PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 09/18

CM-OPL: Configuration Management Ontology Pattern Language Specification – Revised Edition

Ana Carolina Brito de Almeida
Maria Luiza Machado Campos
Fernanda Baião
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. 09/18 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December, 2018

CM-OPL: Configuration Management Ontology Pattern Language Specification – Revised Edition

Ana Carolina Brito de Almeida[1], Maria Luiza Machado Campos[2]
Fernanda Baião, 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, {fbaiao, dschwabe, sergio}@inf.puc-rio.br

Abstract. This document presents the Configuration Management Ontology Pattern Language (CM-OPL). It is the second 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 segunda 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 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.
CIDecomposition	Configuration Item Decomposition	Represents a decomposition of the configuration item of the product/item which could be configured and managed.

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 configuration item and can be atomic or composite. A composite CI has others versions, and they are called configuration. The Atomic CI can have 1..* atomic versions. When a configuration has a markup, it practices the role of baseline done by Configuration Manager.

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
CIVersion	Configuration Item Version	Represents the version of the configuration item that has configuration changed.
CIVDecomposition	Configuration Item Version Decomposition	Represents the decomposition of versions: an atomic or composite/configuration version of the CI.
CIVBaseline	Configuration Item Version Baseline	Defines a configuration snapshot to the CI verison at any given time.
CIVMode	Configuration Item Version Mode	Represents the variation of the configuration item version – parallel versions or the revision of the item – when versions overwrite others versions.

Change Control deals with change management during the product life cycle. The Requester requires a change of a configuration 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. 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 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.
CIVCRequest	Configuration Item Version Change Request	Represents the change request mediated by a Requester and a version, when 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.
CIVCREvaluation	Configuration Item Version Change Request Evaluation	Represents the evaluation if the configuration item version can have the change applied.
CIVCheckout	Configuration Item Version Check-out	Represents the last version of the configuration item that will be changed.
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.
CIVCRExecution	Configuration Item Version Change Request Execution	Represents the execution of the change in a version of the configuration item.
CIVCheckin	Configuration Item Version Check-in	Represents the register of the version of the modified 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.
CIVCRVerification	Configuration Item Version Change Request Verification	Represents the verification of the configuration item version with the change 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* with red dotted edges *delimit groups of variant pattern*s. *Variant patterns* are patterns that solve the same problem but in different ways. Thus, from a set of variant patterns, only one can be used to solve the problem when developing an ontology. 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 pattern application action from different groups.

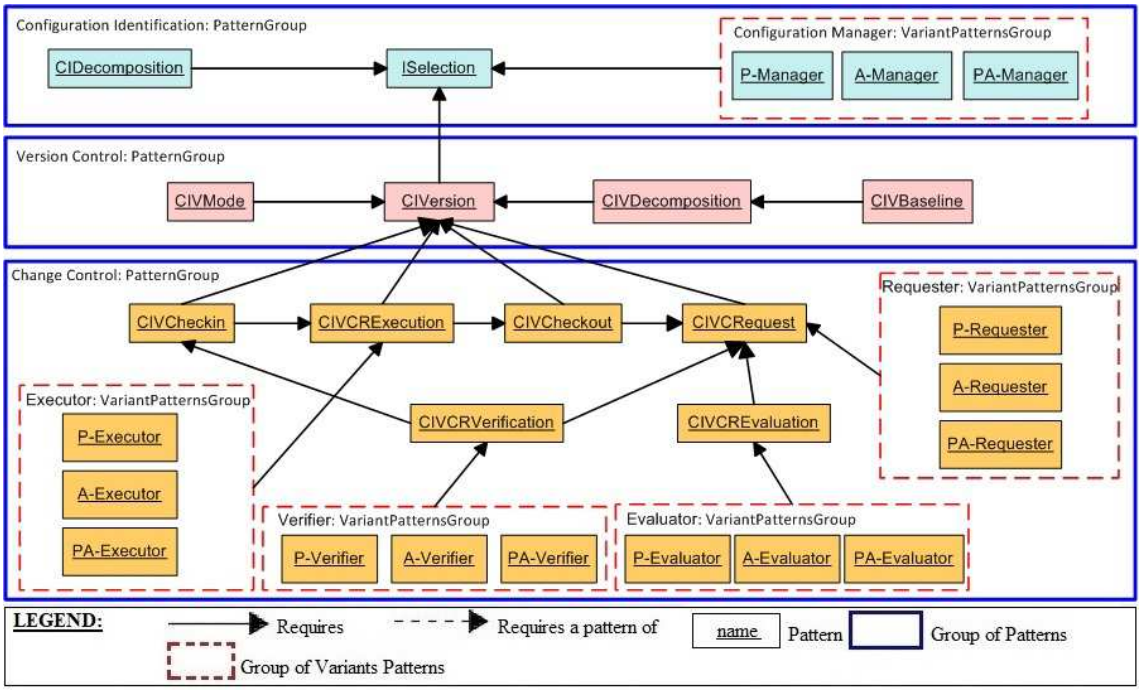

Figure 1 CM-OPL Structural Model

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. 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 pattern application process can be finished. Like in the structural model, different colors are used in the process models (Figures 2-6) 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.

Figure 2 – CM-OPL Process (general view)

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 graphical representation of the structural model. Moreover, we group *variant pattern application actions* inside rectangles with red dotted edges.

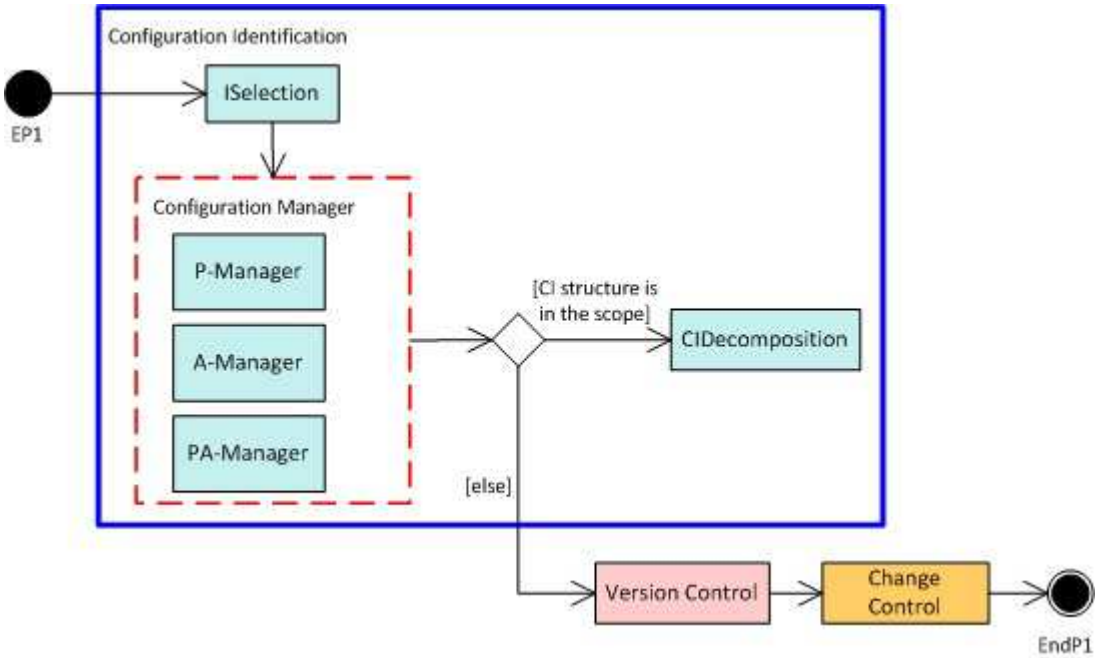

Figure 3 – Detailed Process Model of the Configuration Identification Group

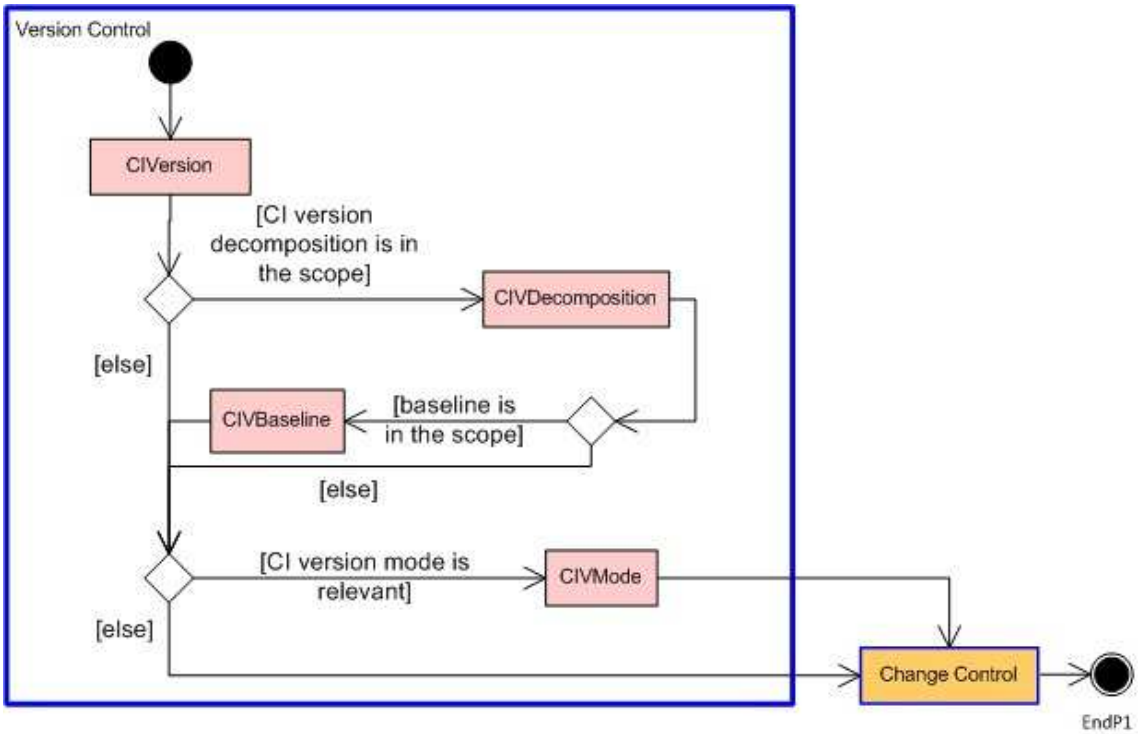

Figure 4 - Detailed Process Model of the Version Control Group

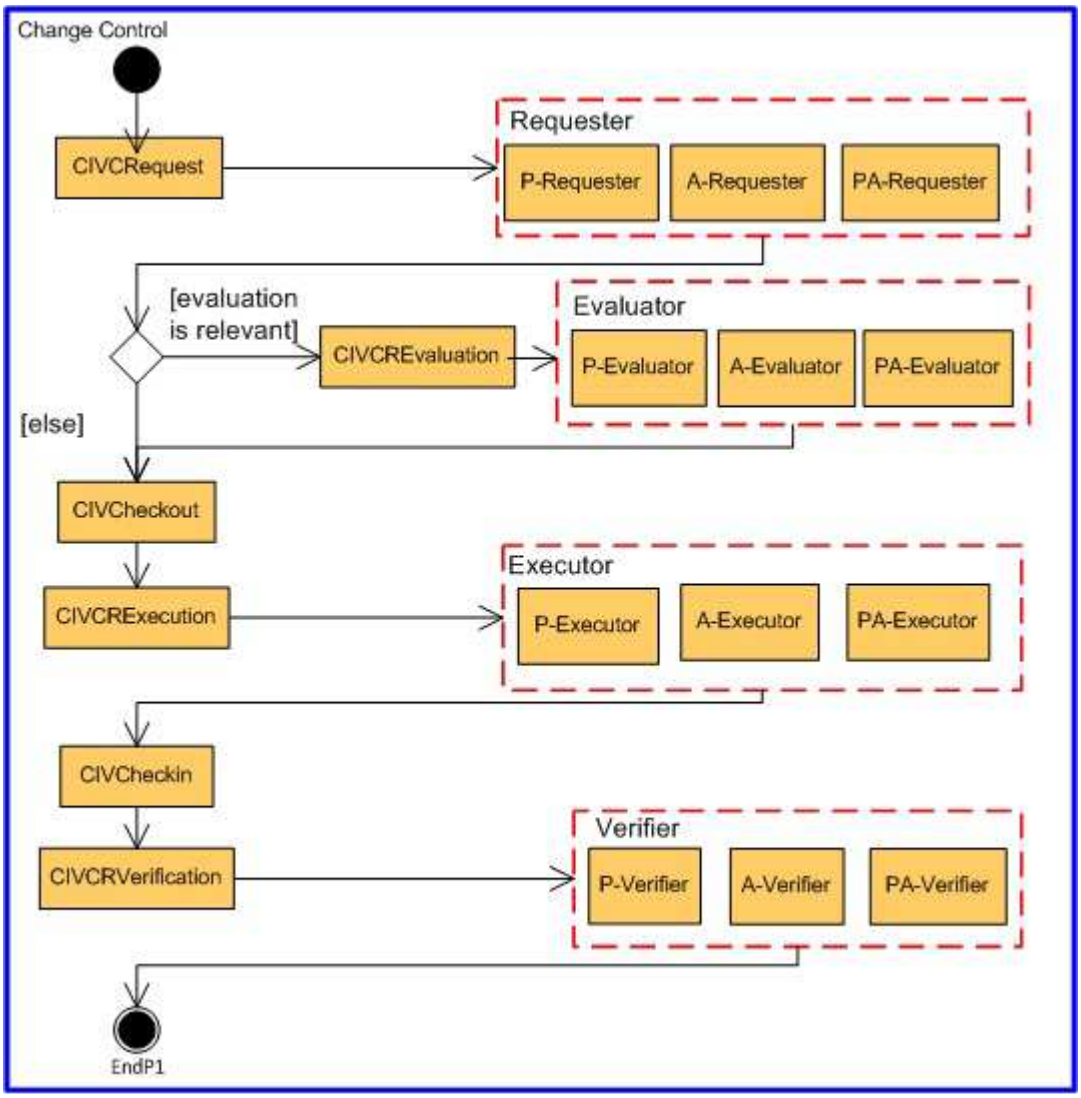

Figure 5 - Detailed Process Model of the Change Control Group

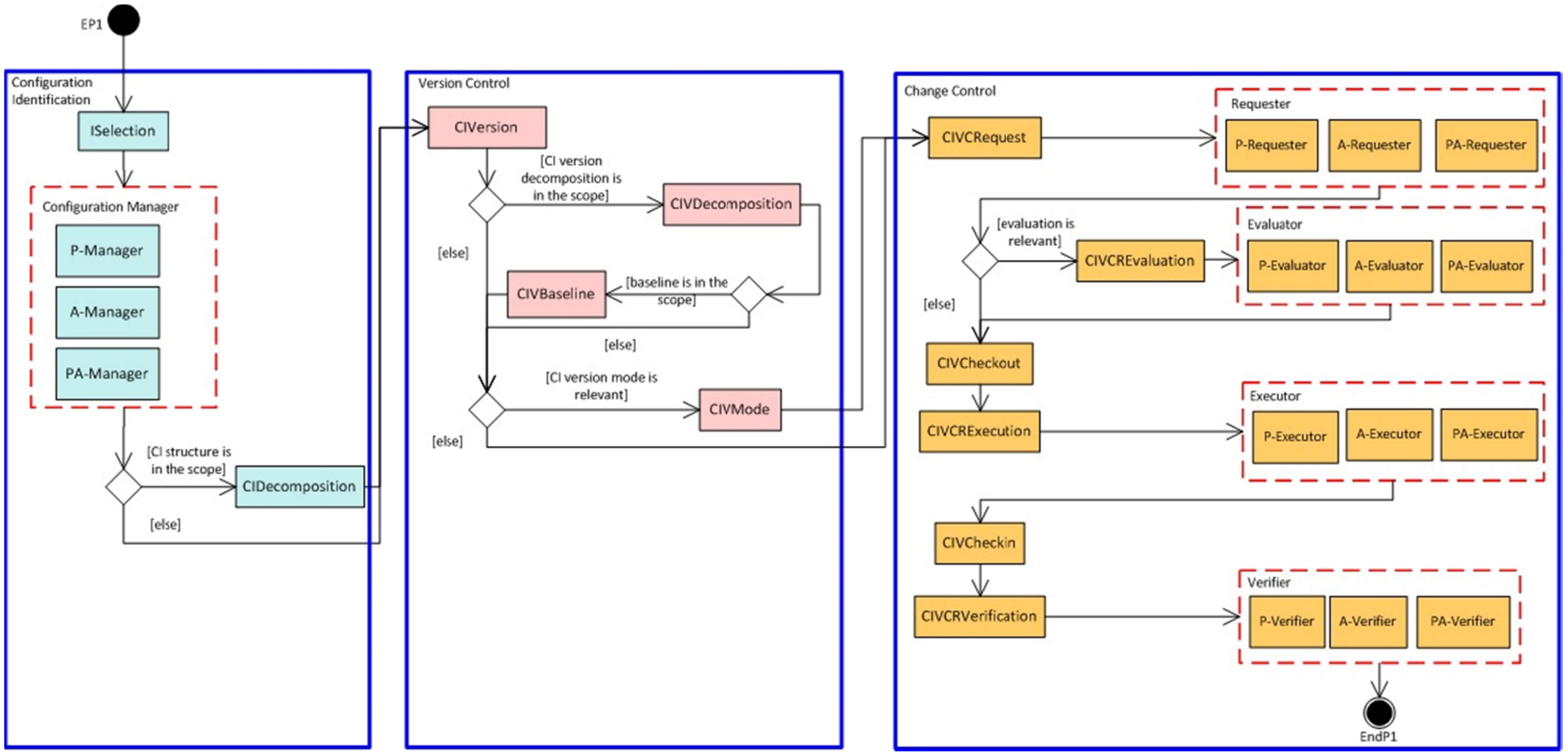

Figure 6 – CM-OPL Process (detailed view)

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 which configuration item that will be configured (*CIDecomposition*).

After, the OE needs to apply the *CIVersion* pattern. This pattern includes the relationship between Version and Configuration Item, since Version is a mode of a Configuration Item. Next, we have a pattern dealing with the decomposition (*CIVDecomposition*) of versions. This version can be atomic or complex, i.e., a Version composed of other Versions, and it characterizes Composite CI. For each CI that is part of a Composite CI, there must be a Version that is part of a Configuration. The next pattern addresses the baseline of the item (*CIVBaseline*). A baseline is a product configuration that was revised and designated to be a basis for future development [Calhau et al. 2012]. Also, there is the mode of the version (*CIVMode*), that is, a variant or revision of the configuration item. This is a complete and disjoint generalization set of Version.

After modeling the version control, the *CIVCRequest* pattern is used. This pattern models a change request that is submitted by the Requester. The Version mediates the change request and 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 (*CIVCREvaluation*). Following the process, the last version of the configuration item registered can be checked out to the computational agent/person to change (*CIVCheckout*). Thus, the Executor implements the modification modeling through the *CIVCRExecution* pattern. After the modification, s/he can do the checkin to register the new version (*CIVCheckin*). After registering the change, validation occurs. The pattern corresponding to the last configuration step (*CIVCRVerification*) presents the Verification relator mediating Verified Change and the Verifier. Finally, the process ends.

5 CM-OPL Pattern 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,* that is the role played by an *Item* when it is selected in a Configuration Selection. *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:

- ✓ *Which items have to be their configuration managed?*
- ✓ *Who is the Configuration Manager that selects each configuration item?*

Conceptual Model:

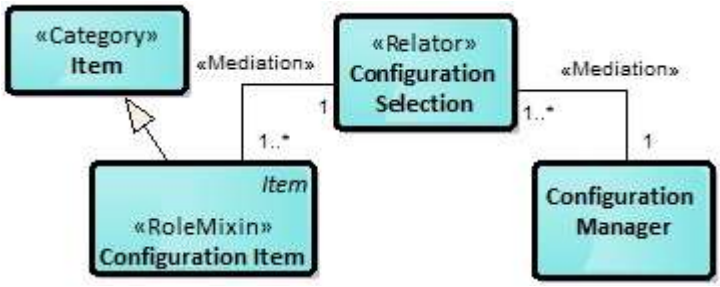

Figure 7 ISelection – Conceptual Model

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 and 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 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 a configuration 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:

- ✓ *Who can play the role of configuration manager?*

Conceptual Model:

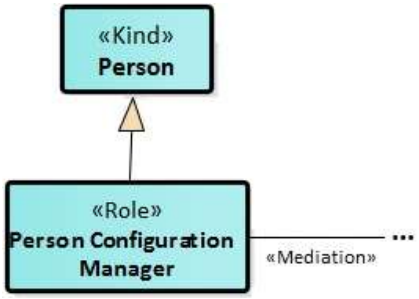

Figure 8 P-Manager – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

A-Manager – Computational Agent Configuration Manager

Name: Computational Agent Configuration Manager

Intent: Represents computational 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:

- ✓ *Who can play the role of configuration manager?*

Conceptual Model:

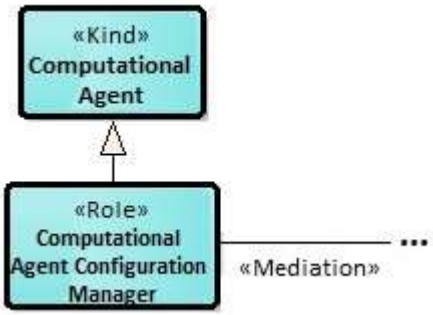

Figure 9 A-Manager – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

PA-Manager – Person/ Computational Agent Configuration Manager

Name: Person/ Computational Agent Configuration Manager

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

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

Competency Questions:

- ✓ *Who can play the role of configuration manager?*

Conceptual Model:

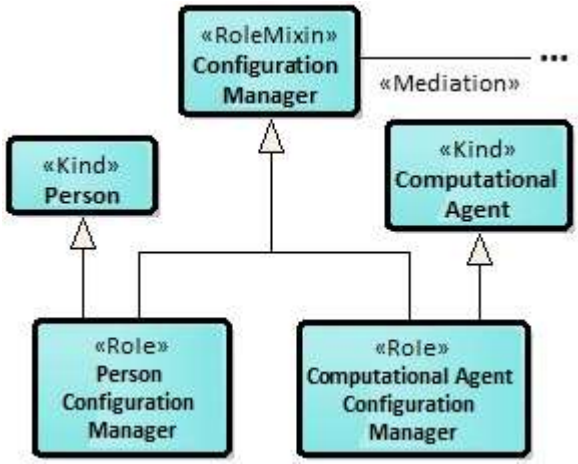

Figure 10 PA-Manager – Conceptual Model

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

CIDecomposition - Configuration Item Decomposition

Name: Configuration Item Decomposition

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

Rationale: when a *Configuration Item* is atomic, i. e. , it is not composed by other configuration items, it can specialize in a rolemixin called *AtomicCI*. It is classified as rolemixin because it is an antirigid type whose instantiation depends on a relational property (as a role of an *Item* Category). On the other hand, a *Configuration Item* can be composite (*Composite CI*). In this case, other configuration items compose a *Configuration Item* and 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:

- ✓ *How is a configuration item decomposed?*

Conceptual Model:

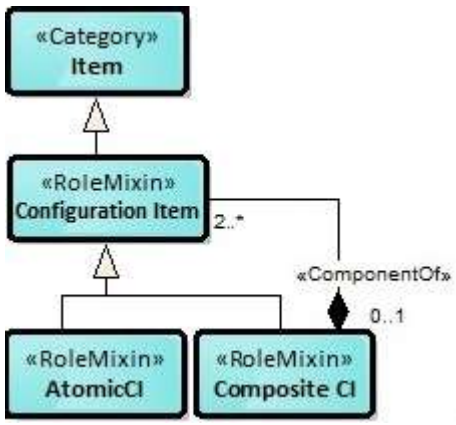

Figure 11 CIDecomposition – Conceptual Model

Axiomatization:

A1	∀ ci: ConfigurationItem, cci: CompositeCI (isA(cci, ci)) → (ComponentOf(ci,cci) ^ ∃cii: ConfigurationItem ^ ComponentOf(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.

5.2 Version Control Group

CIVersion – Configuration Item Version

Name: Configuration Item Version

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

Rationale: models the *Version* that is a mode of a *Configuration Item*.

Competency Questions:

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

Conceptual Model:

Figure 12 CIVersion – Conceptual Model

Axiomatization: -

FOPs Support: Mode Pattern.

Term Definitions:

Configuration Item	An item of product that has a configuration that can be managed.
Version	Represents a specific state of a *Configuration Item* at a given point in time of the product development.

CIVDecomposition – Configuration Item Version Decomposition

Name: Configuration Item Version Decomposition

Intent: Represents a decomposition of the configuration item version.

Rationale: If the *Configuration Item* is atomic means that it has *atomic versions* (*Atomic Version* mode) when it evolves. If the *Configuration Item* is composite, it means that it has composite *versions* and these versions are called *Configuration*. If there is a Configuration, the *Configuration 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:

- ✓ *How is an item version decomposed?*

Conceptual Model:

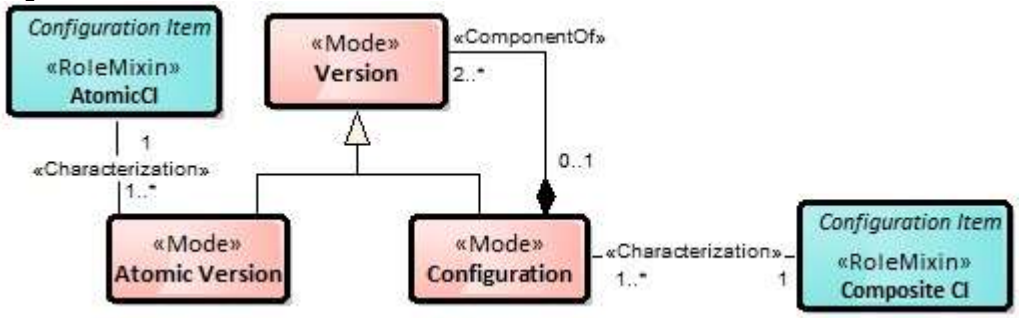

Figure 13 CIVDecomposition – Conceptual Model

Axiomatization: -

FOPs Support: Mode Pattern.

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*.
AtomicCI	Configuration Item that is not composed by another one.
Configuration	Set of physical and functional characteristics that describe the product at a given time. It is a version of the composite *Configuration Item*.
Composite CI	Configuration Item composed by other configuration items.

CIVBaseline – Configuration Item Version Baseline

Name: Configuration Item Version 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 *computational 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:

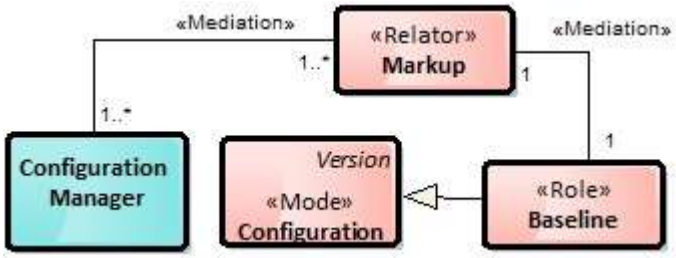

Figure 14 CIVBaseline – Conceptual Model

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 a configuration 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.

CIVMode – Configuration Item Version Mode

Name: Configuration Item Version Mode

Intent: Represents the mode (variant - parallel versions or revision – overwritten versions) of the configuration item version.

Rationale: A *Configuration Item* may have multiple *Versions*. *Versions* of configuration 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*. Also, *Versions* of configuration 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*. This type of generalization is complete and disjoint.

Competency Questions:

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

Conceptual Model:

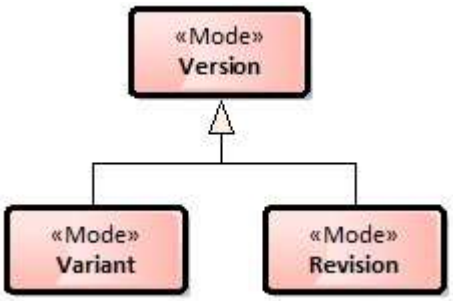

Figure 15 CIVMode – Conceptual Model

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 a configuration item with specific characteristics that differ from other versions.
Revision	A revised version of a configuration item that overlaps another (original) version.

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 can play the role of requester?*

Conceptual Model:

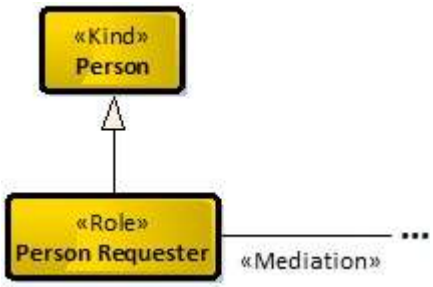

Figure 16 P-Requester – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

A-Requester - Computational Agent Requester

Name: Computational Agent Requester

Intent: Represents computational 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 can play the role of requester?*

Conceptual Model:

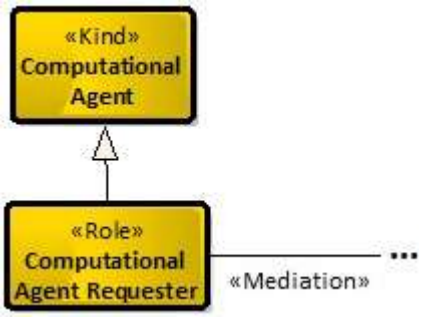

Figure 17 A-Requester – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

PA-Requester - Person/ Computational Agent Requester

Name: Person/ Computational Agent Requester

Intent: Represents persons and computational agents or machines as requesters.

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

Competency Questions:

- ✓ *Who can play the role of requester?*

Conceptual Model:

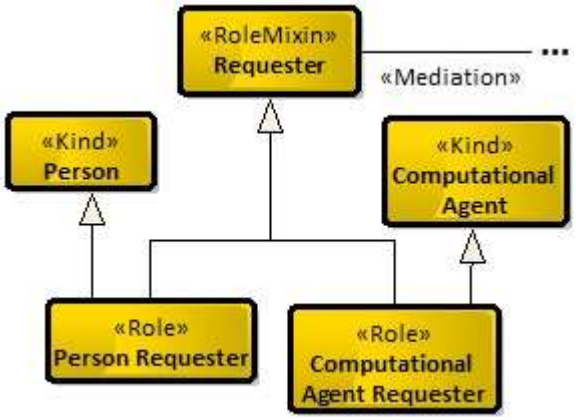

Figure 18 PA-Requester – Conceptual Model

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

CIVCRequest – Configuration Item Version Change Request

Name: Configuration Item Version Change 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 a *Configuration Item version*, the *Version is submitted for change*.

Competency Questions:

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

Conceptual Model:

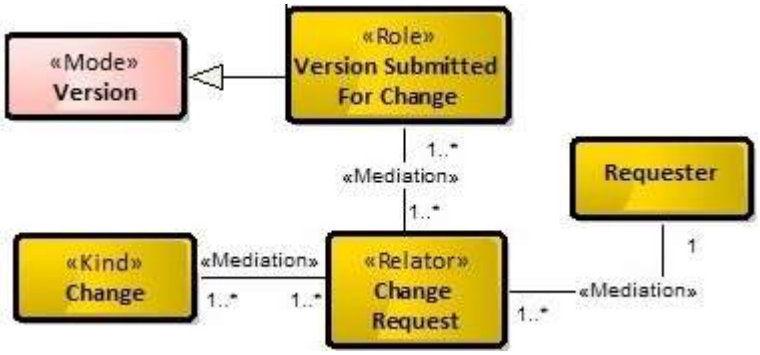

Figure 19 CIVCRequest – Conceptual Model

Axiomatization:

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

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

Term Definitions:

Requester	The role played by a *Person* or by *a Computational Agent* when they request a change of a configuration item version.
Change Request	Request for change by a *Requester* to change the configuration of a CI version.
Change	Specified modification to be performed on configuration items versions that may or not be implemented.
Version Submitted For Change	A version of the configuration 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.

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 can play the role of evaluator?*

Conceptual Model:

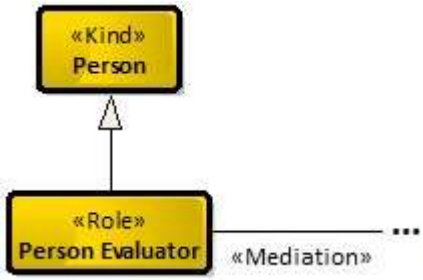

Figure 20 P-Evaluator – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

A-Evaluator - Computational Agent Evaluator

Name: Computational Agent Evaluator

Intent: Represents computational 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 can play the role of evaluator?*

Conceptual Model:

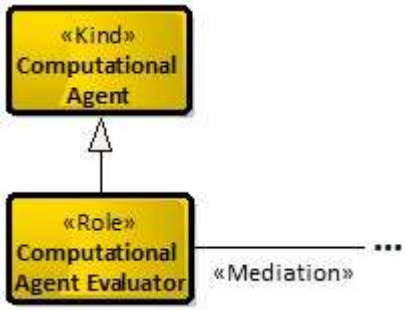

Figure 21 A-Evaluator – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

PA-Evaluator - Person/ Computational Agent Evaluator

Name: Person/ Computational Agent Evaluator

Intent: Represents persons and computational agents or machines as evaluators.

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

Competency Questions:

- ✓ *Who can play the role of evaluator?*

Conceptual Model:

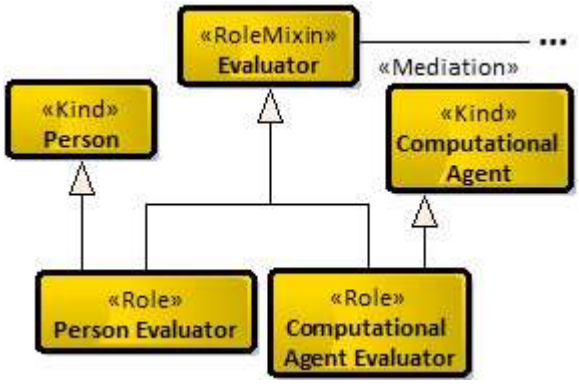

Figure 22 PA-Evaluator – Conceptual Model

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

CIVCREvaluation - Configuration Item Version Change Request Evaluation

Name: Configuration Item Version Change Request Evaluation

Intent: Represents the evaluation if the configuration item version can have the change 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 is the result of the evaluation of the change request?*

Conceptual Model:

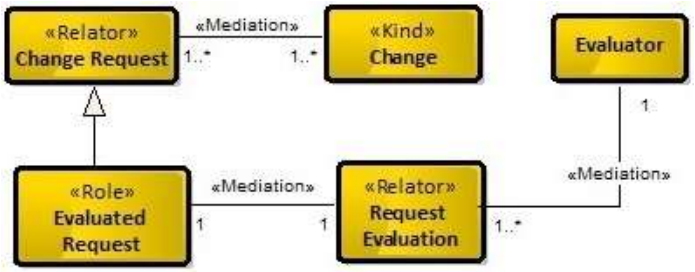

Figure 23 CIVCREvaluation – Conceptual Model

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 change of the configuration item version.
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 item version.
Change	Specified modification to be performed on configuration items versions that may or not be implemented.

CIVCheckout – Configuration Item Version Check-out

Name: Configuration Item Version Check-out

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

Rationale: when a *Version* of the *Configuration Item* needs to be modified, it may be prepared for modification, that is, it is checked-out before. When it occurs, the Version takes on the role of *Checked-Out Version* and the change takes on the role of *On Going Change*, that is, the *Change* that is in progress. A *Check-Out* mediates the relation between a *Version (Checked-Out Version)*, a *Change (On-Going Change*) and an *Executor* (responsible to check-out).

Competency Questions:

- ✓ *Which version of the configuration item does the person/computational agent wants to modify or check out?*
- ✓ *Who checked out the version to modify in the future?*
- ✓ *Which change is going to be performed on the item?*

Conceptual Model:

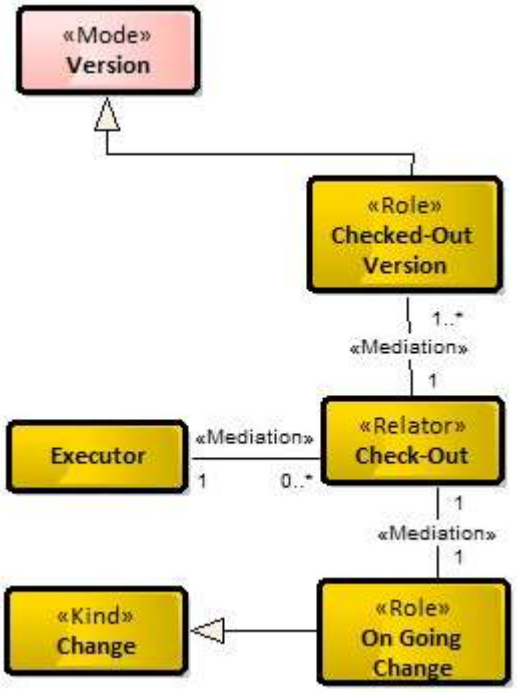

Figure 24 CIVCheckout – Conceptual Model

Axiomatization: -

FOPs Support: Relator Pattern – Variant 1 and Role 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 a configuration item to make a change.

Executor	The role played by a *Person*, *an Agent* or both when they execute a configuration change of a configuration item.
On-Going Change	Change a configuration item in progress.
Change	Record of the modification action of a configuration item version.

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 can play the role of executor?*

Conceptual Model:

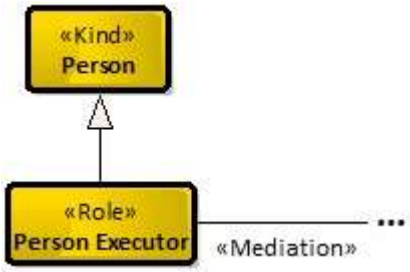

Figure 25 P-Executor – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

A-Executor - Computational Agent Executor

Name: Computational Agent Executor

Intent: Represents computational 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 (automatic) execution.

Competency Questions:

- ✓ *Who can play the role of executor?*

Conceptual Model:

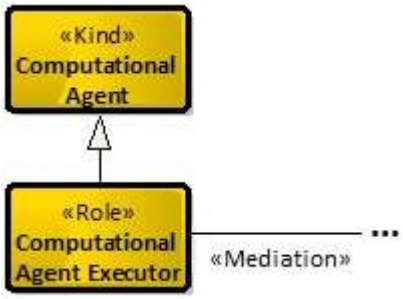

Figure 26 A-Executor – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

PA-Executor - Person/ Computational Agent Executor

Name: Person/ Computational Agent Executor

Intent: Represents persons and computational agents or machines as executors.

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

Competency Questions:

- ✓ *Who can play the role of executor?*

Conceptual Model:

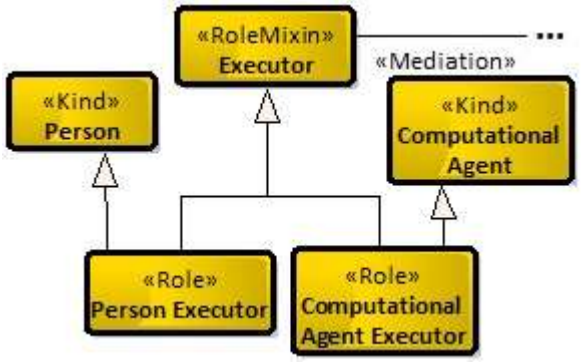

Figure 27 PA-Executor – Conceptual Model

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

CIVCRExecution – Configuration Item Version Change Request Execution

Name: Configuration Item Version Change Request Execution

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

Conceptual Model:

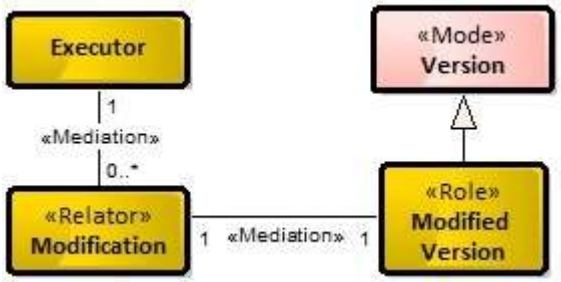

Figure 28 CIVCRExecution – Conceptual Model

Axiomatization: -

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

Term Definitions:

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

CIVCheckin – Configuration Item Version Check-in

Name: Configuration Item Version Check-in

Intent: Represents the register of the version of the modified configuration 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 CI version the person/computational agent wants to become current CI version?*
- ✓ *Who implemented the new CI version that will be checked-in?*

Conceptual Model:

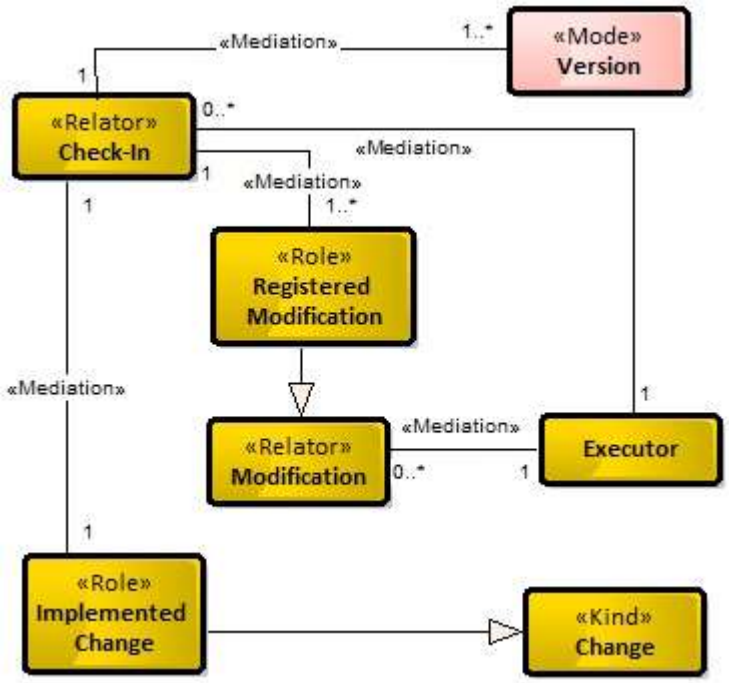

Figure 29 CIVCheckin – Conceptual Model

Axiomatization:

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

FOPs Support: Relator Pattern – Variant 1 and Role 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 configuration items versions.
Registered Modification	Records of the change.
Modification	Records the action of the change of a configuration item version.

Executor	The role played by a *Person*, *an Agent* or both when they execute a configuration change of a configuration item version.
Implemented Change	Specified change that has been implemented and recorded through a check-in.
Change	Specified modification to be performed on configuration items versions that may or may not be implemented.

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 can play the role of verifier?*

Conceptual Model:

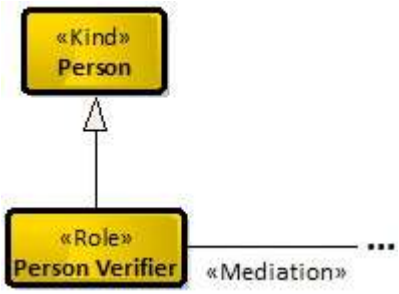

Figure 30 P-Verifier – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

A- Verifier – Computational Agent Verifier

Name: Computational Agent Verifier

Intent: Represents computational 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 can play the role of verifier?*

Conceptual Model:

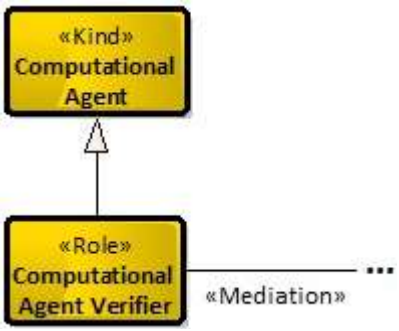

Figure 31 A-Verifier – Conceptual Model

Axiomatization: -

FOPs Support: Role Pattern.

Term Definitions:

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

PA- Verifier - Person/ Computational Agent Verifier

Name: Person/ Computational Agent Verifier

Intent: Represents persons and computational agents or machines as verifiers.

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

Competency Questions:

- ✓ *Who can play the role of verifier?*

Conceptual Model:

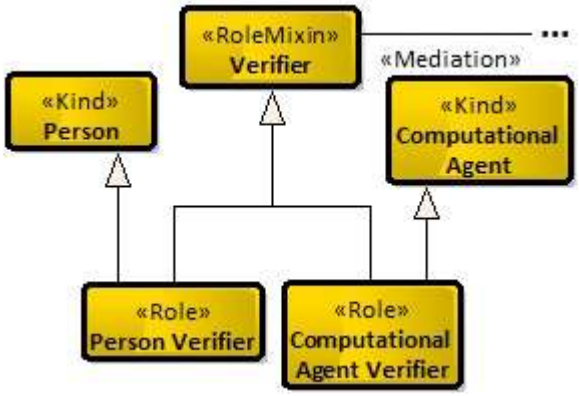

Figure 32PA-Verifier – Conceptual Model

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

CIVCRVerification - Configuration Item Version Change Request Verification

Name: Configuration Item Version Change Request Verification

Intent: Represents the verification of the configuration item version with the change 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. 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:

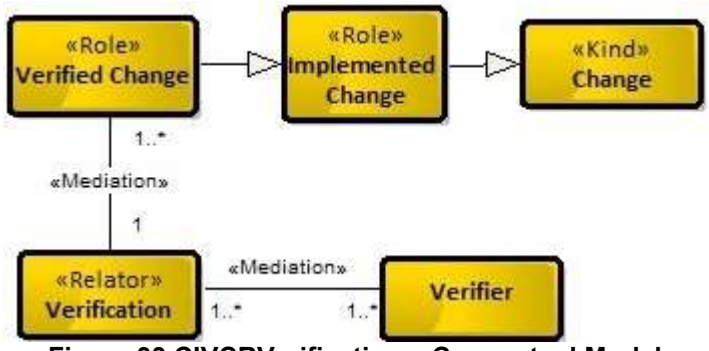

Figure 33 CIVCRVerification – Conceptual Model

Axiomatization: -

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

Term Definitions:

Verifier	The role played by a *Person* and *an Agent* when they validate a configuration change of a configuration item version.
Verification	Validates the configuration change of a configuration item version.
Verified Change	Records the verified change of a configuration item version.
Implemented Change	Records the implemented change of a configuration item version.
Change	Specified modification to be performed on configuration items versions 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.