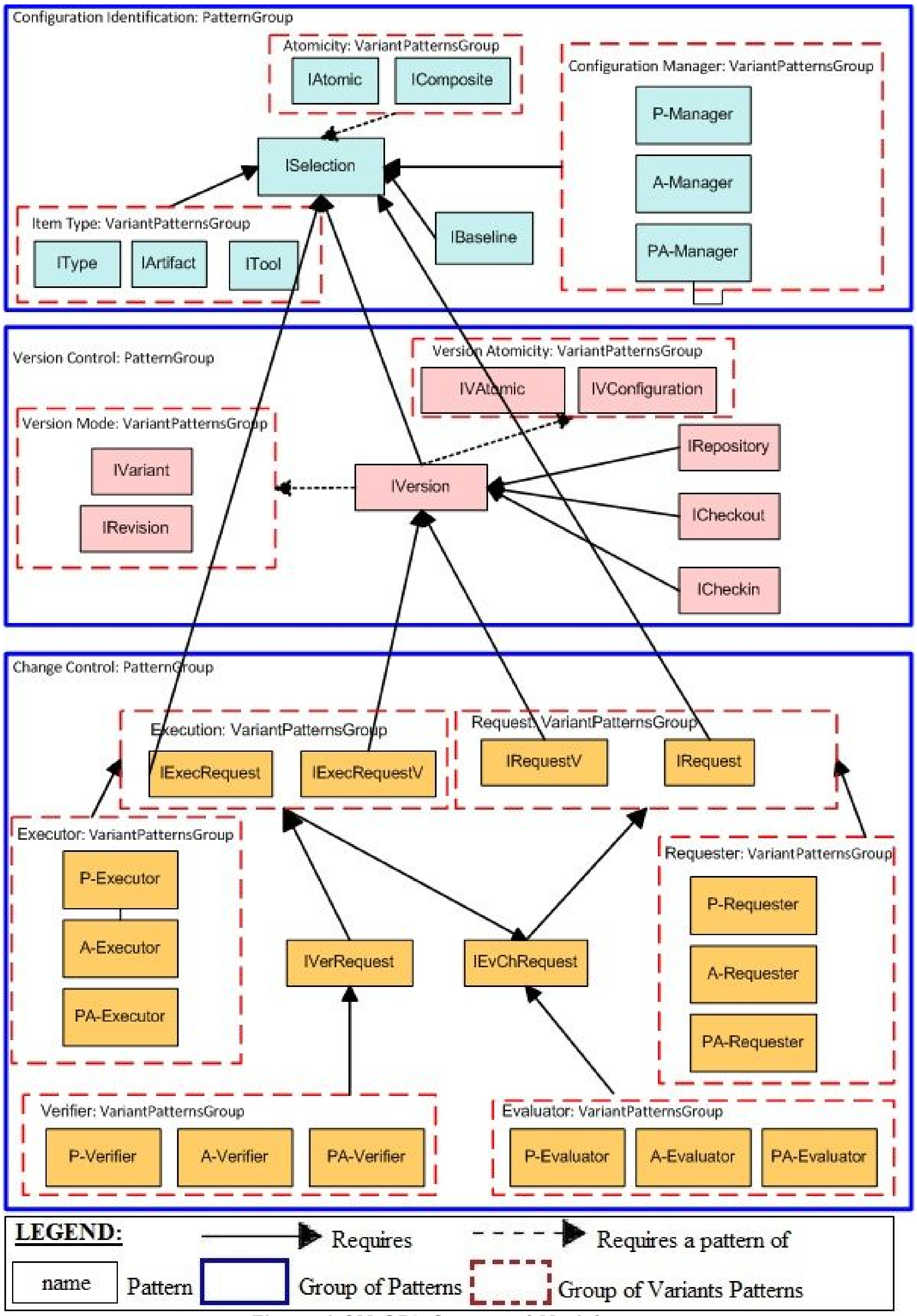
Configuration Identification: PatternGroup
Atomicity: VariantPatternsGroup
IAtomic
IComposite
Configuration Manager: VariantPatternsGroup
P-Manager
A-Manager
PA-Manager
ISelection
IBaseline
Item Type: VariantPatternsGroup
IType
IArtifact
ITool
Version Control: PatternGroup
Version Atomicity: VariantPatternsGroup
IVAtomic
IVConfiguration
Version Mode: VariantPatternsGroup
IVariant
IRevision
IVersion
IRepository
ICheckout
ICheckin
Change Control: PatternGroup
Execution: VariantPatternsGroup
IExecRequest
IExecRequestV
Request: VariantPatternsGroup
IRequestV
IRequest
Executor: VariantPatternsGroup
P-Executor
A-Executor
PA-Executor
IVerRequest
IEvChRequest
Requester: VariantPatternsGroup
P-Requester
A-Requester
PA-Requester
Verifier: VariantPatternsGroup
P-Verifier
A-Verifier
PA-Verifier
Evaluator: VariantPatternsGroup
P-Evaluator
A-Evaluator
PA-Evaluator
LEGEND:
Requires
Requires a pattern of
name
Pattern
Group of Patterns
Group of Variants Patterns

Figure 1 CM-OPL Structural Model

4 CM-OPL Process

Figure 2 provides a general view of the CM-OPL process. P*attern application action groups* are represented as black boxes, providing a more general view of CM-OPL. In this figure, pattern application action groups are represented by labeled rectangles with blue edges and with the symbol in the corner. A *pattern application action* refers to the application of a specific pattern. Initial nodes (solid circles) are used to represent entry points in the OPL, i.e., pattern application actions in the language that can be performed first, without performing other pattern application actions. *Decision nodes* (represented by diamonds) are used to represent alternative paths. Thus, if the ontology engineer decides to follow the decision node input path, then s/he must select one and only one of the decision node output paths. Control flows (arrowed lines) represent the sequences of paths that the ontology engineer can follow in the OPL. Endpoints (solid circle doubly circled) are used to indicate where the patterns application process can be finished. Like in the structural model, in the process models, different colors are used to identify application actions patterns from different groups.

We have extracted the patterns in CM-OPL from the CM Task Ontology, mentioned previously. The CM-OPL patterns are organized into three groups according to the process presented in [Calhau et al, 2012]: Configuration Identification, Version Control and Change Control and represented in Figure 2.

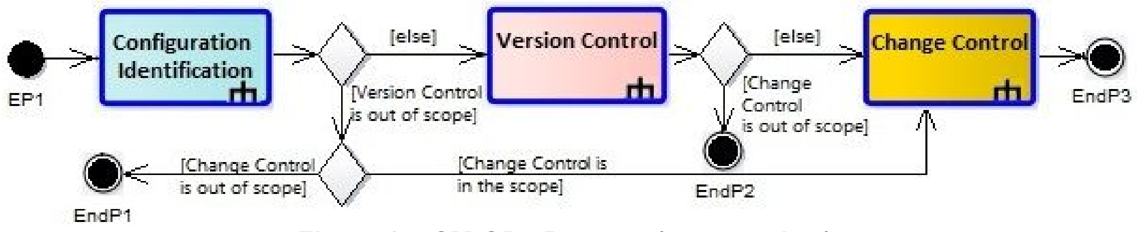
EP1
Configuration Identification
[else]
Version Control
[Version Control is out of scope]
[else]
Change Control
[Change Control is out of scope]
EndP3
EndP1
[Change Control is out of scope]
[Change Control is in the scope]
EndP2

Figure 2 – CM-OPL Process (general view)

Like in the structural model, in the process models (Figures 3-6), different colors are used to identify application actions patterns from different groups. Initial nodes (solid circles), pattern application action nodes (the labeled rounded rectangles), decision nodes (diamonds), control flows (arrowed lines) and end points (solid circle doubly circled) have the same representation of the structural model. Moreover, we group *variant pattern application actions* inside rectangles with red dotted edges.

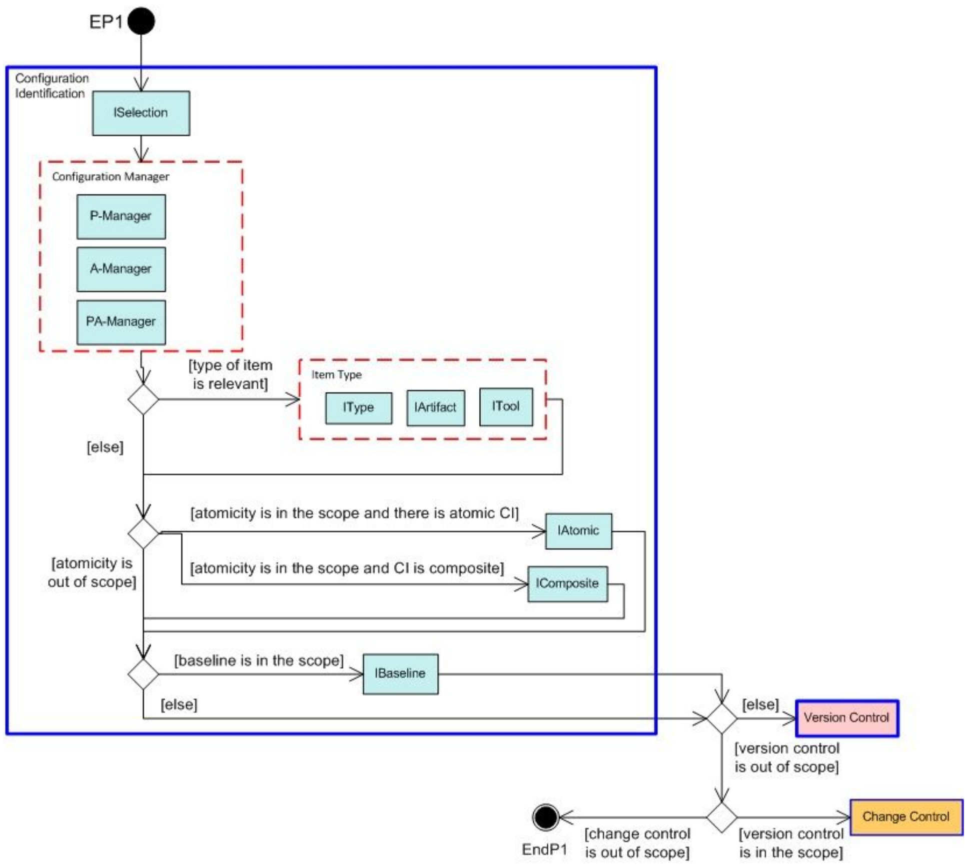
EP1
Configuration Identification
ISelection
Configuration Manager
P-Manager
A-Manager
PA-Manager
[type of item is relevant]
Item Type
IType
IArtifact
ITool
[else]
[atomicity is in the scope and there is atomic CI]
IAtomic
[atomicity is in the scope and CI is composite]
IComposite
[atomicity is out of scope]
[baseline is in the scope]
IBaseline
[else]
[else]
Version Control
[version control is out of scope]
EndP1
[change control is out of scope]
[version control is in the scope]
Change Control

Figure 3 – Detailed Process Model of the Configuration Identification Group

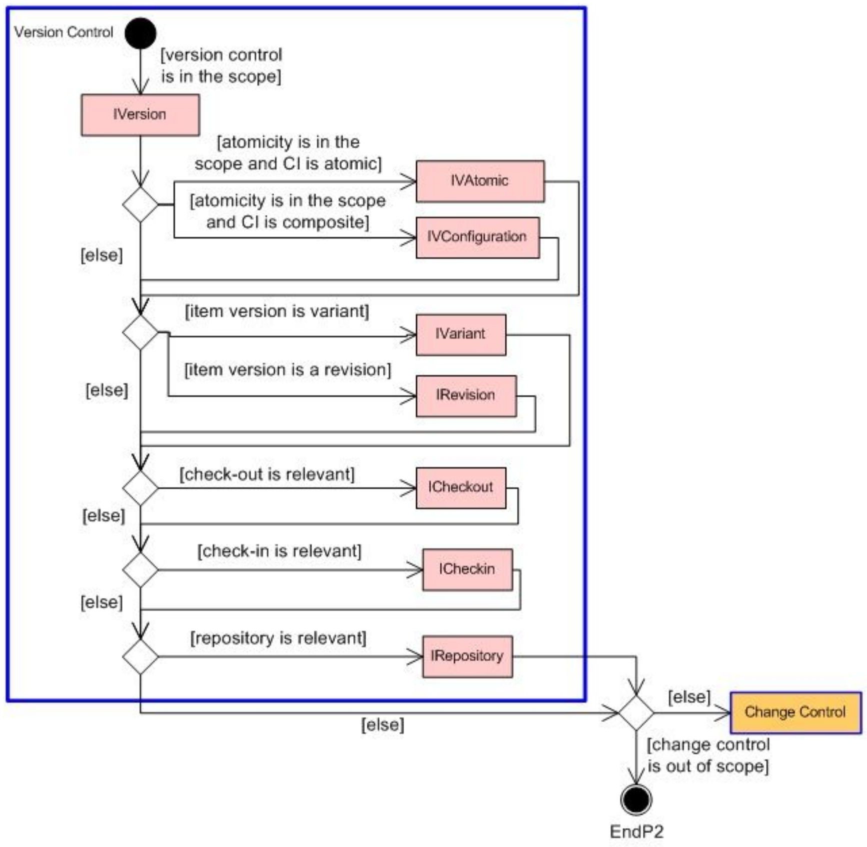

Figure 4 - Detailed Process Model of the Version Control Group

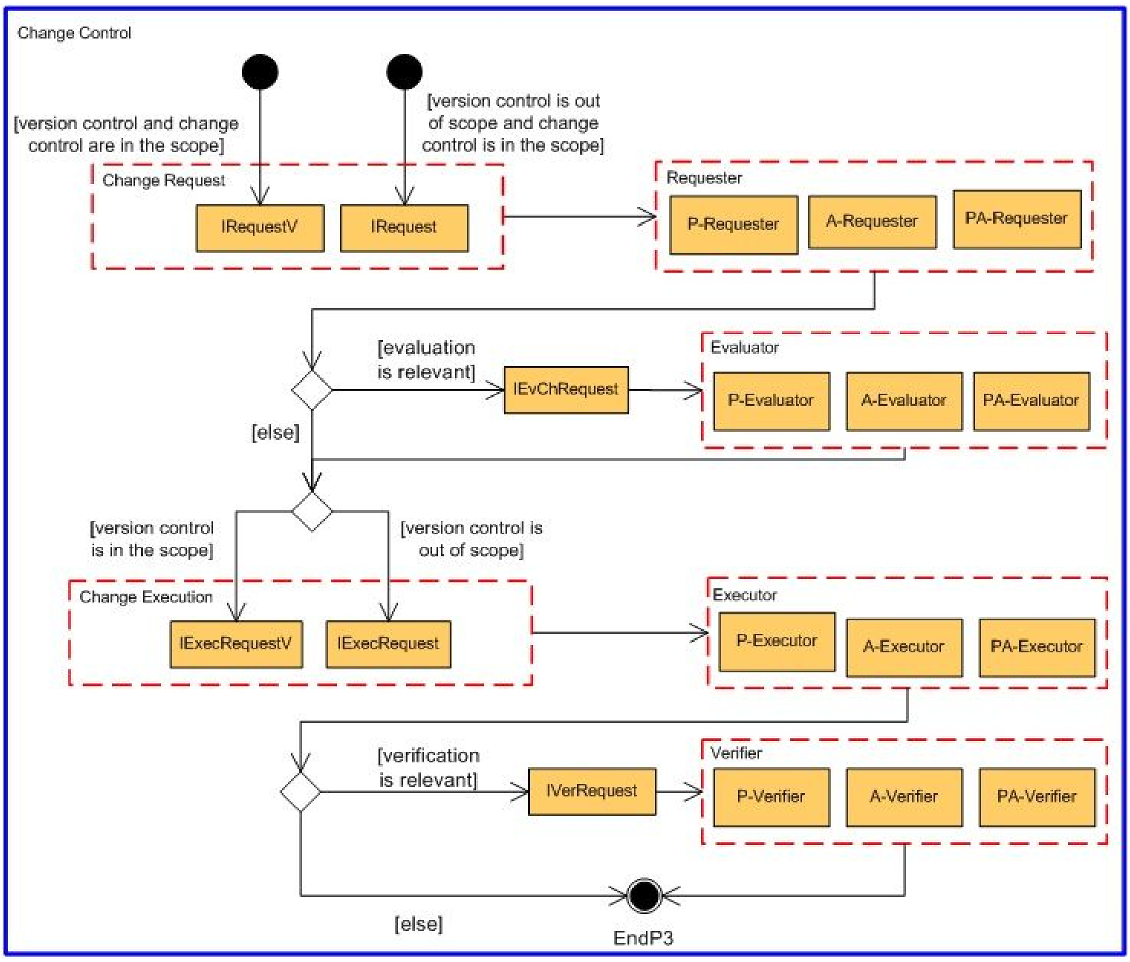

Figure 5 - Detailed Process Model of the Change Control Group

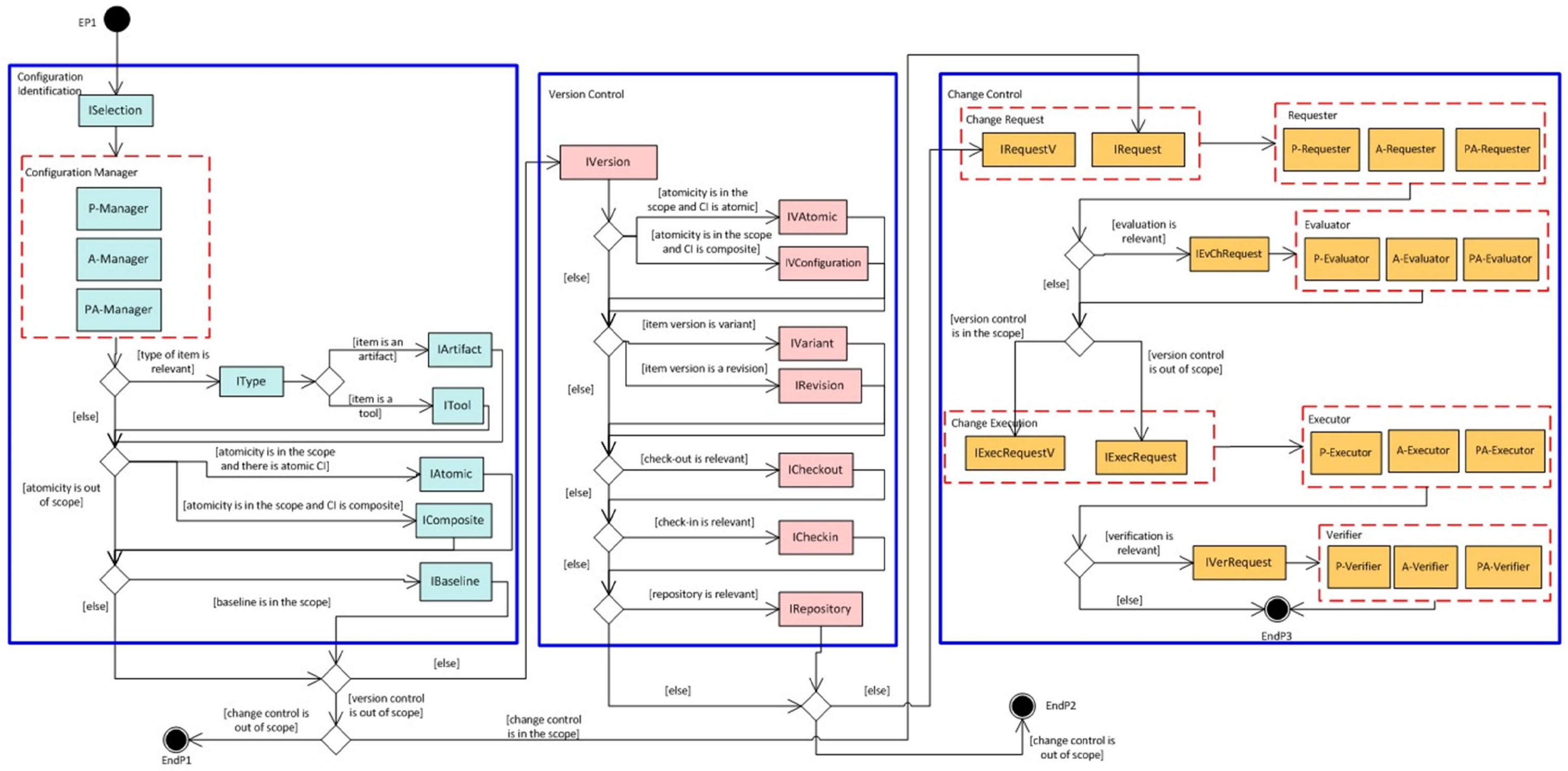

Figure 6 – CM-OPL Process (detailed view)

As Figure 6 shows, CM-OPL has only one entry point (EP1). The ontology engineer (OE) must start the new ontology by selecting the configuration that s/he needs to do (*ISelection*). Next, s/he decides who will manage the configuration. The OE has to select a pattern from the *Configuration Manager* group of variant patterns. Also, it is necessary to define the item type (*IType*), when relevant, that is the subject of the configuration being modeled (*IArtifact* or *ITool*). Then, s/he must select one of the patterns of the Atomicity variant group according to the atomicity of the item that will be configured (*IAtomic* or *IComposite*). The last pattern of this group addresses the baseline of the item (*IBaseline*). A baseline is a product configuration that was revised and designated to be a basis for future development [Calhau et al. 2012].

After, if the version control is in the scope, the patterns of this group should be used. If versioning is out of the scope, the OE needs to decide if there is change control or not. If change control is in the scope, the OE jumps to patterns of the change control group. On the other hand, the process ends.

If the OE wants models the version of the Item, s/he needs to apply the *IVersion* pattern. This version can be atomic (*IVAtomic*) or composite (*IVConfiguration*) that involves the old and new version at least. There is the mode of the version, that is, a variant (*IVariant*) or revision (*IRevision*) of the item. Also, the last version of the item registered can be checked out of a repository to the agent/person to change (*ICheckout*). After the modification, s/he can do the checkin to register the new version (*ICheckin*). All the versions are registered in the repository (*IRepository*).

After modeling the version control, if change control is in the scope, the *IRequestV* pattern is used. If the OE chooses not to use the version control group, instead of this, the *IRequest* pattern needs to be used. Both of patterns model a change request that is submitted by the Requester. If *IRequestV* is used, the Version mediates the change request. On the other hand, the Configuration Item mediates the change request. Regardless of the chosen pattern, the Requester must have its chosen pattern from the variant group (*Requester*).

Next, the OE decides about the relevance of the evaluation. If it is relevant, the Evaluator decides if the change should be implemented or not (*IEvCHRequest*). Thus, the Executor implements the modification modeling through the *IExecRequest* pattern (version control is out of scope) or *IExecRequestV* pattern (version control is in the scope). After implementing the change, validation occurs. The pattern corresponding to the last configuration step (*IVerRequest*), if it is relevant, presents the Verification relator mediating Verified Change and the Verifier. Finally, the process ends.

5 CM-OPL Patterns Descriptions

The description of CM-OPL patterns includes the following items:

- ✓ **Name**: provides the name of the pattern.
- ✓ **Intent**: describes the pattern purpose.
- ✓ **Rationale**: describes the rationale underlying the pattern. A short statement answering the following question: What is the pattern rationale?
- ✓ **Competency Questions**: describes the competency questions that the pattern aims to answer.
- ✓ **Conceptual Model**: depicts the OntoUML diagram representing the pattern elements.
- ✓ **Axiomatization**: presents the axioms related to the pattern conceptual model.
- ✓ **FOPs Support:** lists Foundational Ontology Pattern (FOPs) used, FOPs are reusable fragments derived from foundational ontologies [Falbo et al, 2013].
- ✓ **Term Definitions:** Definition of the class in the context of the conceptual model in the pattern.

5.1 Configuration Identification Group

ISelection – Item Selection

Name: Item Selection

Intent: Allows selecting the configuration that is necessary, which items are managed and who is responsible for it. Represents an object that formalizes which items of a product/item that are managed.

Rationale: A *Configuration Selection* mediates the relation between a *Configuration Manager* and a *Configuration Item*. *Configuration Selection* defines the selection on an item configuration. *Configuration Manager* is the role played by the persons, the agents or both when they become a *Configuration Manager*. The stereotype of the *Configuration Manager* class is given by the pattern selected from the *Configuration Manager* sub-group.

Competency Questions:

- ✓ *Who is the Configuration Manager that selects each configuration item?*

Conceptual Model:

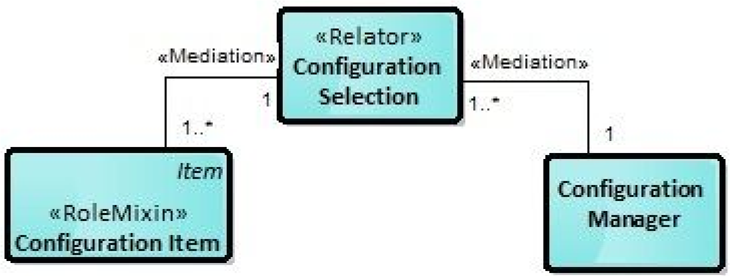

Figure 7 ISelection – Conceptual Model

Note: The stereotype of the *Configuration Manager* class is given by the pattern selected from the Configuration Manager sub-group. For instance, if the P-Manager pattern is selected, then *Configuration Manager* is a <<role>>; if the PA-Manager pattern is selected, then *Configuration Manager* is a <<rolemixin>>. Due to this fact, the *Configuration Manager* class is not stereotyped in the current pattern.

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1.

Term Definitions:

Configuration Item	An item of product that has a configuration which can be managed.
Configuration Selection	Formalizes which items of a product that are managed. Registers the act of selecting items to be managed and transformed them into *Configuration Items*.
Configuration Manager	The role played by a *Person, an Agent* or both when they manage a configuration of an item.

P-Manager – Person Configuration Manager

Name: Person Configuration Manager

Intent: Represents persons as configuration managers.

Rationale: *Persons* can act as (play the role of) *Configuration Managers*, i. e., the ones responsible for the configuration management.

Competency Questions:
- ✓ Which are the types of configuration managers?
- ✓ *What are the possible types (person, agent or both) of configuration manager?*

Conceptual Model:

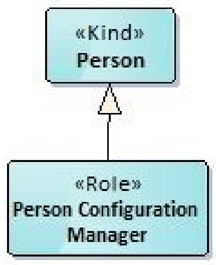

Figure 8 P-Manager – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Person	An individual human being.
Person Configuration Manager	The role played by a *Person* when s/he manages a configuration of an item.

A-Manager - Agent Configuration Manager

Name: Agent Configuration Manager

Intent: Represents agents or machines as configuration managers.

Rationale: Software *Agents* or machines can act as (play the role of) *Configuration Managers*, i. e., the ones responsible for the configuration management (automatic).

Competency Questions:

- ✓ *What are the possible types (person, agent or both) of configuration manager?*

Conceptual Model:

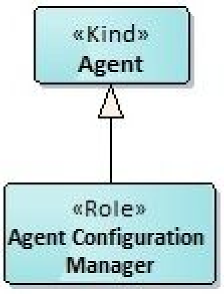

Figure 9 A-Manager – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Configuration Manager	The role played by an *Agent* when it manages a configuration of an item.

PA-Manager – Person/Agent Configuration Manager

Name: Person/ Agent Configuration Manager

Intent: Represents *persons* and *agents* or machines as *configuration managers.*

Rationale: *Persons* (playing the role of *Person Configuration Manager*) and *Agents* (playing the role of *Agent Configuration Manager*) can act as *Configuration Managers*, i.e., the ones responsible for the configuration management (semi-automatic).

Competency Questions:

✓ *What are the possible types (person, agent or both) of configuration manager?*

Conceptual Model:

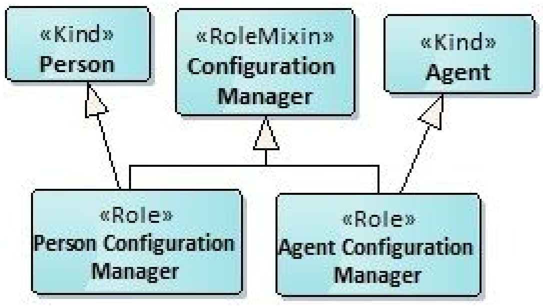

Figure 10 PA-Manager – Conceptual Model

Axiomatization: -

FOPs Support: Rolemixin Pattern – Variant 2.

Term Definitions:

Person	An individual human being.
Person Configuration Manager	The role played by a *Person* as a *Configuration Manager*.
Configuration Manager	The role played by a *Person* and *an Agent* when they manage a configuration of an item.
Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Configuration Manager	The role played by an *Agent* as a *Configuration Manager*.

ITyPE - Item Type

Name: Item Type

Intent: Defines the *Item* type that is the subject of the configuration. Any item which can have change.

Rationale: models the *Item* itself, when we cannot model the item either as an artifact or as a software tool.

Competency Questions:

- ✓ *For which do I need to manage the configuration?*
- ✓ *Which are the types of configured items that I need to manage?*

Conceptual Model:

Figure 11 IType – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Item	A product that can evolve through new configurations.

IArtifact - Item Artifact

Name: Item Artifact

Intent: Represents an item that is the subject of the configuration as an artifact.

Rationale: *Artifact* is a Category that aggregates properties that are common to *Source Code*, *Document,* and *Diagram*. These items have their principle of identity and become a kind type (rigid sortals). Also, *Artifact* is the specialization of *Item* Category because *Artifact* can be an *Item* that evolves.

Competency Questions:
- ✓ *For which do I need to manage the configuration?*
- ✓ *Which are the types of configured items that I need to manage?*

Conceptual Model:

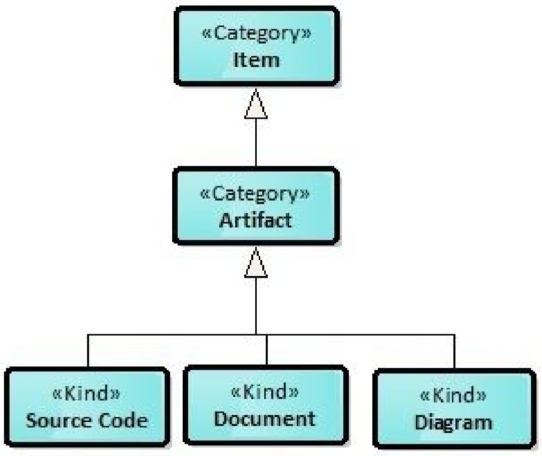

Figure 12 IArtifact – Conceptual Model

Axiomatization: -

FOPs Support: Category Pattern – Variant 2.

Term Definitions:

Item	A product that can evolve through new configurations.
Artifact	Any source code, document, diagram or any tangible items produced during the development of the product.
Source Code	The source code of a software.
Document	Any document that evolves and can be its configuration managed.
Diagram	Any diagram that evolves and can be its configuration managed.

ITool - Item Tool

Name: Item Tool

Intent: Represents an item that is the subject of the configuration as a tool.

Rationale: *Software Tool* is a kind that is the specialization of *Item* Category because *Software Tool* can be an *Item* that evolves.

Competency Questions:

- ✓ *For which do I need to manage the configuration?*
- ✓ *Which are the types of configured items that I need to manage?*

Conceptual Model:

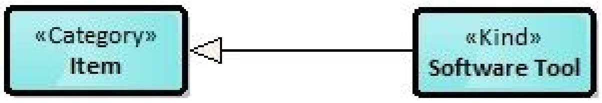

Figure 13 ITool – Conceptual Model

Axiomatization: -

FOPs Support: Category Pattern – Variant 2.

Term Definitions:

Item	A product that can evolve through new configurations.
Software Tool	A program employed in the development, repair, or enhancement of other programs or hardware.

IAtomic - Item Atomic

Name: Item Atomic

Intent: Represents an atomic configuration item of the product/item which could be configured and managed.

Rationale: When an *Item* is atomic, i. e. , it does not have others items, it can specialize in a rolemixin called *Configuration Item*. It is classified as rolemixin because it is an anti-rigid type whose instantiation depends on a relational property (as a role of an *Item* Category).

Competency Questions:

- ✓ *Is the configuration item complex (composed of other items)? In this case, which atomic items compose this composite item?*

Conceptual Model:

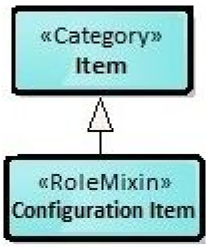

Figure 14 IAtomic – Conceptual Model

Axiomatization: -

FOPs Support: Category Pattern – Variant 1.

Term Definitions:

Item	A product that can evolve through new configurations.
Configuration Item	An item of product that enables management of a configuration.

ICompositeItem - Item Composite

Name: Item Composite

Intent: Represents a composite configuration item of the product/item which could be configured and managed.

Rationale: when others configuration items compose a Configuration Item, there is a relationship ComponentOf between *Configuration Item* and *Composite CI*. If it is composite, this means that it has at least two *Configuration Items*. These parts of a *Composite CI* can be an *AtomicCI* or another *Composite CI*. So, *Composite CI* and *AtomicCI* are a specialization of *Configuration Item* and classified as rolemixin. *Configuration Item* is a role of the *Item* Category (rolemixin).

Competency Questions:

✓ *Is the configuration item complex (composed of other items)? In this case, which atomic items compose this composite item?*

Conceptual Model:

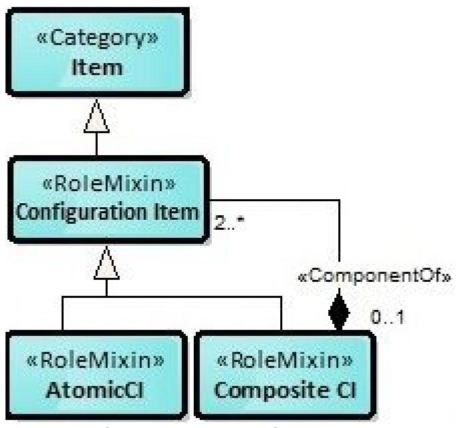

Figure 15 IComposite – Conceptual Model

Axiomatization:

```
A1 | ∀ ci: ConfigurationItem, cci: CompositeCI (isA(ci, cci)) →
   | (ComponentOf(ci,cci) ^ ∃cii: ConfigurationItem ^ Componen-
   | tOf(cii,cci))
```

FOPs Support: Category Pattern – Variant 1.

Term Definitions:

Item	A product that can evolve through new configurations.
Configuration Item	An item of product that has a configuration that can be managed.
AtomicCI	Configuration Item that is not composed by another one.
Composite CI	Configuration Item composed by others configuration items.

IBaseline - Item Baseline

Name: Item Baseline

Intent: Defines a configuration snapshot at any given time to the configured item.

Rationale: A *Markup* mediates the relation between a *Configuration Manager* and a *Baseline*. When a *Configuration* of a *Version* receives a *markup*, it plays a role of *Baseline*. *Configuration Manager* is the role played by the *persons*, the *agents* or both when they become a *Configuration Manager*. The stereotype of the *Configuration Manager* class is given by the pattern selected from the *Configuration Manager* sub-group.

Competency Questions:
- ✓ *Which Configuration has the Configuration Manager set as a Baseline?*

Conceptual Model:

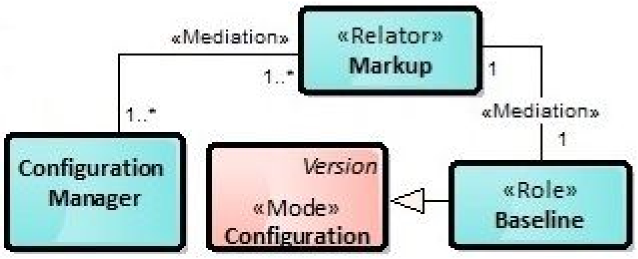

Figure 16 IBaseline – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1.

Term Definitions:

Markup	Markup in the product to indicate the extent to which evolution can suit as a reference (baseline) for making changes.
Configuration Manager	The role played by a *Person* and *an Agent* when they manage a configuration of an item.
Baseline	Configuration snapshot at any given time. When a product configuration that has been revised and designed to serve as a reference for future development or changes. It is a reference formally defined at a particular stage in the evolution of a product lifecycle.
Configuration	Set of physical and functional characteristics that describe the product at a given time.

5.2 Version Control Group

IVersion - Item Version

Name: Item Version

Intent: Represents the version of the item that has configuration changed.

Rationale: models the *Version* of an item itself.

Competency Questions:

- ✓ *Which version of the item will be changed?*

Conceptual Model:

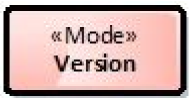

Figure 17 IVersion – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.

IVariant - Item Variant

Name: Item Variant

Intent: Represents the variation of the item – parallel versions.

Rationale: An *Item* may have multiple *Versions*. *Versions* of items that may exist in parallel are said to be *Variant*. So, *Variant* is a mode of the *Version*, i. e., intrinsic moments in one single individual of the *Version*.

Competency Questions:

- ✓ *Does the version change correspond to a revision or a parallel version (variant)?*

Conceptual Model:

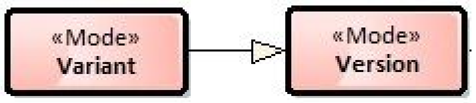

Figure 18 IVariant – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of product development.
Variant	A parallel version of an item with specific characteristics that differ from other versions.

IRevision - Item Revision

Name: Item Revision

Intent: Represents the revision of the item – when versions overwrite others versions.

Rationale: An *Item* may have multiple *Versions*. *Versions* of items that may overlap others *Versions* are said to be *Revision*. So, *Revision* is a mode of the *Version*, i. e., intrinsic moments in one single individual of the *Version*.

Competency Questions:
- ✓ *Does the version change correspond to a revision or a parallel version (variant)?*

Conceptual Model:

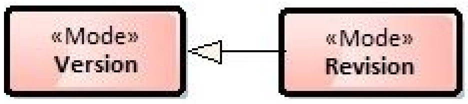

Figure 19 IRevision – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of product development.
Revision	A revised version of an item that overlaps another (original) version.

IVAtomic - Item Version Atomic

Name: Item Version Atomic

Intent: Represents an atomic version of the CI.

Rationale: If the *Configuration Item* is atomic means that it has one *version* at least (itself). So, the *Version* of the Item has an *Atomic Version* mode.

Competency Questions:
- ✓ *CQ12: Does the item has only one version?*

Conceptual Model:

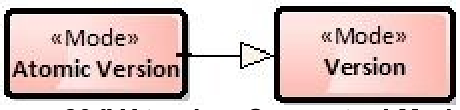

Figure 20 IVAtomic – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.
Atomic Version	A version of an atomic *Configuration Item.*

IVConfiguration - Item Version Configuration

Name: Item Version Configuration

Intent: Represents the configuration with a composite CI.

Rationale: If the *Configuration Item* is composite, it means that it has two *versions* at least. If there is a Configuration, the *Item* has its characteristics changed. Therefore, the *Version* of the *Item* has a *Configuration* mode, and the *Versions* of the *Item* (before and after the *configuration* change) is a component of the *Configuration*.

Competency Questions:

✓ *What versions are component of the configuration?*

Conceptual Model:

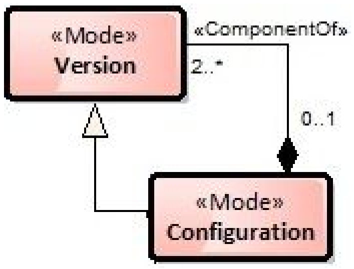

Figure 21 IVConfiguration – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.
Configuration	Set of physical and functional characteristics that describe the product at a given time. It is a version of the composite *Configuration Item.*

IRepository - Item Repository

Name: Item Repository

Intent: Represents the location that holds all versions of CI, including the project, repository and its ramifications composed of some versions of CI.

Rationale: the *Branch* concept that is a Collective type because it is a Kind whose instances are composed of collections of different *Versions*. Consequently, we have a *Project* (Kind) that owns *Repository* (Collective), which is composed by many *Branches* (SubcollectionOf).

Competency Questions:

- ✓ *Where can I get all versions of the item?*
- ✓ *What is the project that corresponds to the version?*
- ✓ *What are the branches with the versions?*

Conceptual Model:

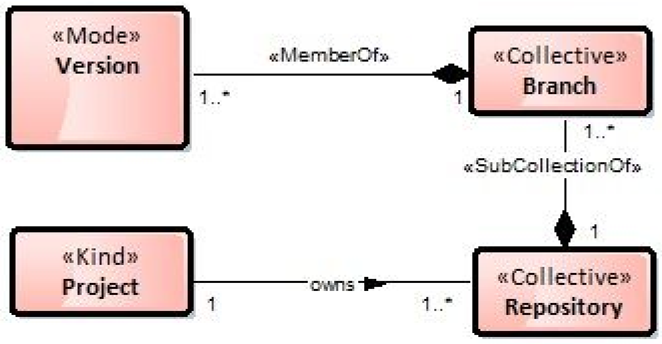

Figure 22 IRepository – Conceptual Model

Axiomatization: -

FOPs Support: Collective Pattern.

Term Definitions:

Project	An individual or collaborative effort that is carefully planned and designed to achieve a particular aim such as developing a product.
Repository	Organizes the versions of *items* in a project.
Branch	A subcollection of a repository that has the versions of items in the same evolution line.
Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.

ICheckout - Item Check-out

Name: Item Check-out

Intent: Represents the last version of the item that will be changed.

Rationale: The *Change* that is in progress (as role *On Going Change*) in the *Version* of the item that was downloaded or prepared (as role *Checked-Out Version*) for modification. A *Check-Out* mediates the relation between a *Version (Checked-Out Version)*, a *Change (On-Going Change*) and an *Executor*.

Competency Questions:

- ✓ *Which version of the item does the person/agent want to modify or checked out?*
- ✓ *Who checked out the version to modify in the future?*
- ✓ *Which change is going to the item?*

Conceptual Model:

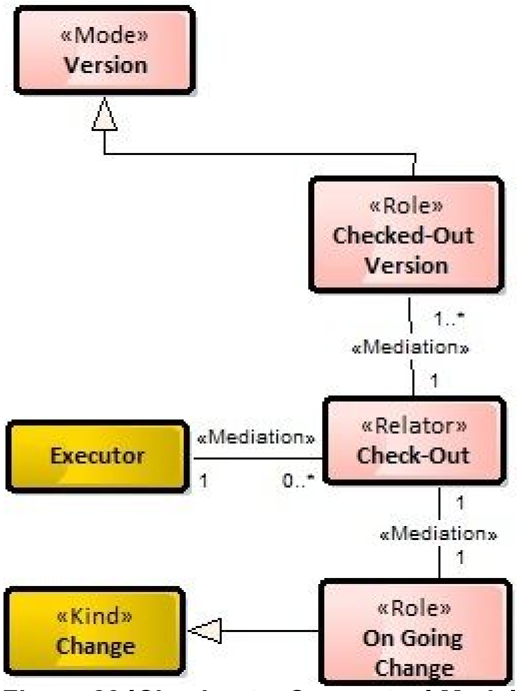

Figure 23 ICheckout – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1 and Mode Pattern.

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.
Checked-Out Version	The version that will be changed.
Check-Out	Recording of the withdrawal of an item to make a change.
Executor	The role played by a *Person, an Agent* or both when they execute a configuration change of an item.
On-Going Change	Change an item in progress.
Change	Record of the modification action of an item version.

ICheckin - Item Check-in

Name: Item Check-in

Intent: Represents the register of the version of the modified item.

Rationale: when an *Implemented Change* (role) occurs, a *Check-In* is established, and it corresponds to a new *Version* of the *Configuration Item* that is registered. The *Implemented Change* has a mediation relationship with *Version* through the *Check-In* Relator, and the modification of the item has a role of *Registered Modification* as there is a *check-in*.

Competency Questions:

- ✓ *Which the new version does the person/agent want to become current version?*
- ✓ *Who implemented the new version that will be checked-in?*

Conceptual Model:

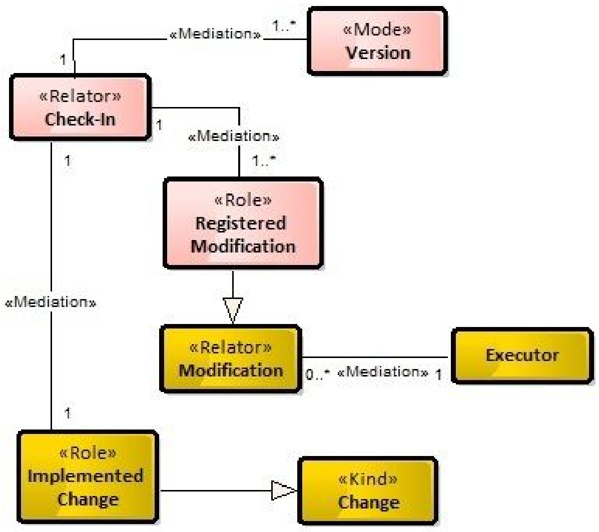

Figure 24 ICheckin – Conceptual Model

Axiomatization:

A1	`∀cki: Check-In, rm: RegisteredModification, v: Version (generates(cki, v)) ^ enables(rm,cki) → (∃c: Change ^∃ic: Implemented-Change ^ isA(ic,c))`

FOPs Support: Relator Pattern – Variant 1 and Mode Pattern.

Term Definitions:

Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.
Check-In	Records of changed items.
Registered Modification	Records of the change.
Modification	Records the action of the change of an item version.
Executor	The role played by a *Person, an Agent* or both when they execute a configuration change of an item.
Implemented Change	Specified change that has been implemented and recorded through a check-in.
Change	Specifying a modification to be performed on items that may or may not be implemented.

5.3 Change Control Group

P-Requester - Person Requester

Name: Person Requester

Intent: Represents persons as requesters.

Rationale: *Persons* can act as (play the role of) *Requester*, i. e., the ones responsible for the configuration change request.

Competency Questions:
- ✓ *Who requested the modification of the configuration item?*

Conceptual Model:

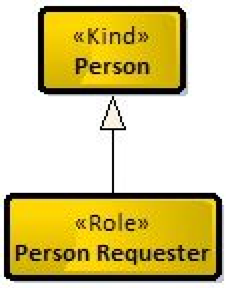

Figure 25 P-Requester – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Person	An individual human being.
Person Requester	The role played by a Person as a Requester of the configuration change.

A-Requester - Agent Requester

Name: Agent Requester

Intent: Represents agents/machines as requesters.

Rationale: *Software Agents* or machines can act as (play the role of) *Requester*, i. e., the ones responsible for the configuration change request (automatic).

Competency Questions:

- ✓ *Who requested the modification of the configuration item?*

Conceptual Model:

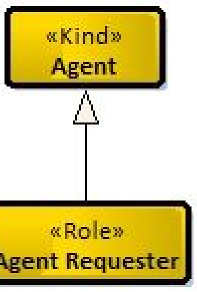

Figure 26 A-Requester – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Requester	The role played by an *Agent* as a Requester of the configuration change.

PA-Requester - Person/Agent Requester

Name: Person/ Agent Requester

Intent: Represents persons and agents or machines as requesters.

Rationale: *Persons* (playing the role of *Person Requester*) and *Agents* (playing the role of *Agent Requester*) can act as *Requesters*, i.e., the ones responsible for the configuration change request (semi-automatic).

Competency Questions:

✓ *Who requested the modification of the configuration item?*

Conceptual Model:

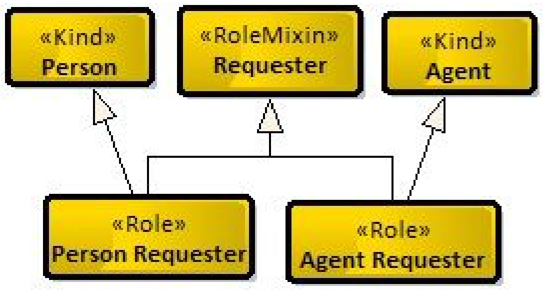

Figure 27 PA-Requester – Conceptual Model

Axiomatization: -

FOPs Support: Rolemixin Pattern – Variant 2.

Term Definitions:

Person	An individual human being.
Person Requester	The role played by a Person as a Requester of the configuration change.
Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Requester	The role played by an *Agent* as a Requester of the configuration change.
Requester	The role played by a *Person* and *an Agent* when they request a configuration of an item.

IRequestV - Item Version Request

Name: Item Version Request

Intent: Represents the change request mediated by a Requester and a version that is submitted for change.

Rationale: A *Change Request* mediates the relation among a *Requester*, a *Version*, and a *Change*. When a Version is submitted for *Change*, it plays a role of *Version Submitted For Change*. So, when the *Requester* requests a *Change* of an *Item*, the *Version is submitted for change*.

Competency Questions:

- ✓ *Who requested the modification of the configuration item?*
- ✓ *Which change the person/agent requests?*
- ✓ *Which item version or configuration item the person/agent submitted for a change?*

Conceptual Model:

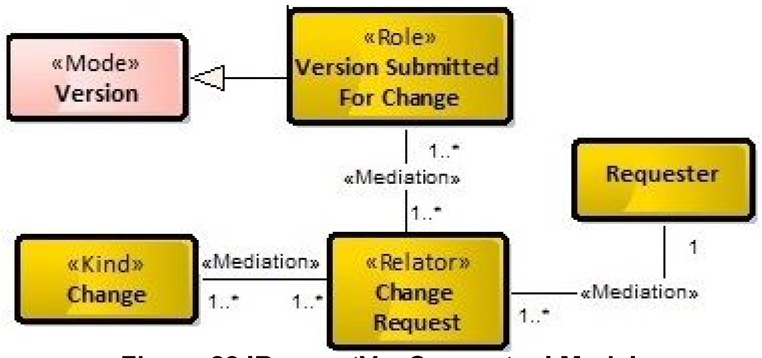

Figure 28 IRequestV – Conceptual Model

Axiomatization:

A1	`∀cr: ChangeRequest, vs: VersionSubmittedForChange, r: Requester (requests(r, cr)) ^ enables(cr,vs) → (∃c: Change ^ correspond-sTo(c,cr))`

FOPs Support: Relator Pattern – Variant 1 and Mode Pattern.

Term Definitions:

Requester	The role played by a *Person* and *an Agent* when they request a configuration of an item.
Change Request	Request for change by a *Requester* to change the configuration of an item.
Change	Specifying a modification to be performed on items that may or not be implemented.
Version Submitted For Change	A version of the item that is submitted for a configuration change.
Version	Represents a specific state of a *Configuration Item* at a given point in time of product development.

IRequest - Configuration Item Request

Name: Configuration Item Request

Intent: Represents the change request mediated by a Requester and a configuration item, which submitted for change.

Rationale: A *Change Request* mediates the relation among a *Requester*, a *Configuration Item,* and a *Change*. When there is not a Version Control, the *Configuration Item* is submitted for *Change,* and it plays a role of *CI Submitted For Change*. So, when the *Requester* requests a *Change* of an *Item*, the *Configuration Item is submitted for change*.

Competency Questions:

- ✓ *Who requested the modification of the configuration item?*
- ✓ *Which change the person/agent requests?*
- ✓ *Which item version or configuration item the person/agent submitted for a change?*

Conceptual Model:

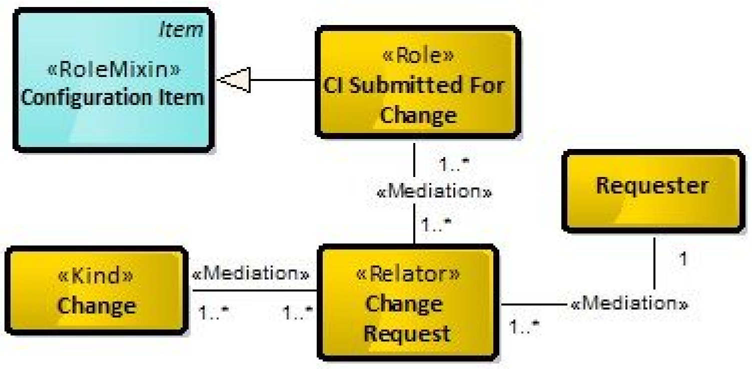

Figure 29 IRequest – Conceptual Model

Axiomatization:

A1 ∀cr: ChangeRequest, cis: CISubmittedForChange, r: Requester (requests(r, cr)) ^ enables(cr,cis) → (∃c: Change ^ correspondsTo(c,cr))

FOPs Support: Relator Pattern – Variant 1.

Term Definitions:

Requester	The role played by a *Person* and *an Agent* when they request a configuration of an item.
Change Request	Request for change by a *Requester* to change the configuration of an item.
Change	Specifying a modification to be performed on items that may or not be implemented.
CI Submitted For Change	Configuration item that is submitted for a configuration change.
Configuration Item	An item of product that has a configuration that can be managed.

P-Evaluator - Person Evaluator

Name: Person Evaluator

Intent: Represents persons as evaluators.

Rationale: *Persons* can act as (play the role of) *Evaluator*, i. e., the ones responsible for the configuration change evaluation.

Competency Questions:

- ✓ *Who evaluated if the modification of the configuration item is possible?*

Conceptual Model:

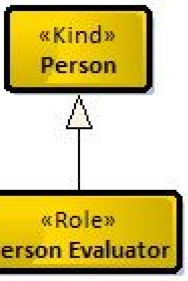

Figure 30 P-Evaluator – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Person	An individual human being.
Person Evaluator	The role played by a Person as an Evaluator of a configuration request.

A-Evaluator - Agent Evaluator

Name: Agent Evaluator

Intent: Represents agents/machines as evaluators.

Rationale: *Software Agents* or machines can act as (play the role of) *Evaluator*, i. e., the ones responsible for the configuration change evaluation (automatic).

Competency Questions:

✓ *Who evaluated if the modification of the configuration item is possible?*

Conceptual Model:

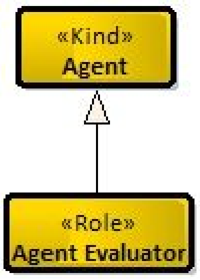

Figure 31 A-Evaluator – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Evaluator	The role played by an *Agent* as an Evaluator of the configuration change.

PA-Evaluator - Person/Agent Evaluator

Name: Person/Agent Evaluator

Intent: Represents persons and agents or machines as evaluators.

Rationale: *Persons* (playing the role of *Person Evaluator*) and *Agents* (playing the role of *Agent Evaluator*) can act as *Evaluators*, i.e., the ones responsible for the configuration change evaluation (semi-automatic).

Competency Questions:

✓ *Who evaluated if the modification of the configuration item is possible?*

Conceptual Model:

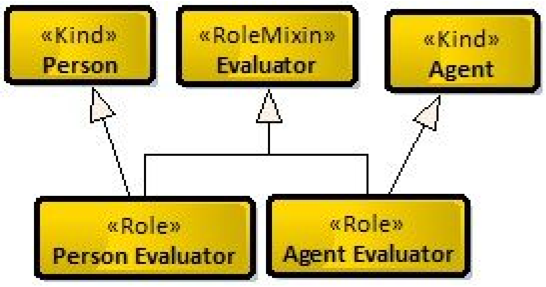

Figure 32 PA-Evaluator – Conceptual Model

Axiomatization: -

FOPs Support: Rolemixin Pattern – Variant 2.

Term Definitions:

Person	An individual human being.
Person Evaluator	The role played by a Person as an Evaluator of a configuration request.
Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Evaluator	The role played by an *Agent* as an Evaluator of the configuration change.
Evaluator	The role played by a *Person* and *an Agent* when they evaluate a configuration of an item.

IEvChRequest - Configuration Item Evaluation Change Request

Name: Configuration Item Evaluation Change Request

Intent: Represents the evaluation if the item can have the CI applied.

Rationale: When a *Change Request* is evaluated (as a role *Evaluated Request*), it can be accepted or not. This result is represented as a quality of the relator *Request Evaluation*. The *Evaluator* is responsible to the *Request Evaluation*.

Competency Questions:

✓ *What the result of the evaluation of the change request?*

Conceptual Model:

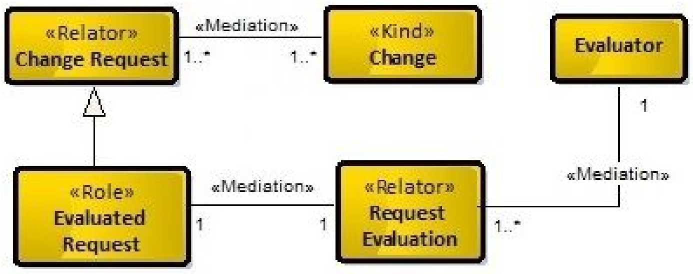

Figure 33 IEvChRequest – Conceptual Model

Axiomatization:

A1	∀re: RequestEvaluation, er: EvaluatedRequest, e: Evaluator (evaluates(e, re)) ^ enables(re,er) → (∃cr: ChangeRequest ^∃c: Change ^ isA(er,cr) ^ correspondsTo(cr,c) ^)

FOPs Support: Relator Pattern – Variant 1 and Relational Dependence Pattern.

Term Definitions:

Evaluator	The role played by a *Person* and *an Agent* when they evaluate a configuration of an item.
Request Evaluation	Record the action made by an evaluator of evaluating a change request.
Evaluated Request	When an Evaluator evaluates the change request.
Change Request	Request for change by a *Requester* to change the configuration of an item.
Change	Specifying a modification to be performed on items that may or not be implemented.

P-Executor - Person Executor

Name: Person Executor

Intent: Represents persons as executors.

Rationale: *Persons* can act as (play the role of) *Executor*, i. e., the ones responsible for the configuration change execution.

Competency Questions:
- ✓ *Who executed the modification of the configuration item?*

Conceptual Model:

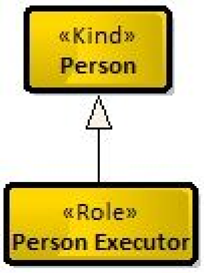

Figure 34 P-Executor – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Person	An individual human being.
Person Executor	The role played by a Person as an Executor of a configuration change.

A-Executor - Agent Executor

Name: Agent Executor

Intent: Represents agents/machines as executors.

Rationale: *Software Agents* or machines can act as (play the role of) *Executor*, i. e., the ones responsible for the configuration change execution (automatic).

Competency Questions:

✓ *Who executed the modification of the configuration item?*

Conceptual Model:

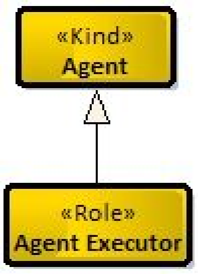

Figure 35 A-Executor – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Executor	The role played by an *Agent* as an Executor of the configuration change.

PA-Executor - Person/Agent Executor

Name: Person/Agent Executor

Intent: Represents persons and agents or machines as executors.

Rationale: *Persons* (playing the role of *Person Executor*) and *Agents* (playing the role of *Agent Executor*) can act as *Executors*, i.e., the ones responsible for the configuration change execution (semi-automatic).

Competency Questions:

✓ *Who executed the modification of the configuration item?*

Conceptual Model:

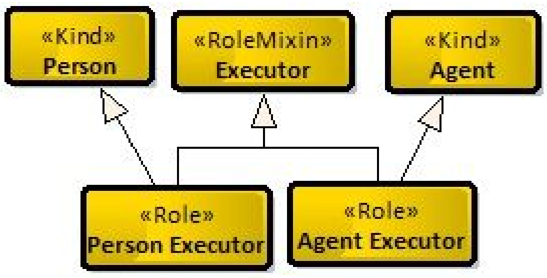

Figure 36 PA-Executor – Conceptual Model

Axiomatization: -

FOPs Support: Rolemixin Pattern – Variant 2.

Term Definitions:

Person	An individual human being.
Person Executor	The role played by a Person as an Executor of a configuration change.
Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Executor	The role played by an *Agent* as an Executor of the configuration change.
Executor	The role played by a *Person, an Agent* or both when they execute a configuration change of an item.

IExecRequestV - Item Version Execution Request

Name: Item Version Execution Request

Intent: Represents the execution of the change in a version of the configuration item.

Rationale: The effective configuration is developed and implemented. A *Modification* mediates the relationship between the roles *Executor* and *Modified Version*.

Competency Questions:

- ✓ *Who executed the modification of the configuration item?*
- ✓ *Which modification or change the person/agent does?*
- ✓ *Which modified version or configuration item the person/agent generates?*

Conceptual Model:

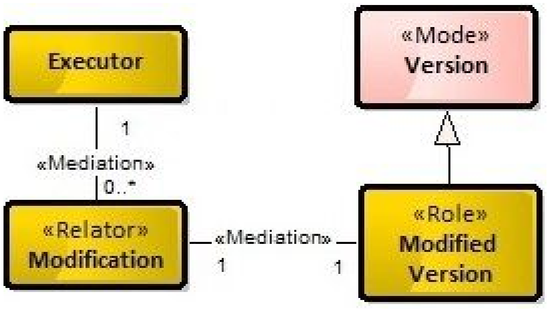

Figure 37 IExecRequestV – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1 and Mode Pattern.

Term Definitions:

Executor	The role played by a *Person, an Agent* or both when they execute a configuration change of an item.
Modification	Records the modify action for a version.
Modified Version	Records the modified version of an item.
Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.

IExecRequest - Configuration Item Execution Request

Name: Configuration Item Execution Request

Intent: Represents the execution of the change in a configuration item.

Rationale: The effective configuration is developed and implemented. A *Modification* mediates the relationship between the roles *Executor* and *Modified CI*.

Competency Questions:

- ✓ *Who executed the modification of the configuration item?*
- ✓ *Which modification or change the person/agent does?*
- ✓ *Which modified version or configuration item the person/agent generates?*

Conceptual Model:

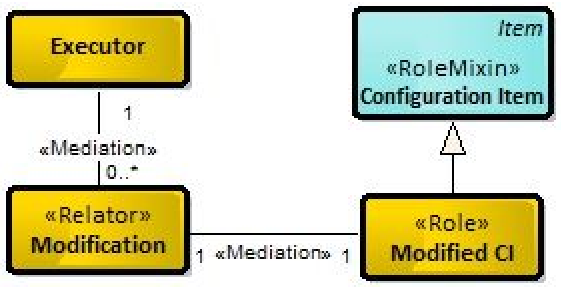

Figure 38 IExecRequest – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1.

Term Definitions:

Executor	The role played by a *Person, an Agent* or both when they execute a configuration change of an item.
Modification	Records the modify action for a configuration item.
Modified CI	Records the modified configuration item.
Configuration Item	An item of product that has a configuration that can be managed.

P-Verifier - Person Verifier

Name: Person Verifier

Intent: Represents persons as verifiers.

Rationale: *Persons* can act as (play the role of) *Verifier*, i. e., the ones responsible for the configuration change validation.

Competency Questions:
- ✓ *Who validated the item after the modification?*

Conceptual Model:

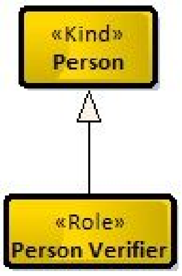

Figure 39 P-Verifier – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Person	An individual human being.
Person Verifier	The role played by a Person as a Verifier of a configuration change.

A- Verifier - Agent Verifier

Name: Agent Verifier

Intent: Represents agents/machines as verifiers.

Rationale: *Software Agents* or machines can act as (play the role of) *Executor*, i. e., the ones responsible for the configuration change execution (automatic).

Competency Questions:
- ✓ *Who validated the item after the modification?*

Conceptual Model:

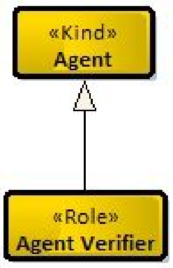

Figure 40 A-Verifier – Conceptual Model

Axiomatization: -

FOPs Support: -

Term Definitions:

Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Verifier	The role played by an *Agent* as a Verifier of the configuration change.

PA- Verifier - Person/Agent Verifier

Name: Person/ Agent Verifier

Intent: Represents persons and agents or machines as verifiers.

Rationale: *Persons* (playing the role of *Person Verifier*) and *Agents* (playing the role of *Agent Verifier*) can act as *Verifiers*, i.e., the ones responsible for the configuration change validation (semi-automatic).

Competency Questions:

✓ *Who validated the item after the modification?*

Conceptual Model:

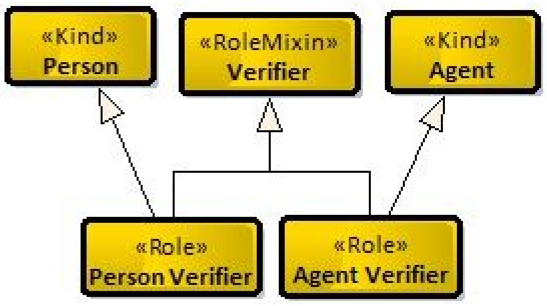

Figure 41PA-Verifier – Conceptual Model

Axiomatization: -

FOPs Support: Rolemixin Pattern – Variant 2.

Term Definitions:

Person	An individual human being.
Person Verifier	The role played by a Person as a Verifier of a configuration change.
Agent	Encapsulated system that is situated in an environment and that presents characteristics like flexibility and autonomy to reach its objectives.
Agent Verifier	The role played by an *Agent* as a Verifier of the configuration change.
Verifier	The role played by a *Person* and *an Agent* when they validate a configuration of an item.

IVerRequest - Configuration Item Verification Request

Name: Configuration Item Verification Request

Intent: Represents the verification of the item with the CI applied through a specification.

Rationale: the validation of the configuration. This pattern captures the *Change* verified by the *Verifier* (*Verification*). The *Implemented Change* is a role of the *Change* (Kind) when the *Check-In* operation (Relator) occurs that is a committed version submitted to the repository. After the validation of the *Change*, the *Change* assumes the role of a *Verified Change*.

Competency Questions:

✓ *Has the change been effectively implemented?*

Conceptual Model:

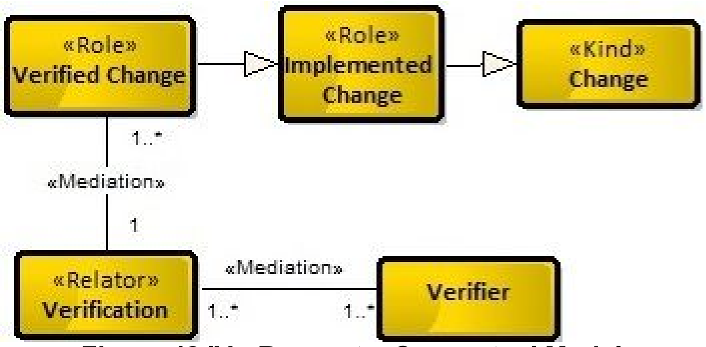

Figure 42 IVerRequest – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1.

Term Definitions:

Verifier	The role played by a *Person* and *an Agent* when they validate a configuration of an item.
Verification	Validates the configuration change of an item.
Verified Change	Records the verified change of an item.
Implemented Change	Records the implemented change of a configuration item.
Change	Specifying a modification to be performed on items that may or not be implemented.

References

CALHAU, R. F. Uma abordagem baseada em ontologias para integração semântica de sistemas. MSc thesis presented in Federal University of Espirito Santo, 2011.

CALHAU, R. F., FALBO, R. A. A Configuration Management Task Ontology for Semantic Integration, Proceedings of the ACM/SIGAPP Symposium On Applied Computing (SAC 2012), pp. 348-353, 2012.

FALBO, R.A., GUIZZARDI, G., GANGEMI, A. AND PRESUTTI, V. Ontology patterns: clarifying concepts and terminology. In Proc. of the 4th Workshop on Ontology and Semantic Web Patterns, Sidney, Australia, 2013.

GUIZZARDI, G. Ontological Foundations for Structural Conceptual Models. In: Universal Press, The Netherlands, 2005.

QUIRINO, G. K. S., BARCELLOS, M. P., FALBO, R. OPL-ML: A Modeling Language for Representing Ontology Pattern Languages, Lecture Notes in Computer Science, November 2017, pp. 187-201, 2017.

QUIRINO, G. K., FALBO, R. A., BARCELLOS, M. P., NARDI, J. C. S-OPL: Service Ontology Pattern Language. Specification. Version 1.6. April, 2017, available in: https://nemo.inf.ufes.br/wp-content/uploads/2017/04/s_opl_v1_6.pdf, accessed in June, 2018.

RUY, F. B., GUIZZARDI, G., FALBO, R. A., REGINATO, C. C., SANTOS, V. A. From Reference Ontologies to Ontology Patterns and Back, Journal Data & Knowledge Engineering, May 2017, v. 109, issue C, pp. 41-69, 2017.