# A Guide to SABiOx – the Extended Systematic Approach for Building Ontologies

Camila Zacché de Aguiar, Vítor E. Silva Souza
Ontology & Conceptual Modeling Research Group – NEMO
Federal University of Espírito Santo, Brazil

September 23, 2024

**Abstract**

This document is a guide for the application of the Ontology Engineering method SABiOx – the Extended Systematic Approach for Building Ontologies. It presents the entire method in detail and is intended to be used by ontology engineers that want to build reference and operational ontologies using a systematic approach.

# Contents

# Chapter 1

# Introduction

SABiOx is the Extended Systematic Approach for Building Ontologies. It builds from the initial proposals made in a PhD Thesis [Agu21], which in turn were based on the existing Ontology Engineering method SABiO, the Systematic Approach for Building Ontologies [Fal14]. SABiOx incorporates agile principles and provides more details on the activities that compose its process in order to help guide inexperienced ontology engineers.

This technical report is structured as follows: Section 2 provides an overview of the SABiOx method, describing its phases and activities. The following chapters, then, provide detailed descriptions for the activities of each phase, namely Requirements (Chapter 3), Setup (Chapter 4), Capture (Chapter 5), Design (Chapter 6) and Implementation (Chapter 7).

At the end of this document, in page 56, a Glossary section provides an index of SABiOx's phases and their activities.

# Chapter 2

# Overview

The life cycle proposed by the SABiOx method is composed of two complementary types of cycles: the **ontology life cycle** presents the dynamic aspect of the ontology building process, expressed through phases and iterations; the **phase life cycle** presents the dynamic aspect of the phases that make up the ontology life cycle, expressed through activities and iterations.

## 2.1 Ontology Life Cycle

The **ontology life cycle** is presented in Figure 2.1. It is formed by five phases that indicate the emphasis of the activities at each moment of the life cycle and the evolutionary nature of ontology construction, namely:

1. Requirements phase: elicits the requirements for the ontology, i.e., what is the ontology intended to capture and represent;
2. Setup phase: defines the baseline (e.g., modeling language, reuse of other ontologies, etc.) for building ontology models;
3. Capture phase: identifies and models the conceptualization (i.e., concepts, relations, axioms, etc.) to meet the elicited requirements;
4. Design phase: specifies technological features (e.g., encoding language, reuse of existing vocabularies, etc.) for the implementation of the ontology;
5. Implementation phase: encodes the ontology in an operational language (e.g., OWL).

At each cycle, i.e., at each passage through the five phases of the ontology life cycle, a new complete version of the ontology is built. The first ontology cycle is defined as the *development cycle* and subsequent ones as *evolution cycles*. The development cycle is triggered by the need to build an ontology and the

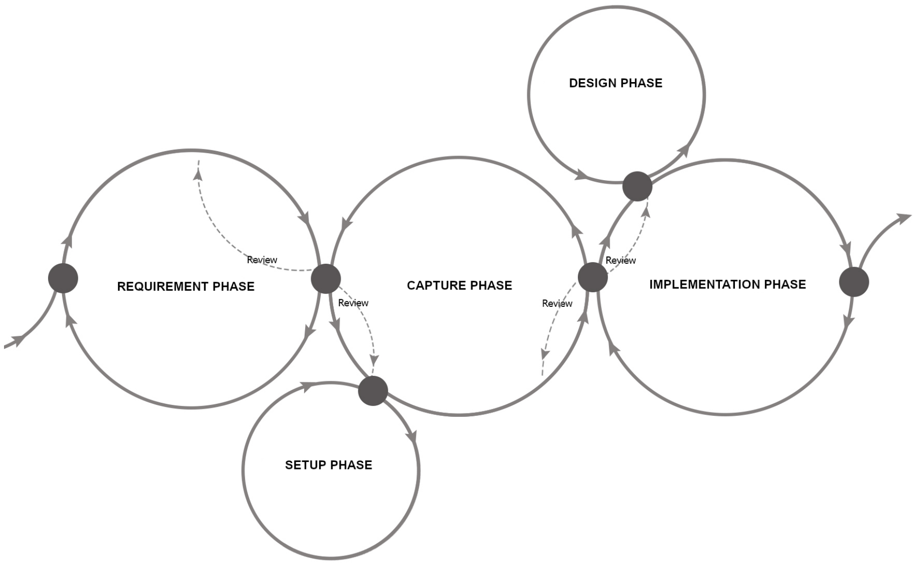


Figure 2.1: Overview of SABiOx's ontology life-cycle.

evolution cycles are triggered by changes or improvements in the already built ontology.

The phases are defined in a progressive way so that each phase uses the result of the previous phase to provide gradual enrichment in the life cycle. The result of the Requirements phase is a **specification document**, the Setup and Capture phases produce a **reference ontology** and the Design and Implementation phases result in an **operational ontology**.[1] Therefore, the Setup phase starts from the result of the Requirements phase and the Design phase starts from the result of the Capture phase.

Conversely, the outcome of each phase can be revised (represented by the dashed lines). For instance, the outcome of the Implementation phase can give rise to a revision in the Design and Capture phases. In turn, the outcome of the latter can give rise to a revision in the Setup and Requirements phases. In addition, the life cycle is designed to accommodate the construction of ontologies that evolve over time, i.e., evolution cycles generate new increments of the ontology.

## 2.2 Phase Life Cycles

Each one of SABiOx's phases defines a **phase life cycle**, consisting of **main activities**, which define the main procedures for producing the result of that

[1]For a detailed explanation on the difference between reference and operational ontologies, refer to [Gui07].

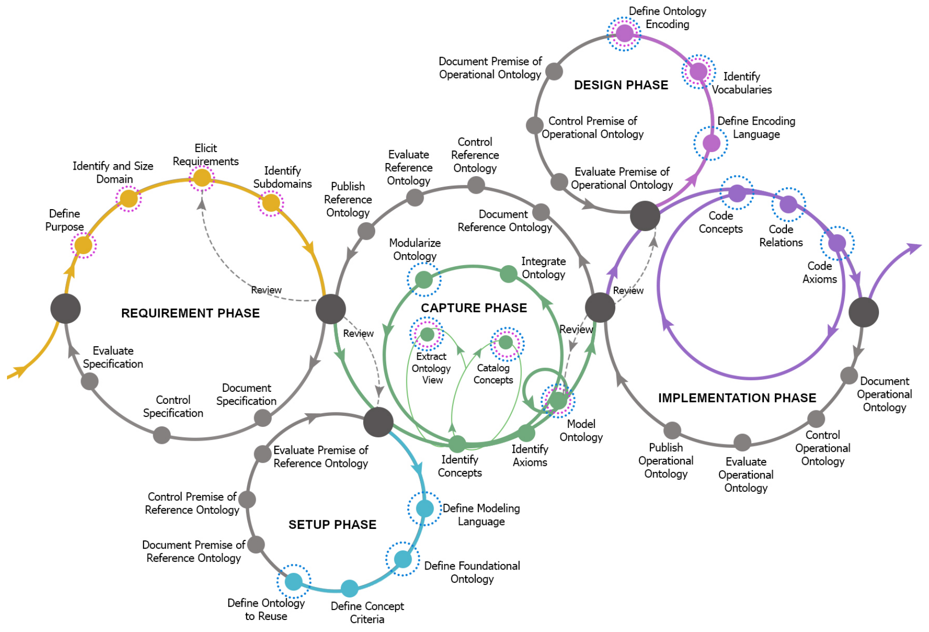


Figure 2.2: Overview of SABiOx' phases life cycles.

phase, and **supporting activities**, which define auxiliary procedures for the main activities.

Figure 2.2 illustrates the life cycle for all phases of the SABiOx method, with dots representing activities and lines representing the flow of each phase. Each phase is represented by a different color, whereas supporting activities are represented by gray dots/lines following the main activities (when performed in sequence) or colored dotted circles around main activities (when performed in parallel).

The goals of each phase of SABiOx have already been presented in Section 2.1. The supporting activities that complement the main activities of these phases are divided in six categories:

- Knowledge Acquisition: extracts knowledge from different sources;
- Documentation: records the results of the activities;
- Configuration Management: controls the versioning and changes in the produced artifacts;
- Evaluation: assesses the quality of the produced results;
- Reuse: reuses existing ontological and non-ontological resources;
- Publication: makes the produced ontology available.

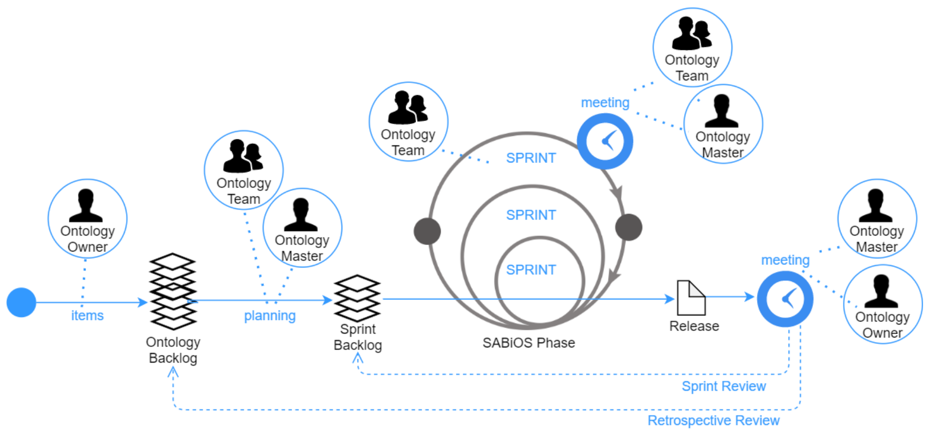


Figure 2.3: Application of Scrum principles on a SABiOx' phase life-cycle.

The supporting activities from categories Knowledge Acquisition (represented by the pink dotted circle) and Reuse (blue dotted circle) are performed simultaneously with the main activities they support.

## 2.3 Agile Processes and Roles

For each phase of the ontology life cycle there can be several phase life cycles, executed based on agile principles. SABiOx does not prescribe a specific agile process to be followed. Each organization is free to define its own process based on their own context, such as the number of people that will participate, their expertise, the complexity of the domain to be modeled, etc.

For instance, a given organization could choose to use principles from the Scrum framework [SS20] in order to conduct the process of building ontologies with SABiOx. Figure 2.3 illustrates how such an organization could perform the phases of SABiOx as Scrum sprints. Requirements and other necessary tasks are registered in an Ontology Backlog, which could be continuously updated and prioritized. A set of items is then moved from this backlog to a Sprint Backlog and stay stable during the execution of the sprint, in which ontology engineers go through the processes and activities defined for the respective phase in a predetermined period of time. Daily meetings can be held to track the work done, resolve obstacles, and align priorities for the day. At the end of each sprint, reviews and retrospectives ensure early feedback and allow accommodation of new requirements and improvements in the next iterations.

Although a specific agile process is not prescribed, SABiOx takes inspiration on Scrum [SS20] to at least define the roles involved in its phases and activities. Such roles are used in the detailed description of each activity in the following chapters and could be performed by the same person in an individual effort or by different people in a collective effort in the context of an organization. The roles involved in SABiOx activities are:

- **Ontology Owner**: a person interested in building the ontology for a given purpose. Responsible for defining what kind of ontology should be built and what are its requirements. Participates in the ontology building process in order to prioritize and define changes;
- **Ontology Team**: a person or group of people effectively responsible for building the ontology and working on tasks based on its requirements. Ideally, the Ontology Team should be multidisciplinary and preferably small, composed by one or more people with the following roles:
  - **Domain Experts**: a person with knowledge/expertise on the domain of the ontology;
  - **Ontology Engineers**: a person with knowledge/expertise on ontology design and implementation;

Roles can also be extended or adjusted in case a specific process is adopted by the organization. Going back to the Scrum example presented earlier, a new role could be included, based on the *Scrum Master* role:

- **Ontology Master**: a person with knowledge of the SABiOx method that acts as a facilitator for the construction of the ontology with the Ontology Team and a mediator between the team and the Ontology Owner.

## 2.4 Running Example

With the purpose of guiding the different roles involved in building the ontology, the following chapters provide detailed descriptions of each activity in order to enable their reproduction even by inexperienced ontology engineers.

To make such descriptions more concrete, a running example is used throughout the chapters: the development of OOC-O, an ontology on Object-Oriented Code [AFS19]. OOC-O aims to identify and represent the semantics of the entities present at compile time in object-oriented (OO) source code, with the intention to assist the understanding of different programming languages in this paradigm and to support the development of tools that work with these languages.

Chapter 3

# Requirements Phase

The Requirements Phase aims to identify the purpose of the ontology within a defined domain and to discover the needs that the ontology should satisfy regarding its content and characteristics. Its main activities have the aid of supporting activities of the following categories:

- Knowledge Acquisition (KNO): to extract knowledge through the use of *brainstorming*, interview, questionnaire, concept mapping and other elicitation techniques;
- Documentation (DOC): to register the acquired knowledge as an ontology specification;
- Configuration Management (MAN): to control the changes, versions and deliveries of the information recorded in the ontology specification;
- Evaluation (EVA): to validate whether the information extracted and registered in the ontology specification meets the expectations.

The activities that compose this phase are presented in the following sections. Each activity is represented by a 4-letter acronym with a 3-letter prefix indicating if it is a main activity of this phase (REQ) or a supporting activity, whose prefixes are indicated above.

## 3.1 Define Purpose (**REQ-PURP**)

Requirements activity that defines and describes the purpose of the ontology in order to answer: **what** is the domain that the ontology intends to represent, **for what** the ontology will be useful, and **why** the ontology should be built.

### Activity definition:

**Who:** *ontology engineer* as responsible and *ontology owner* as participant;

**What-Input:** need to build an ontology;

**What-Output:** answers to the questions what, for what, and why.

### Support activities in parallel:

**Capture Ontology Purpose (KNO-PURP)**
Knowledge Acquisition activity in which the *ontology engineer* extracts expectations from the *ontology owner* in order to answer the three questions (what, for what, and why) for the ontology. This knowledge is usually acquired through brainstorming and interview techniques.

### Running Example:

For OOC-O, the purpose of the ontology is: to represent the concepts of object orientation present in a source code (*what*); so that these concepts can be interpreted and identified by different programming languages in a unique way (*for what*); because each programming language defines its own syntax and semantics, and there is no standardization of source code concepts nor mature initiatives that meet these requirements, resulting in source code with heterogeneity and interoperability difficulties (*why*).

## 3.2 Identify and Size Domain (REQ-DOMN)

Requirements activity that identifies the domain that the ontology is intended to represent according to its purpose and sizes its boundaries. It should identify the knowledge as well as the level of detail that the ontology should cover of the domain. It can also explicitly indicate which details or parts should not be covered. Although the development of an ontology is primarily driven by the application-related scenarios of that ontology [SFGP12], the goal of this activity is not to list the application scenarios but rather to extract the knowledge domain embedded in those scenarios.

### Activity definition:

**Who:** *ontology engineer* as responsible and *ontology owner* as participant;

**What-Input:** purpose of the ontology defined in the activity Define Purpose (REQ-PURP);

**What-Output:** the domain to be represented in the ontology and its boundaries.

**Support activities in parallel:**

**Understanding Ontology Domain (KNO-DOMU)**

Knowledge Acquisition activity in which the *ontology engineer* aims to understand the domain related to the purpose of the ontology, considering the prior knowledge of the *ontology owner*, *domain expert* and sources from the literature on the domain.

**Size Ontology Domain (KNO-DOMS)**

Knowledge Acquisition activity in which the *ontology engineer* aims to size the domain boundaries to be represented in the ontology. This activity is challenging, because real-world domains are interconnected and it is not possible to define a clear boundary between them. Therefore, the prior knowledge of the *ontology owner* about the threshold dimensions of this domain must be considered.

Boundaries should be defined for *horizontal dimension* in order to limit the external areas that are part of the domain and *vertical dimension* in order to limit the level of detail that should be delved into in the domain.

**Running Example:**

For OOC-O, the domain is defined as object-oriented (OO) programming, a method of software implementation in which programs are organized as cooperative collections of objects, whose objects represent an instance of some class and whose classes are members of a hierarchy of classes linked by inheritance relationships. A class serves as a template from which objects can be created, containing attributes and methods. In order for the attributes and methods of a class to be used in the definition of a new class, inheritance is applied as a means of creating abstractions.

In the horizontal dimension, the domain is defined as concepts related to object-oriented software development; for the vertical dimension, the domain is limited in representing source code at compile time and does not cover the execution (runtime) perspective, such as objects and messages exchanged between objects.

## 3.3 Elicit Requirements (REQ-ELIC)

Requirements activity that elicits functional and non-functional requirements for the ontology.

For the *functional requirements*, SABiO [Fal14] suggests the use of informal competency questions, which are questions not expressed in formal language [GF95]. Such questions are represented by interrogative sentences and should be defined in a stratified manner, with higher-level questions requiring the solution of lower-level questions [GF95].

The questions serve as constraints on what the ontology can be, i.e., they restrict what is or is not relevant to the ontology. Furthermore, these questions are used in the future to evaluate the ontology's ontological commitment to check whether the ontology meets the requirements.

For the *non-functional requirements*, SABiO suggests ontology quality requirements (e.g., reasoning performance, availability, usability, maintainability); design requirements (e.g., implementation language definition, consensus and concepts, adherence to process model); and intended use requirements (e.g., data source selection, ontology grouping, etc.) [Fal14], i.e., requirements not related to the contents of the ontology.

## Activity definition:

**Who:** *ontology engineer* as responsible and *ontology owner* as participant;

**What-Input:** ontology domain identified in the activity Identify and Size Domain (REQ-DOMN);

**What-Output:** functional and non-functional requirements that the ontology must answer/satisfy.

## Support activities in parallel:

**Capture Ontology Requirements (KNO-REQI)**
Knowledge Acquisition activity in which the *ontology engineer* aims to discover the needs or knowledge that should be represented in the ontology. One should consider that the needs will hardly be exhausted and, therefore, the requirements must be discovered, analyzed, negotiated and updated continuously, that is, one must return to this activity whenever necessary. This knowledge is usually acquired through brainstorming, interview and concept mapping techniques.

**Negotiate Ontology Requirements (KNO-REQN)**
Knowledge Acquisition activity in which the *ontology engineer* aims to negotiate with the *ontology owner* which requirements should be considered for building the ontology. It should be considered that not all the initial needs of the *ontology owner* are relevant to the ontology domain.

**Running Example:**

For OOC-O, the following functional requirements (competency questions) are defined:

- What are the main elements of an OO source code?
- Which classes are present in an OO source code?
- Which elements make up a class?
- What inheritance is present in a given class? ;
- What methods are present in OO code ..;
- What elements make up a method ?
- What methods of a class ?
- What variables are present in an OO code ?
- Which variables compose a class?
- Which variables are part of a method?

Further, the following non-functional requirements are elicited:

- Be modular to facilitate reuse by other ontologies;
- Be incorporated into a network of ontologies to facilitate the reuse of other ontologies and, consequently, expand its own possibilities of reuse and association;
- Be defined from recognized knowledge sources in the literature;
- Be defined from consensus knowledge of the domain;
- Be grounded by a foundation ontology in order to reuse ontological commitments and axioms already consensually established.

## 3.4 Identify Subdomains (**REQ-SUBD**)

Requirements activity that identifies subdomains related to the competency questions defined for the domain, in order to facilitate the identification of parts of the domain present in the ontology purpose and modularize the ontology requirements.

Modularization in subdomains allows the targeted or distributed construction of the ontology, i.e., subdomains can be treated in parallel in later phases. To identify the subdomains, it is suggested to use pre-established categories in the domain or to create categories that reflect the high frequency terms present in the competency questions [SFGP12].

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** functional requirements identified in the activity Elicit Requirements (REQ-ELIC);

**What-Output:** subdomains of the domain to be represented in the ontology.

### Running Example:

For OOC-O, three subdomains are defined: *Core*, to represent the general object-oriented concepts; *Class*, to represent the class-related concepts; and *Class Member*, to represent the concepts related to class members (e.g., attributes and methods).

## 3.5 Document Specification (DOC-SPEC)

Documentation activity that documents the specification of the ontology in an Ontology Specification Document, which should include the requirements raised for the ontology, as well as the following considerations:

- **Purpose of the Ontology**: natural language text following the standard convention: *The purpose of the ontology is to represent* $\langle what \rangle$ $\langle what\ for \rangle$ $\langle why \rangle$;
- **Ontology Domain**: natural language descriptive text describing the domain to be represented in the ontology;
- **Ontology Dimension**: natural language descriptive text clearly highlighting the horizontal and vertical boundaries of the ontology;
- **Subdomains**: name of the subdomains identified in the domain from the competency questions;
- **Functional Requirements**: list of competency questions with a unique identifier for each requirement, grouped by subdomains;
- **Non-Functional Requirements**: descriptive text in natural language with a unique identifier for each item.

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** results from the main activities of this phase;

**What-Output:** ontology specification document.

## Running Example:

For OOC-O, a fragment of the Ontology Specification Document for the object-orientation domain is presented in Table 3.1.

Table 3.1: Ontology Specification Document

| | | | |
|---|---|---|---|
| **Document:** | Ontology Specification | **Version:** | v.01 |
| **Ontology:** | OOC-O – Object-Oriented Code | **Date:** | Sep 13, 2018 |

**Purpose of the Ontology:**
The purpose of the ontology is to represent the object-oriented concepts present in source code so that these concepts can be interpreted and identified in different programming languages in a unique way. This is motivated by the fact that each programming language defines its own syntax and semantics, and there is no standardization of source code concepts nor mature initiatives that meet these requirements, resulting in source code with heterogeneity and interoperability difficulties.

**Ontology Domain:**
Object-oriented programming (OO) is defined as a method of implementing software in which programs are organized as cooperative collections of objects, whose objects represent an instance of some class and whose classes are members of a hierarchy of classes linked by inheritance relationships. A class serves as a template from which objects can be created, containing attributes and methods. In order for the attributes and methods of a class to be used in the definition of a new class, inheritance is applied as a means of creating abstractions.

**Ontology Dimension:**
In the horizontal dimension, the domain is defined as concepts related to object-oriented software development; for the vertical dimension, the domain is limited in representing source code at compile time and does not cover the execution (runtime) perspective, such as objects and messages exchanged between objects.

**Functional Requirements:**

| **ID** | **Description** |
|---|---|
| **Subdomain:** Core | |
| RF01 | What are the main elements of an OO source code? |
| **Subdomain:** Class | |
| RF02 | What classes are present in an OO code? |
| RF03 | Which elements make up a class? |
| RF04 | What inheritance relations are associated with a given class? |
| **Subdomain:** Class Member | |
| RF05 | Which methods are present in an OO code? |
| RF06 | Which elements make up a method? |
| RF07 | Which methods are part of a class? |
| RF08 | Which variables are present in an OO code? |
| RF09 | Which variables are part of a class? |
| RF10 | Which variables are part of a method? |

**Non-Functional Requirements:**

| **ID** | **Description** |
|---|---|
| RNF01 | Be modular to facilitate reuse by other ontologies; |

| | |
|---|---|
| RNF02 | Be part of a network of ontologies to facilitate reuse by other ontologies and, consequently, expand its own possibilities of reuse and association; |
| RNF03 | Be defined from recognized knowledge sources in the literature; |
| RNF04 | Be defined from consensus knowledge of the domain; |
| RNF05 | Be grounded on a foundational ontology in order to reuse well-established ontological commitments and axioms. |

## 3.6 Control Specification (MAN-SPEC)

Configuration Management activity that controls the changes, versions and deliveries of the information present in the Ontology Specification Document, described in the activity Document Specification (DOC-SPEC).

Version control can be as simple as using the document header to register the version and its respective date, storing the history of all versions in case of need, to the use of version control systems such as, e.g., Git.[1]

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** functional requirements identified in the activity Elicit Requirements (REQ-ELIC);

**What-Output:** subdomains of the domain to be represented in the ontology.

**Running Example:**

For OOC-O, version and date information are included in the Ontology Specification Document header, in order to control the different versions of the document, as presented in Table 3.1.

## 3.7 Evaluate Specification (EVA-SPEC)

Evaluation activity that assesses whether the information extracted and registered in the Ontology Specification Document meets the expectations of the *ontology owner*. To do so, it is necessary to validate the registered information with the *ontology owner* and repeat the phase life cycle if necessary.

### Activity definition:

**Who:** *ontology owner* as responsible;

[1]`https://git-scm.com/`

**What-Input:** ontology specification document prepared in the activity Document Specification (DOC-SPEC);

**What-Output:** evaluation of the ontology specification document.

**Running Example:**

For OOC-O, meetings are held with the *ontology owner* in order to validate the ontology specification document, presented in Table 3.1.

# Chapter 4

# Setup Phase

The Setup Phase aims to define questions that will guide the elaboration of the reference ontology, that is, decisions that have an impact on the conceptual modeling tasks to follow. Its main activities have the aid of supporting activities of the following categories:

- Knowledge Acquisition (KNO): to extract knowledge from resources;
- Reuse (REU): to reuse recognized ontological and non-ontological resources;
- Documentation (DOC): to register this knowledge as part of the Reference Ontology Document;
- Configuration Management (MAN): to control the changes, versions and deliveries of the documented information;
- Evaluation (EVA): to verify that the questions defined for the reference ontology meet its purpose.

The activities that compose this phase are presented in the following sections. Each activity is represented by a 4-letter acronym with a 3-letter prefix indicating if it is a main activity of this phase (SET) or a supporting activity, whose prefixes are indicated above.

## 4.1 Define Modeling Language (**SET-LANG**)

Setup activity that defines the modeling language to be used in the construction of the reference ontology, i.e., the conceptual model. The choice of language directly influences the integration and reuse of the ontology under construction, since ontologies developed in different languages may require greater adaptation efforts.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** ontology specification document prepared in the activity Document Specification (DOC-SPEC);

**What-Output:** choice of modeling language to be adopted in the reference ontology.

### Support activities in parallel:

**Adopt Modeling Language (REU-LANG)**
Reuse activity that aims to adopt existing modeling languages for building the reference ontology. There are several languages to model, communicate and negotiate ontology concepts.

**Running Example:**

For OOC-O, the OntoUML language [GFB$^+$18] is adopted for modeling the reference ontology.

## 4.2 Define Foundational Ontology (SET-FOUN)

Setup activity that defines the foundational ontology to be adopted, which guides the modeling process of the ontology under construction and the views of the reused ontologies. To do this, the popularity, usability, and adherence of the foundational ontology for the defined domain must be considered, as well as the expertise of the ontology engineer in the chosen ontology.

SABiO highlights the importance of concepts and relationships being previously analyzed in light of a foundational ontology. Reuse of grounding ontologies can be done by specialization or analogy, when concepts and relations are not explicitly extended but rather implicitly used to derive the ontology under construction [Fal14].

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** ontology domain identified in activity Identify and Size Domain (REQ-DOMN) and modeling language defined in activity Define Modeling Language (SET-LANG);

**What-Output:** choice of foundational ontology and conceptual patterns to be applied to the ontology under construction.

### Support activities in parallel:

**Adopt Foundational Ontology (REU-FOUN)**
Reuse activity that aims to adopt an existing foundational ontology for the construction of the reference ontology.

**Adopt Ontology Patterns (REU-PATT)**
Reuse activity that aims to define the ontology patterns to be adopted in the ontology construction process. Such patterns can be classified as [FGGP13]:

- *Conceptual Pattern*: reusable fragments of grounding or domain reference ontologies;
- *Architectural Pattern*: patterns that describe how to organize the ontology in terms of subontologies or modules;
- *Design Pattern*: patterns related to reasoning and logical implementation constructs; and
- *Programming Pattern*: patterns related to the ontology implementation language.

**Running Example:**

For OOC-O, the Unified Foundational Ontology (UFO) [GBBF+22] and its patterns are chosen for modeling the reference ontology.

## 4.3 Define Concept Criteria (SET-CRIT)

Setup activity that defines the criteria that will be adopted to identify if a concept is adherent to the domain of the ontology under construction. For this, a list of objective criteria, with quantitative information, or subjective criteria, with qualitative information, is defined. These criteria can be defined for individual analysis of each knowledge source or joint analysis over all the cataloged sources.

Given the depth of domain knowledge at this point, defining criteria can be challenging. Thus, as knowledge is enriched and decisions about the domain are made, one should return to this activity to update these criteria.

### Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* as participant. Alternatively, to optimize the process, this activity can be conducted unilaterally by the *domain expert*;

**What-Input:** ontology specification document prepared in the activity Document Specification (DOC-SPEC);

**What-Output:** criteria defined for the domain concepts.

**Running Example:**

For OOC-O, the following criteria define whether a concept is adherent to the object-orientation domain of the ontology under construction: the concept refers to programming constructs used at compile-time; the concept is present in more than 50% of the programming languages analyzed; and the concept is related to the main elements of object-orientation: class, method and attribute.

## 4.4 Define Ontologies to Reuse (SET-REUS)

Setup activity that defines the reference ontologies to be reused, i.e., ontologies or ontology fragments that match the purpose of the ontology under construction.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** ontology specification document prepared in the activity Document Specification (DOC-SPEC);

**What-Output:** ontologies to be reused in the ontology under construction.

### Support activities in parallel:

**Adopt Reused Ontology (REU-ONTO)**

Reuse activity that aims to adopt core and domain ontologies related to the purpose of the ontology under construction, regardless of its ontology grounding level. The reuse of ontologies is a challenge, since it involves heterogeneity of formalism, diversity of languages, lexical and semantic problems, implicit representation assumptions, loss of consensus knowledge, lack of documentation, and others [FLGP02].

The reuse of a core ontology will position the ontology under construction within a broader domain, while that of a domain ontology will provide concepts already formalized for its domain. To do this, one must search for candidate ontologies in search engines, repositories, and known ontologies.

To select core ontologies, one must analyze whether the candidate ontology represents appropriate concepts to anchor the ontology under construction. To select domain ontologies, one should analyze the coverage of the candidate ontology with respect to the requirements of the ontology under construction.

Further, one must consider: (i) if any ontology analysis was applied to the candidate ontologies; (ii) if the foundational ontology adopted by the candidate ontology is adherent to the ontology under construction; and (iii) if the domain perspective represented in the candidate ontology corresponds to that of the ontology under construction. The candidate ontology may refer to an individual ontology or to a network of ontologies.

**Running Example:**

For OOC-O, existing ontologies that deal with source code and object-orientation are searched in search engines and known ontologies. Each ontology found is analyzed according to its purpose and adherence to the domain of the ontology under construction, namely:

- SWO: software ontology that represents software artifacts of a complex nature, including systems, programs, and software code. It is identified that an OO code can be represented in the SWO ontology as a source code specification that makes up a piece of software;
- SPO: software process ontology that describes processes and activities using process assets, such as, e.g., Artifact. It is identified that an OO code can be represented in the SPO ontology as a software artifact;
- SCO: source code ontology that describes the general concepts of a source code, independent of programming language. It is identified that an OO code can be represented in the SCO ontology as a specialization;
- JAVAOWL: Java programming language ontology that describes the main elements of the language captured from the JavaElement class. It is identified that the JAVAOWL ontology only represents the structure of the Java language and is not adherent to the purpose of the ontology under construction;
- O3: ontology that represents the concepts of the object-oriented and programming paradigm. It is identified that the O3 ontology represents a different perspective of the domain of the ontology under construction.

Of the analyzed ontologies, the SCO ontology was selected in order to specialize the source code domain to the object orientation domain, as well as position the source code domain within the software process domain, given that SCO specializes SPO and SWO. The other ontologies are disregarded, since they do not present the same perspective as the source code domain of the ontology under construction.

## 4.5 Document Premise of Reference Ontology (DOC-REFE)

Documentation activity that documents the assumptions of the reference ontology in the Reference Ontology Document, which must include the questions previously defined in this phase, namely:

- Modeling Language: description of the modeling language;
- Foundational Ontology: description of the foundational ontology and its ontology patterns;
- Ontologies to Reuse: presentation of the core and/or domain ontologies to be reused in the ontology under construction.

### Activity definition:

**Who:** *ontology engineer* as responsible;
**What-Input:** results from the activities performed earlier in this phase;
**What-Output:** Reference Ontology Document with its respective premises.

### Running Example:

In OOC-O, the Reference Ontology Document is prepared with the premises established for the object-oriented reference ontology, as presented in Table 4.1.

Table 4.1: Ontology Reference Document

| | | | | |
|---|---|---|---|---|
| **Document:** | Reference Ontology | | **Version:** | v.01 |
| **Ontology:** | OOC-O Object-Oriented Code Ontology | | **Date:** | Jan 11, 2019 |
| **Modeling Language:** | | | | |
| OntoUML | | | | |
| **Foundational Ontology:** | | | | |
| UFO: Ontology characterized in modal logic and cognitive psychology that aims to provide a higher semantic level about the world in a conceptual model of a given knowledge domain, systematizes issues such as notions of types and their instances; objects, and their intrinsic properties; the relationship between identity and classification; distinctions between types and their relationships; part-whole relationships, among others. | | | | |
| **Criteria:** | | | | |
| CR01 | Concept refers to programming constructs used at compile-time; | | | |
| CR02 | Concept is present in more than 50% of the programming languages analyzed; | | | |
| CR03 | Concept related to the main elements of object-orientation: class, method and attribute. | | | |
| **Ontologies to Reuse:** | | | | |
| **Prefix** | **Definition** | **Foundation** | **Type** | **Analysis** |
| SCO | Source code Ontology that describes the general concepts of a source code, independent of programming language. | UFO | Core | OO code is a specialization of source code. |

## 4.6 Control Premise of Reference Ontology (MAN-REFE)

Configuration Management activity that controls the changes, versions and deliveries of the information registered in the Reference Ontology Document, described in the activity Document Premise of Reference Ontology (DOC-REFE).

It is suggested that the same version control mechanism adopted in activity Control Specification (MAN-SPEC) continues to be used here.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in activity Document Premise of Reference Ontology (DOC-REFE);

**What-Output:** control of the Reference Ontology Document.

**Running Example:**

For OOC-O, version and date headers int the Reference Ontology Document are adopted to control its versions, as presented in Table 4.1, following the same choice made earlier in activity Control Specification (MAN-SPEC).

## 4.7 Evaluate Premise of Reference Ontology (EVA-REFE)

Evaluation activity that assesses whether the information registered in the Reference Ontology Document meets the expectations of the *ontology owner* and is in accordance with the *domain expert*. This is done by validating the registered information with the *ontology owner* and returning to the beginning of the phase whenever necessary.

### Activity definition:

**Who:** *ontology owner* and *domain expert* as responsible;

**What-Input:** Reference Ontology Document prepared in activity Document Premise of Reference Ontology (DOC-REFE);

**What-Output:** evaluation of the Reference Ontology Document.

**Running Example:**

In OOC-O, meetings are held with the *ontology owner* and the *domain expert* in order to validate the Reference Ontology Document, presented in Table 4.1.

# Chapter 5

# Capture Phase

The Capture Phase aims to represent in a well-founded way the domain conceptualization based on the specification elaborated in the Requirements Phase and the questions defined in the Setup Phase, which should be revised at each iteration in order to add, detail, and delete requirements and questions related to the domain needs. In this context, conceptualization is an intentional semantic framework that encodes the implicit rules that constrain the structure of a part of reality [UK95].

The main activities of this phase have the aid of supporting activities of the following categories:

- Knowledge Acquisition (KNO): to extract the knowledge from the domain;
- Reuse (REU): to reuse recognized ontological and non-ontological resources;
- Documentation (DOC): to register this knowledge as a conceptual model and part of the Reference Ontology Document;
- Configuration Management (MAN): to control the changes, versions and deliveries of the documented information;
- Evaluation (EVA): to verify if the ontology is being built according to the requested requirements and to validate if it meets its purpose;
- Publication (PUB): to make the reference ontology available.

Note that a single ontology specification can give rise to distributed processes in the reference phase, according to the subdomains defined in the activity Identify Subdomains (REQ-SUBD).

The activities that compose this phase are presented in the following sections. Each activity is represented by a 4-letter acronym with a 3-letter prefix indicating if it is a main activity of this phase (CAP) or a supporting activity, whose prefixes are indicated above.

## 5.1 Identify Concepts (CAP-CONC)

Capture activity that identifies which domain concepts are part of the purpose of the ontology according to the knowledge sources. This activity treats *concept* as a conception or idea of a reality and therefore types, properties and examples are not identified as concepts, but as part of the description of concepts.

For this, the following sub-activities are defined:

### 5.1.1 Catalog Concepts (CAP-CATA)

Capture activity that catalogs the concepts related to the ontology domain, according to the criteria defined in the activity Define Concept Criteria (SET-CRIT).

The concepts can be cataloged using three different knowledge sources: data source, *domain expert*, and *ontology owner*. A data source can be a book, a standard, a specification, an article, or another source recognized by the domain-related communities. These concepts form a simple list of terms and meanings extracted from the different sources, i.e., containing conflicts, overlaps and synonyms.

Although this activity is guided by criteria, it remains subjective and dependent on the analysis of the roles involved. Only criteria defined from the individual analysis of knowledge sources are applied in this activity. Criteria that require joint analysis of knowledge sources will be applied in the activity Model Ontology (CAP-MODE).

It is worth noting that the results of this activity are important to optimize activity Model Ontology (CAP-MODE), since domain knowledge can be identified by different roles involved so that basic domain instructions are identified and organized.

### Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* and/or *ontology owner* as participants. To optimize the process, this activity can be conducted in a distributed way across roles, so that concept capture can be conducted by the *ontology engineer* and/or *domain expert* and/or *ontology owner*;

**What-Input:** ontology specification document prepared in the activity Document Specification (DOC-SPEC);

**What-Output:** domain concepts extracted from the knowledge sources.

### Support activities in parallel:

**Capture Concepts (KNO-CONC)**

Knowledge Acquisition activity that aims to capture the concepts related to the ontology domain. To capture concepts from the *ontology owner* and the *domain expert*, we suggest the application of brainstorming, interview, form and concept mapping techniques to extract the concepts with more evidence in the domain. To capture concepts from data sources, it is suggested to search for data sources in search engines and data repositories in order to apply text analysis to extract the concepts with greater evidence in the domain. We emphasize that, in this activity, it is important to capture all the concepts related to the domain without worrying about their overlaps or relationships at this point.

**Reuse Data Source (REU-DATA)**
Reuse activity that aims to adopt data sources related to the ontology domain for capturing concepts. Good engineering practice is not to "invent" descriptions, but to import them from authoritative sources [NM16].

**Running Example:**

For OOC-O, books and specification documents of programming languages that apply object orientation were defined as knowledge source and used for capturing the concepts, according to the established criteria (as defined in activity Define Concept Criteria (SET-CRIT)). Since there are a large number of programming languages related to the domain, the knowledge source search was targeted at Eiffel, Smalltalk, Java, C++ and Python programming languages. Table 5.1 presents a fragment of the concept catalog developed for the object-orientation domain.

Table 5.1: Fragment of the Concept Catalog for OOC-O

| | | |
|---|---|---|
| **Source:** | Smalltalk and Object Orientation [Hun12] | |
| **Concept** | **Definition** | **Instance** |
| Class | Basic building block that acts as a template for constructing instances. | `Object subclass: #Polygon` |
| Instance Variable | Variable that holds its own value for each instance of the class. | `instanceVariableNames: 'side'` |
| **Source:** | Eiffel: Analysis, Design and Programming Language [Eif06] | |
| **Concept** | **Definition** | **Instance** |
| Class | Implementation of an abstract data type intended to be the modular unit of software decomposition and to provide the basis for the type system. | `class Polygon end` |
| Attribute | Feature type that will be associated with each instance of the class. | `side : INTEGER` |
| **Source:** | Object-Oriented Programming in C++ [Laf97] | |
| **Concept** | **Definition** | **Instance** |
| Class | Class serves as a blueprint that specifies which data and which functions will compose the objects of that class. | `class Polygon{ };` |
| Data Member | Data item included within a class. | `private: int side;` |

| **Source:** | Java Language Specification [GJS+18] | |
|---|---|---|
| **Concept** | **Definition** | **Instance** |
| Class | Defines new reference types and describes how they are implemented. | `public class Polygon{ }` |
| Instance Variable | Variable defined for each instance of the class. | `private int side;` |

| **Source:** | Python 3 Object Oriented Programming [Phi10] | |
|---|---|---|
| **Concept** | **Definition** | **Instance** |
| Class | A class is like a blueprint for creating objects. | `class Polygon:` |
| Data Attribute | Attribute that defines a data value for each instance. | `side = None` |

**References:**

[Eif06] EIFFEL, E. Analysis, design and programming language. ECMA Standard ECMA-367, ECMA, 2006.

[GJS+18] GOSLING, J. et al. The Java language specification: Java SE 10 edition, 20 February 2018. 2018.

[Hun12] HUNT, J. Smalltalk and object orientation: an introduction. Springer Science & Business Media, 2012.

[Laf97] LAFORE, R. Object-oriented programming in C++. Pearson Education, 1997.

[Phi10] PHILLIPS, D. Python 3 object oriented programming. Packt Publishing Ltd, 2010.

### 5.1.2 Extract Ontology View (CAP-VIEW)

Capture activity that extracts the view of the reference ontologies to be reused in the modeling of the ontology under construction.

For the definition of the reused ontology, a modularization strategy should be used over the original ontology, since a view of the ontology will be developed that will reflect a part or a set of concepts extracted from it. As the view is being built, ontological analysis should be applied to ensure compatibility with the ontology under construction. Thus, the view of the reused ontology should be built following some guidelines:

1. If the reused ontology completely matches the purpose of the ontology under construction, the completely reused ontology should compose the view;
2. if only a part of the reused ontology corresponds to the purpose of the ontology under construction, the reused ontology must be partitioned and only this part must compose the view; and
3. if only some concepts of the reused ontology correspond to the purpose of the ontology under construction, the relevant concepts of the reused ontology should be identified so that their relations can be recursively visited in order to gather the concepts to compose the view.

It is worth noting that this activity is more related to modularization and application of ontology analysis than ontology reengineering, which is the process of retrieving and mapping a conceptual model from an implemented ontology to another, more suitable conceptual model that is reimplemented [GPRA99].

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** ontologies to be reused in the ontology under construction, identified in the activity Define Ontologies to Reuse (SET-REUS);

**What-Output:** view of the ontologies to be reused in the ontology under construction.

## Support activities in parallel:

**Capture View Concepts (KNO-VIEW)**
Knowledge Acquisition activity that aims to identify the ontology concepts that will form the view of the reused ontology as part of the ontology under construction. For reused core ontologies, one should identify the concepts that can anchor the concepts of the ontology under construction. For reused domain ontologies, one should identify the concepts that answer the competency questions of the ontology under construction.

## Running Example:

For OOC-O, the view of the SCO ontology was elaborated in order to represent only the concepts from the object-orientation domain, presented in Figure 5.1.

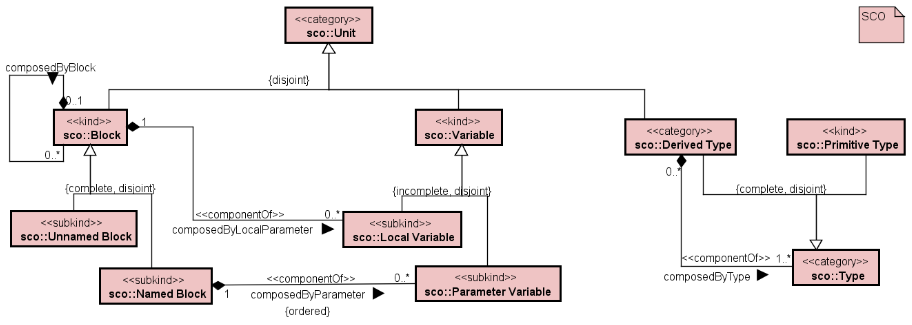


Figure 5.1: View of the SCO ontology to be reused.

## 5.2 Identify Axioms (CAP-AXIM)

Capture activity that identifies the constraint and inference axioms that the conceptual model of the ontology under construction must consider. You should identify the constraints that the conceptual model cannot represent and record them either as informal axioms (i.e., in natural language) or using some formalism (e.g., first-order logic, Object Constraint Language – OCL). As the conceptual model is developed, one should return to this activity to ensure a more complete and unambiguous model.

### Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* as participant;

**What-Input:** concepts identified in the activity Catalog Concepts (CAP-CATA);

**What-Output:** informal axioms of the ontology under construction.

### Running Example:

For OOC-O, axioms were defined to establish model constraints, such as the inheritance relationship, which is established when a subclass is inherited from a superclass, by stating the following axiom: $\forall c_1, c_2 : Class, i : Inheritance, inheritsIn(c_1, i) \land inheritedFrom(c_2, i) \rightarrow subClassOf(c_1, c_2)$

## 5.3 Model Ontology (**CAP-MODE**)

Capture activity that models concepts and relationships covered by the purpose of the ontology in a technology-independent ontology representation language. This modeling should apply both ontological analysis, in order to categorize concepts according to a foundational ontology, and conceptual ontology patterns, in order to leverage fragments of foundational ontologies. This representation of knowledge in the form of a conceptual model establishes ontological commitments about the domain, since one must represent part of the information about the world and ignore other parts [GDD06].

Ontology modeling can follow: (i) a top-down approach, starting with the most general domain concepts and then specializing them; (ii) a bottom-up approach, starting with the most domain-specific concepts and then grouping them into more general concepts; or (iii) a middle-out approach, starting with the most relevant concepts and then generalizing and specializing them by combining the top-down and bottom-up approaches. The middle-out approach is recommended, as it can decrease rework and modeling effort by finding balance in the levels of detail, starting with the most relevant concepts and, only when necessary, specializing or generalizing them [FLGP02].

The ontology concepts and relationships should be modeled using the modeling language defined in activity Define Modeling Language (SET-LANG). For each modeled concept, you must define its category according to the underlying ontology defined in the activity Define Foundational Ontology (SET-FOUN). For each relationship established, you must define its name, direction, type and cardinality. In addition, as the model is being built, conceptual ontology standards must be applied to ensure the standard quality of the model. This activity requires decision making about the domain and its representation, and should consider the knowledge acquired from the knowledge sources.

This activity can be performed in a directed or distributed manner according to the subdomains identified in activity Identify Subdomains (REQ-SUBD). If performed in a distributed manner, this activity will generate views of the ontology under construction that must later be integrated.

One must consider that modeling is evolutionary and, therefore, the model must continuously undergo adjustments, apply ontology analysis, integrate ontology views, and reorganize its modules, i.e., one must return to this activity whenever necessary.

## Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* as participant;

**What-Input:** concepts identified in the activity Identify Concepts (CAP-CONC) from the knowledge sources and the reused ontologies;

**What-Output:** conceptual model of the ontology under construction.

## Support activities in parallel:

**Concepts Consensus (KNO-CONS)**

Knowledge Acquisition activity that aims to establish semantic consensus on the concepts of the domain of the ontology under construction, in order to define their conceptualization and clarify doubts, ambiguities and conflicts. This activity is interested in mitigating the semantic differences of the knowledge sources, regardless of lexical differences. Thus, for each concept cataloged for the domain, one must exchange information, argument different perceptions, and arrive at a shared conceptualization.

Furthermore, the criteria for the set of knowledge sources, defined in the Define Concept Criteria (SET-CRIT) activity, must be applied. To do this, catalog concepts from different sources are validated against the criteria and discussed with domain experts. The consensus can be created from a synthesis of the knowledge sources or, according to the purpose of the ontology and the experts' discussion, favoring one source or another.

**Concepts Terminology (KNO-TERM)**

Knowledge Acquisition activity that aims to define the representative term for each consensus concept defined in the Concepts Consensus (KNO-CONS) activity. *Term* is defined as the representation of the domain concept in the ontology.

This is done by revisiting the cataloged concepts, considering: (i) if the term is consensual among the different sources, then this should be defined as the representative term of the concept in the ontology; (ii) if the term is not consensual, then the representative term of the ontology concept must be subjectively defined according to the greatest relevance for the domain.

**Running Example:**

For OOC-O, the conceptual model of the object-oriented domain has been built, and a fragment of it is presented in Figure 5.2.

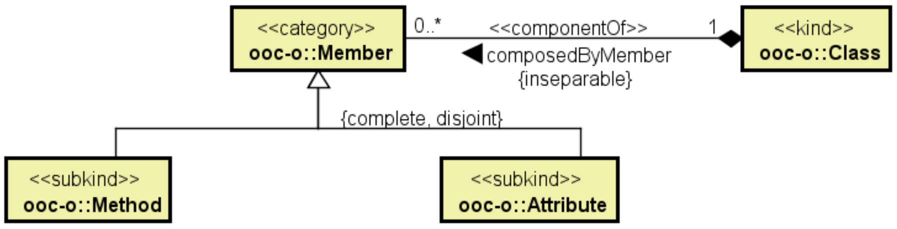


Figure 5.2: Conceptual model of the main object-oriented concepts in OOC-O.

## 5.4 Integrate Ontology (CAP-INTE)

Capture activity that integrates both the views of the ontology under construction and the views of the reused ontologies. If the ontology is built in a distributed fashion according to the identified subdomains, each subdomain will give rise to a view of the ontology. If the ontology is built by reusing existing ontologies, each existing ontology will give rise to one or more views of the ontology. Such views must be integrated into the ontology under construction and therefore this activity is performed iteratively with the Model Ontology (CAP-MODE) activity, in order to elaborate a more cohesive and complete model.

For views built from foundational ontologies, the integration can be done through specialization (concepts are explicitly extended) or analogy (concepts are implicitly used to derive the ontology under construction) [Fal14], so that the concepts of the reused ontology underlie the concepts represented in the ontology under construction. Furthermore, the integration with the grounding ontology makes the ontology commitment explicit and produces a uniform conceptual model to integrate different views of ontologies.

For views built from core ontologies, integration can be done primarily through specialization, so that concepts and relations from the core ontology are extended to represent more domain-specific conceptualizations of the ontology under construction [Fal14].

For views built from domain ontologies, integration can be based on the idea of composition [MW04], via the join operation. In this case, an ontology $O$ is formed by merging the concepts of the ontology under construction $O_1$ and the view of the reused ontology $O_2$ by merging the common concepts. Furthermore, association, extension and specialization relations between the entities of $O_1$ and $O_2$ may be necessary to establish the union of the ontologies. For the ontology views under construction, the integration may follow a combination of these strategies.

### Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* as participant;

**What-Input:** views of the ontology under construction and the views of the reused ontologies, elaborated in the activities Model Ontology (CAP-MODE) and Extract Ontology View (CAP-VIEW);

**What-Output:** conceptual model of the ontology under construction.

**Running Example:**

For OOC-O, the integration was performed iteratively so that the view of the reused SCO ontology (red color) anchored the concepts of the object-oriented ontology (yellow color) with specialization relationships (yellow color), as presented in Figure 5.3.

## 5.5 Modularize Ontology (**CAP-MODU**)

Capture activity that identifies the modules into which the ontology can be decomposed and which can be considered separately while being interconnected with other modules [d'A12].

A complex domain ontology consisting of many concepts cannot be built and maintained easily and is difficult to reuse in its entirety. Thus, the goal of using modularization includes [d'A12]:

1. improving performance by allowing distributed or targeted processing;
2. facilitating the development and maintenance of the ontology by breaking it into independent, loosely coupled components;
3. facilitating the reuse of parts of the ontology, since relevant adaptations have already been applied.

This activity is executed iteratively in order to elaborate a more understandable and decoupled model, prioritizing more cohesion and less dependency. Different scenarios and applications require different ways to modularize [DSHS07]

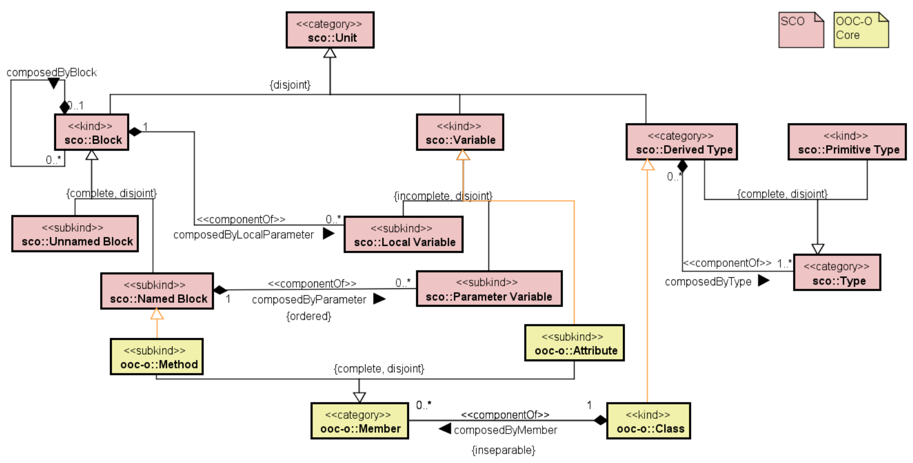


Figure 5.3: Integration of the reused ontology (SCO) with the ontology under construction (OOC-O).

and therefore an appropriate modularization strategy should be applied, considering: (i) comprehension criteria and ontology size; and (ii) preservation of the completeness of the modules, assigning relationships among the associated modules and replicating concepts needed for their understanding and visualization.

## Activity definition:

**Who:** *ontology engineer* as responsible and *domain expert* as participant;

**What-Input:** conceptual model of the ontology under construction, elaborated in the activities Model Ontology (CAP-MODE) and Integrate Ontology (CAP-INTE);

**What-Output:** conceptual model of the ontology under construction, modularized.

## Support activities in parallel:

**Adopt Modularization Strategy (REU-MODU)**

Reuse activity that aims to define the modularization strategy to be adopted in modeling the ontology under construction and the views of the reused ontologies. For this, it is suggested to consider two main modularization strategies [d'A12]:

1. If the ontology aggregates many concepts that make it difficult to understand and maintain it is recommended to apply the partitioning strategy so that subdomains are treated independently and

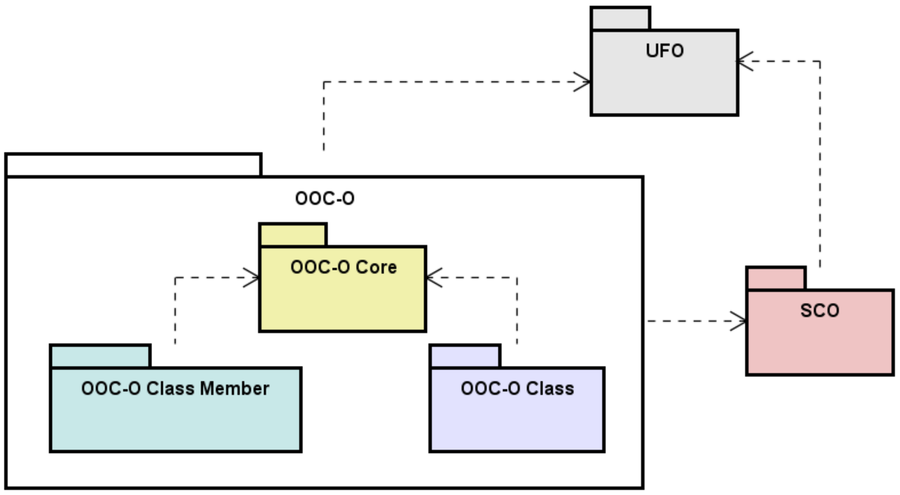


Figure 5.4: Modularization of OOC-O.

maintain their completeness in the form of modules, i.e. partitioning the ontology;

2. If some concepts are treated in the ontology jointly, it is recommended to apply the transversal extraction method so that, starting from these concepts, their relationships are visited recursively and, from them, the set of concepts and relationships needed to be gathered is identified.

**Running Example:**

For OOC-O, the ontology was modularized adopting the partitioning strategy, where *Core* represents the main concepts of the object-oriented code, *Class* represents the concepts derived from class and *Class Member* represents the concepts derived from class members, i.e., attributes and methods. Figure 5.4 presents the modularization adopted in OOC-O, as well as its relation to the reused ontologies.

## 5.6 Document Reference Ontology (DOC-MODE)

Documentation activity that documents the conceptual model of the ontology under construction by means of the Reference Ontology Document, complementing the document elaborated in the activity Document Premise of Reference Ontology (DOC-REFE), with the following considerations:

- Modularization: representation of the modules defined for the ontology under construction. It is suggested to use a UML package diagram whose packages are related through dependencies [Fal14] and standardized by colors;
- Conceptual Model: presentation of the ontology conceptual model;
- Model Axioms: presentation of the constraints defined in informal axioms for axioms written in first-order logic or OCL. Formal axioms support the descriptions of the informal axioms, not replace them;
- Model Description: text in natural language containing a description of the model and highlighting its concepts and relationships;
- Model Dictionary: presentation of concepts and their meanings in the form of a dictionary.

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document produced in activity Document Premise of Reference Ontology (DOC-REFE) and the results from the activities performed earlier in this phase;

**What-Output:** a complete Reference Ontology Document.

**Running Example:**

For OOC-O, the Reference Ontology Document presented in Table 4.1 is complemented with the conceptual model documentation, presented in Table 5.2.

Table 5.2: Fragment of OOC-O's Reference Ontology Document.

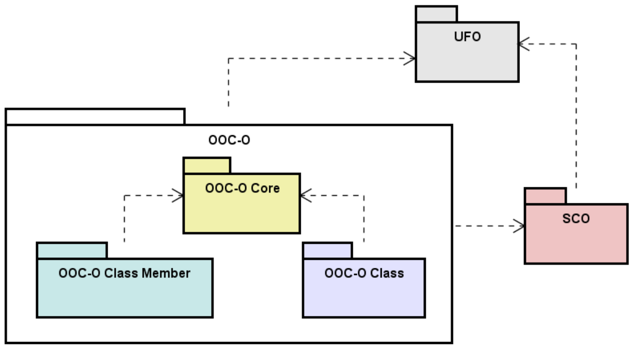
| | | | |
|---|---|---|---|
| **Document:** | Reference Ontology | **Version:** | v.01 |
| **Ontology:** | OOC-O Object-Oriented Code Ontology | **Date:** | April 11, 2019 |
| **Modeling Language:** | | | |
| OntoUML | | | |
| **Foundational Ontology:** | | | |
| UFO | Ontology characterized in modal logic and cognitive psychology that aims to provide a higher semantic level about the world in a conceptual model of a given knowledge domain, systematizes issues such as notions of types and their instances; objects, and their intrinsic properties; the relationship between identity and classification; distinctions between types and their relationships; part-whole relationships, among others. | | |
| **Criteria:** | | | |
| CR01 | Concept refers to programming constructs used at compile-time; | | |
| CR02 | Concept is present in more than 50% of the programming languages analyzed; | | |
| CR03 | Concept related to the main elements of object-orientation: class, method and attribute. | | |

**Ontologies to Reuse:**

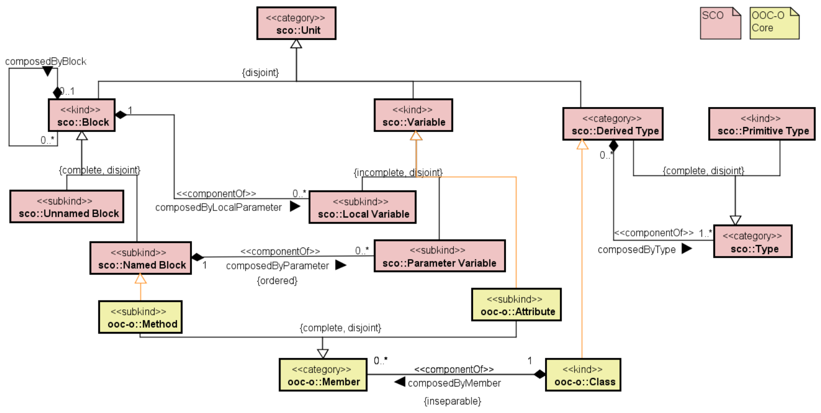
| **Prefix** | **Definition** | **Analysis** |
|---|---|---|
| SCO | Core Ontology - UFO<br>Source code Ontology that describes the general concepts of a source code, independent of programming language. | OO code is a specialization of source code. |

**Modularization:**

| **Sub-ontology** | **Definition** |
|---|---|
| Core | Ontology that represents the main concepts of object-oriented code. |
| Class | Ontology that details the concepts derived from class. |
| Class Member | Ontology that details the concepts derived from class members, i.e. attributes and methods. |



**Ontology:** OOC-O Core



**Axioms:**

| **Axiom** | **Definition** |
|---|---|

**Dictionary of Concepts:**

| **Concept** | **Definition** |
|---|---|
| Class | Abstract data type and mechanism for defining an abstract data type in an OO programming language. Class describes the attributes of its objects, as well as the methods they can perform. |
| Member | Members that make up the classes, such as Method and Attribute. |
| Method | Named Block (Function or Procedure) that belongs to the class and provides a way to define the behavior of an object that is invoked when a message is received by the object. |

| | |
|---|---|
| Attribute | Variable that belongs to the class and provides a way to set the state of its objects. |

## 5.7 Control Reference Ontology (MAN-MODE)

Configuration Management activity that controls the changes, versions and deliveries of the information registered in the Reference Ontology Document, described in the activity Document Reference Ontology (DOC-MODE), complementing the records made in the activity Document Premise of Reference Ontology (DOC-REFE).

It is suggested that the same version control mechanism adopted in activity Control Specification (MAN-SPEC) continues to be used here.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in activity Document Reference Ontology (DOC-MODE);

**What-Output:** control of the Reference Ontology Document.

### Running Example:

For OOC-O, the header of the Reference Ontology Document continues to be used to control its versions, complementing the controls recorded in the activity Control Premise of Reference Ontology (MAN-REFE), as presented in Table 5.2.

## 5.8 Evaluate Reference Ontology (EVA-MODE)

Evaluation activity that assesses whether the information recorded in the Reference Ontology Document meet the expectations of the *ontology owner* and are in accordance with the *domain expert*, applying static evaluation over the ontology's requirements and purpose. This is done by validating the information and returning to the beginning of the phase whenever necessary.

For ontology **verification**, one must assess through technical reviews whether the functional requirements raised are answered by the concepts and relationships represented in the ontology's conceptual model. For this, it is suggested that the competency questions raised in the activity Elicit Requirements (REQ-ELIC) be answered exclusively by the concepts and relations of the ontology, relating them in order to build a reasoning path.

The pattern $c_1 > r > c_2$ transcribes a relationship where $c_1$ is the source concept of the relationship, $c_2$ is the target concept of the relationship and $r$ is the relationship between the concepts $c_1$ and $c_2$. The transcription of a given relation $r$ must follow its relation type: association relation is represented by the relation name, composition relation is represented by *componentOf*, specialization relation is represented by *subtypeOf* and generalization relation is represented by *specializedBy*.

For ontology **validation**, one must evaluate whether the ontology meets real world situations by instantiating its concepts. For this, it is suggested that the instances of the concepts cataloged in activity Catalog Concepts (CAP-CATA) be mapped to instances of the ontology concepts.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in the activity Document Reference Ontology (DOC-MODE);

**What-Output:** evaluation of the Reference Ontology Document.

### Running Example:

For OOC-C, ontology verification is provided by the answers to the competency questions, such as for functional requirement RF03, answered by: Member componentOf Class; Attribute and Method subtypeOf Member.

The validation of the ontology is given by instantiating its concepts, such as the Java language code shown in Listing 5.1, in which: Polygon *instanceOf* Class; side *instanceOf* Attribute; perimeter *instanceOf* Method; private and public *instanceOf* Visibility; int *instanceOf* Primitive Type & Value Type; and void, int *instanceOf* Primitive Type & Return Type.

Listing 5.1: Fragment of Java object-oriented code.

```
public class Polygon{
       private int side;
       public void perimeter(){};
}
```

## 5.9 Publish Reference Ontology (PUB-REFE)

Publication activity that publishes the Reference Ontology, defined in the activity Document Reference Ontology (DOC-MODE).

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in the activity Document Reference Ontology (DOC-MODE);

**What-Output:** reference ontology made available.

## Running Example:

For OOC-O, the Reference Ontology Document was published on the project website and is accessible at `https://nemo.inf.ufes.br/projetos/sabiox/`.

# Chapter 6

# Design Phase

The Design Phase aims to define technological issues that will guide the implementation of the operational ontology, i.e., decisions that are dependent on the technological environment. In this context, the reference ontology, elaborated in the Capture Phase, provides the initial information to specify the design that may be originated by different technological choices.

This process aims to bridge the gap between the conceptual modeling of reference ontologies and their encoding in terms of an operational ontology language [Gui07]. To do this, it defines a set of activities that must be performed iteratively until the expected quality standard is defined. Given the technical nature of the process, it is suggested that the non-functional requirements defined in the activity Elicit Requirements (REQ-ELIC) be revisited at each iteration, in order to add, detail and delete non-functional requirements related to the technological issues of the operational ontology. The activities of these respective processes are presented in the following subsections.

## 6.1 Define Encoding Language (DES-LANG)

Design activity that defines the representation language to be applied in the construction of the operational ontology, i.e., in the codification. The choice of language directly influences the integration and reuse of the ontology under construction, since ontologies developed in different languages may require a great deal of effort to adapt.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in the activity Document Reference Ontology (DOC-MODE);

**What-Output:** coding language to be adopted in the operational ontology.

### Support activities in parallel:

**Adopt Encoding Language (REU-ELAN)**
Activity that aims to define the encoding language to be adopted in the construction of the operational ontology. There are several languages to represent ontologies, such as KIF, based on first-order logic; RDF(S), based on semantic networks and frame-based primitives; and OWL, based on properties and attributes of RDF(S) vocabularies corresponding to description logics [MH04]. Given the popularity, well-defined meaning description and rich relationships, the use of the OWL language is suggested.

### Running Example:

For OOC-O, OWL is chosen for encoding the operational ontology.

## 6.2 Identify Vocabularies (DES-VOCA)

Design activity that identifies the vocabularies to be adopted in the coding of the ontology under construction, both base vocabularies to assist in the construction of the ontology and vocabularies of the reused ontologies.

These vocabularies must adopt an encoding language compatible with the language adopted in the ontology under construction, defined in the activity Define Encoding Language (DES-LANG). In this context, *vocabulary* is defined as the set of concepts and relations of a given domain, without necessarily adopting a complex formalism such as that of an ontology.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in the activity Document Reference Ontology (DOC-MODE);

**What-Output:** vocabularies to be reused in the operational ontology.

### Support activities in parallel:

**Understand Vocabulary (KNO-VOCA)**
Knowledge Acquisition activity that aims to understand the classes and properties of the vocabularies adopted for the construction of the operational ontology. Since not all vocabulary elements will be reused, those that correspond to the ontology under construction must be defined.

**Adopt Base Vocabularies (REU-BVOC)**
Reuse activity that aims to select vocabularies to assist the encoding

of the ontology under construction, i.e. vocabularies that do not define concepts and relations of the domain of the ontology under construction. To do this, it is necessary to search for vocabularies in search engines and repositories such as, e.g., `https://lov.linkeddata.es/dataset/lov/`. Such vocabularies usually include, among others, `http://www.w3.org/2002/07/owl#`, `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and `http://www.w3.org/2000/01/rdf-schema`.

**Adopt Vocabularies from Reused Ontologies (REU-RVOC)**
Reuse activity that aims to select vocabularies from the ontologies reused in the reference ontology, i.e., vocabularies that define concepts and relations from the domain of the ontology under construction. To do this, the *ontology engineer* must search the available resources of the reused ontologies. If the vocabulary of the reused ontology is not accessible, it will be built in the Implementation Phase, together with the ontology under construction.

**Running Example:**

For OOC-O, the following vocabularies were reused:

- OWL: `http://www.w3.org/2002/07/owl#`;
- RDF: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`;
- XSD: `http://www.w3.org/2001/XMLSchema#`;
- RDFS: `http://www.w3.org/2000/01/rdf-schema#`;
- SCO: `http://nemo.inf.ufes.br/projetos/oscin/sco#`.

## 6.3 Define Ontology Encoding (DES-CODE)

Design activity that defines the coding to be applied in the operational ontology, i.e., the structure of the elements that represent the reference ontology in the operational language.

This activity indicates how the architecture of the operational ontology will be derived from the reference ontology, considering the defined coding language. An ontology can be defined as a set of representational primitives that model a knowledge domain, characterized as concepts/classes, relations/properties, and attributes/properties of data types [Sli15]. In the case of the OWL language, the architecture will be based on classes and properties.

From the conceptual model, one must define the concepts and relationships that will be represented as classes and properties in the operational ontology. Based on the classes, a taxonomy of the ontology must be defined, i.e., to organize the concepts in a hierarchical manner. In the case of the OWL language, relationships of the type subClassOf can be created, inherited from RDFS.

The nomenclature and the use of vocabularies to be adopted in encoding the ontology must be defined. For the nomenclature, we suggest adopting rules that allow the association with the concepts and relationships defined in the reference ontology. For the vocabulary, it is suggested to clearly define how the concepts and relations will be used in the operational ontology. This activity is relevant to ensure the standard encoding of the ontology, especially when the Implementation Phase is performed in a parallel or distributed way.

The architecture should adopt modularization for ease of understanding and reuse. To do this, one should visit the modularization defined for the reference ontology in order to use it as a basis for the modularization of the operational ontology and apply adjustments for its operationalization. The architecture must consider the ontologies reused in the construction of the reference ontology in the form of vocabularies, as well as consider characteristics related to reasoning and performance.

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Reference Ontology Document prepared in the activity Document Reference Ontology (DOC-MODE);

**What-Output:** coding of the operational ontology.

## Support activities in parallel:

**Map Reference Ontology (KNO-MAP)**

Knowledge Acquisition activity that aims to map the concepts and relationships from the reference ontology that will be represented as classes and properties in the operational ontology.

For OWL, each class is mapped as member of the OWL class Thing and instance of Class, and each property as ObjectProperty or Datatatype-Property, inherited from the RDF class Property. Furthermore, value and cardinality restrictions can be defined via properties.

## Running Example:

In OOC-O, the concepts and relations of the reference ontology are represented as OWL classes and properties. The URI of the individual is formed by the pattern `<module> <class> <method> <variable>` in order to ensure the unique identifier in the ontology.

## 6.4 Document Premise of Operational Ontology (DOC-OPER)

Documentation activity that documents the assumptions of the operational ontology through the Operational Ontology Document, which should include the questions defined in the Design Phase, as well as the following:

- Coding Language: identification of the coding language to be used in the implementation of the ontology;
- Vocabularies: prefix identifying the vocabulary and its definition;
- Coding Rules: description of the rules to be applied in the coding of the ontology;
- Architecture: description or representation of the architecture to be adopted in the coding of the ontology.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** activities performed in the Design Phase;

**What-Output:** Operational Ontology Document with its respective assumptions.

### Running Example:

For OOC-O, the Operational Ontology Document with the premises established for the operational ontology is presented in Table 6.1.

Table 6.1: Operational Ontology Document

| | | | |
|---|---|---|---|
| **Document:** | Operational Ontology | **Version:** | v.01 |
| **Ontology:** | OOC-O Object-Oriented Code Ontology | **Date:** | April 26, 2019 |
| **Codification Language:** | | | |
| OWL | | | |
| **Vocabularies:** | | | |
| **Prefixo** | **Definição** | | |
| owl | http://www.w3.org/2002/07/owl# | | |
| rdf | http://www.w3.org/1999/02/22-rdf-syntax-ns# | | |
| xml | http://www.w3.org/XML/1998/namespace | | |
| xsd | http://www.w3.org/2001/XMLSchema# | | |
| rdfs | http://www.w3.org/2000/01/rdf-schema# | | |
| sco | http://nemo.inf.ufes.br/projetos/oscin/sco# | | |

**Codification Rules:**

| **Rule** | **Definition** |
|---|---|
| Class Name | Classes are named from the reference ontology following the pattern *Pascal Case*, initial letter of each word capitalized, without space, hyphen, underscore or punctuation. |
| Property Name | Properties are named from the reference ontology following the *Camel Case* pattern, initial letter of the first word in lowercase and initial letter of the remaining words in uppercase, without space, hyphen, underscore or punctuation. Properties with the same name in the reference ontology are named by adding the target concept to the relation name. |
| Individual URI | Individuals are named following the pattern `<module>` `<class>` `<method>` `<variable>` according to the type of individual. |
| Class Description | Classes are described using the comment property of RDFS. |
| Specialization Relation | The specialization between classes are defined using the subClassOf property of RDFS. |
| Disjunction Relation | The disjunction between classes are defined using the disjointWith property of OWL. |
| String Datatype | Data of type string is defined as XMLSchema#string from RDFS. |

**Architecture:**
**Ontologies:**

| **Prefix** | **Description** |
|---|---|
| ooc-o | Ontology that corresponds to the OOC-O Core reference sub-ontology. |
| ooc-o-class | Ontology that corresponds to the OOC-O Class reference sub-ontology. |
| ooc-o-classmember | Ontology that corresponds for the reference OOC-O Class Member sub-ontology. |

**Taxonomy:**
- Concepts of the kind and category type are anchored in the Class class of OWL;
- Concepts of the subkind type are anchored to the respective kinds of the domain according to the conceptual model;
- Concepts of type quality are anchored to the ObjectProperty property of owl. Except for the concept name which is anchored in the DatypeProperty property of owl.

## 6.5 Control Premise of Operational Ontology (MAN-OPER)

Configuration Management activity that controls the changes, versions and deliveries of the information registered in the Operational Ontology Document, described in the activity Document Premise of Operational Ontology (DOC-OPER).

It is suggested that the same version control mechanism adopted in activity Control Specification (MAN-SPEC) continues to be used here.

### Activity definition:

**Who:** *ontology engineer* as responsible;;

**What-Input:** Operational Ontology document prepared in the activity Document Premise of Operational Ontology (DOC-OPER);

**What-Output:** control of the Operational Ontology Document.

**Running Example:**

For OOC-O, the header of the Operational Ontology Document is adopted to control its versions, as presented in Table 6.1.

## 6.6 Evaluate Premise of Operational Ontology (**EVA-OPER**)

Evaluation activity that assesses whether the information registered in the Operational Ontology Document meets the expectations of the *ontology owner* and is in accordance with the ontology and quality requirements defined in the Requirements Phase. This should be done by validating the information recorded with the *ontology owner* through technical review and returning to the beginning of the phase whenever necessary.

### Activity definition:

**Who:** *ontology engineer* and *ontology owner* as responsible;

**What-Input:** the Ontology Specification Document and the Reference Ontology Document prepared in the Requirements Phase and Capture Phase, respectively;

**What-Output:** evaluation of the Operational Ontology Document.

**Running Example:**

For OOC-O, meetings are held with *ontology owner* and *ontology engineer* in order to validate the operational ontology document, presented in Table 6.1.

# Chapter 7

# Implementation Phase

The Implementation Phase aims to implement the reference ontology produced in the Capture Phase into an operational ontology, according to the design specifications elaborated in the Design Phase. This phase transforms the reference ontology into a machine-interpretable ontology, making it available for use in applications, decision support, and domain reasoning.

## 7.1 Code Concepts (IMP-CONC)

Implementation activity that encodes the concepts of the reference ontology as elements of the formal operational ontology language, according to the assumptions defined in the Operational Ontology Document. For the OWL language, the concepts are encoded as classes in the operational ontology.

Reuse activities from previous phases — Adopt Encoding Language (REU-ELAN) in Define Encoding Language (DES-LANG); Adopt Base Vocabularies (REU-BVOC) and Adopt Vocabularies from Reused Ontologies (REU-RVOC) from Identify Vocabularies (DES-VOCA) — have an impact on this activity, in terms of adopting the previously defined encoding language and selected vocabularies, respectively.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Operational Ontology Document prepared in activity Document Premise of Operational Ontology (DOC-OPER);

**What-Output:** concepts encoded in the operational ontology.

**Running Example:**

For OOC-O, the concept Member from the reference ontology is encoded in the operational ontology as presented in Listing 7.1.

Listing 7.1: Fragment of the concepts from OOC-O's operational ontology.

```
<owl:Class rdf:about="http://ooc-o#Member">
        <rdfs:comment rdf:datatype="http://www.w3.org/2000/01/rdf-
    schema#Literal">Element which makes up a class, defined as an
    attribute or a method.</rdfs:comment>
</owl:Class>
```

## 7.2 Code Relations (IMP-RELC)

Implementation activity that encodes the relations of the reference ontology as properties and constraints in a formal operational ontology language, according to the assumptions defined in the operational ontology document. For the OWL language, the relations are encoded as properties and constraints in the operational ontology.

Reuse activities from previous phases — Adopt Encoding Language (REU-ELAN) in Define Encoding Language (DES-LANG); Adopt Base Vocabularies (REU-BVOC) and Adopt Vocabularies from Reused Ontologies (REU-RVOC) from Identify Vocabularies (DES-VOCA) — have an impact on this activity, in terms of adopting the previously defined encoding language and selected vocabularies, respectively.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Operational Ontology Document prepared in activity Document Premise of Operational Ontology (DOC-OPER);

**What-Output:** relations encoded in the operational ontology.

**Running Example:**

For OOC-O, the relation componentOfClass from the reference ontology is encoded in the operational ontology as presented in Listing 7.2.

Listing 7.2: Fragment of the relations from OOC-O's operational ontology.

```
<owl:ObjectProperty rdf:about="http://ooc-o#componentOfClass">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
    FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://ooc-o#Member"/>
        <rdfs:range rdf:resource="http://ooc-o#Class"/>
</owl:ObjectProperty>
```

## 7.3 Code Axioms (IMP-AXIO)

Implementation activity that encodes the axioms defined in the reference ontology according to the operational language.

Reuse activity Adopt Encoding Language (REU-ELAN) from Define Encoding Language (DES-LANG) has an impact on this activity, in terms of adopting the previously defined encoding language.

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Operational Ontology Document prepared in activity Document Premise of Operational Ontology (DOC-OPER);

**What-Output:** axioms encoded in the operational ontology.

**Running Example:**

For OOC-O, the axiom defining Method and Attribute as *disjoint* is presented in Listing 7.3.

Listing 7.3: Fragment of the axioms from OOC-O's operational ontology.

```
 <rdf:Description>
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
    AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="http://ooc-o#Method"/>
            <rdf:Description rdf:about="http://ooc-o#Attribute"/>
        </owl:members>
</rdf:Description>
```

## 7.4 Document Operational Ontology (DOC-OPER)

Documentation activity that documents the operational ontology in the coding language defined in the activity Define Encoding Language (DES-LANG), following the results of the previous activities in the Implementation Phase and according to the assumptions defined in the operational ontology document, from activity Document Premise of Operational Ontology (DOC-OPER).

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** results from previous activities from the Implementation Phase;

**What-Output:** operational ontology in the defined language.

**Running Example:**

In OOC-O, the reference ontology is encoded in the operational ontology, a fragment of which is shown in Listing 7.4.

## 7.5 Control Operational Ontology (MAN-OPER)

Configuration Management activity that controls the changes, versions and deliveries of the information registered in the operational ontology, described in the activity Document Operational Ontology (DOC-OPER).

For this, it is suggested to use metadata in the ontology to register the version and its respective date, storing the history of all versions for eventual comparisons or restorations of previous versions, using a mechanism as simple as different files in the file system, to the use of version control systems such as, e.g., Git.[1]

### Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** operational ontology elaborated in the activity Document Operational Ontology (DOC-OPER);

**What-Output:** control of the operational ontology.

**Running Example:**

For OOC-O, the operational ontology relies on the metadata rdfs:isDefinedBy and owl:versionInfo to present the authorship of the ontology and its version.

## 7.6 Evaluate Operational Ontology (EVA-OPER)

Evaluation activity that assesses whether the information recorded in the operational ontology meets the expectations of the *ontology owner* and is in accor-

[1] https://git-scm.com/

Listing 7.4: Fragment of OOC-O's operational ontology.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://ooc-o#"
     xml:base="http://ooc-o"
     xmlns:sco="http://sco#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://ooc-o">
        <rdfs:comment rdf:datatype="http://www.w3.org/2000/01/rdf-
    schema#Literal">The Object-Oriented Code Ontology (OOC-O) aims
    to identify and represent the semantics of the entities present
    at compile time in object-oriented (OO) code.</rdfs:comment>
        <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/2000/01/
    rdf-schema#Literal">https://link.springer.com/chapter
    /10.1007/978-3-030-33223-5_3</rdfs:isDefinedBy>
        <rdfs:label rdf:datatype="http://www.w3.org/2000/01/rdf-
    schema#Literal">OOC-O: Operational Ontology on Object-Oriented
    Code</rdfs:label>
        <owl:versionInfo rdf:datatype="http://www.w3.org/2000/01/rdf
    -schema#Literal">1.0 - 31/05/2019</owl:versionInfo>
    </owl:Ontology>

    <owl:Class rdf:about="http://ooc-o#Class">
        <rdfs:subClassOf rdf:resource="http://sco#Derived_Type"/>
        <owl:disjointWith rdf:resource="http://ooc-o#Attribute"/>
        <owl:disjointWith rdf:resource="http://sco#Primitive_Type"/>
        <owl:disjointWith rdf:resource="http://ooc-o#Method"/>
        <rdfs:comment rdf:datatype="http://www.w3.org/2000/01/rdf-
    schema#Literal">An abstract-definition element in the OO
    programming language to express such definitions, that is, class
     is an abstract data type and a mechanism for defining an
    abstract data type in a program. Class describes the attributes
    of its objects as well as the methods they can execute.</rdfs:
    comment>
    </owl:Class>

    <owl:Class rdf:about="http://ooc-o#Member">
        <rdfs:comment rdf:datatype="http://www.w3.org/2000/01/rdf-
    schema#Literal">Element which makes up a class, defined as a
    variable or method.</rdfs:comment>
    </owl:Class>

    <owl:ObjectProperty rdf:about="http://ooc-o#composedByMember">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
    FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://ooc-o#Class"/>
        <rdfs:range rdf:resource="http://ooc-o#Member"/>
    </owl:ObjectProperty>

</rdf:RDF>
```

dance with the ontology and quality requirements defined in the Requirements Phase.

For ontology **validation**, one should evaluate whether the ontology meets real-world situations by instantiating its concepts as individuals of the ontology.

For ontology **verification**, one must evaluate whether the functional and non-functional requirements raised are answered by means of queries applied on the instantiated ontology. For this, it is suggested that the competency questions raised in the Elicit Requirements (REQ-ELIC) activity be translated into query language relative to the coding language. Thus, from a set of test cases, it is possible to verify the behavior of the operational ontology according to the executed queries and the returned instances. For ontologies built with OWL, it is suggested to use the SPARQL query language.

The evaluation can be performed as [Fal14]:

- Subontology Test, evaluation performed individually for each encoded subontology;
- Integration Test, evaluation performed on the connections between subontologies;
- Ontology Test, evaluation performed in the context of the complete ontology, and may include testing of *stress*, performance, inference, and others.

### Activity definition:

**Who:** *ontology engineer* and *ontology owner* as responsible;

**What-Input:** Ontology Specification Document produced in the Requirements Phase and the operational ontology built in activity Document Operational Ontology (DOC-OPER);

**What-Output:** evaluation of the operational ontology.

### Running Example:

For OOC-O, the source code (shown in Listing 5.1, from activity Evaluate Reference Ontology (EVA-MODE)) is instantiated in the ontology as shown in Listing 7.5. SPARQL queries are crafted to answer the competency questions, such as RF02 answered in Listing 7.6.

## 7.7 Publish Operational Ontology (PUB-OPER)

Publication activity that publishes the Operational Ontology produced in activity Document Operational Ontology (DOC-OPER).

Listing 7.5: Fragment of the ontology instances representing a Java code.

```
<owl:NamedIndividual rdf:about="http://ooc-o#project_Polygon">
    <rdf:type rdf:resource="http://ooc-o#Module"/>
    <visibleBy rdf:resource="http://ooc-o#visibility_public"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://ooc-o#
     project_Polygon_perimeter">
    <rdf:type rdf:resource="http://ooc-o#Concrete_Method"/>
    <componentOfClass rdf:resource="http://ooc-o#project_Polygon"/>
    <visibleBy rdf:resource="http://ooc-o#visibility_public"/>
    <returnedBy rdf:resource="http://ooc-o#type_void"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="http://ooc-o#project_Polygon_side">
    <rdf:type rdf:resource="http://ooc-o#Instance_Variable"/>
    <componentOfClass rdf:resource="http://ooc-o#project_Polygon"/>
    <visibleBy rdf:resource="http://ooc-o#visibility_private"/>
    <valuedBy rdf:resource="http://ooc-o#type_int"/>
</owl:NamedIndividual>
```

Listing 7.6: SPARQL query that answers RF02.

```
SELECT ?class
WHERE {?class rdf:type ooc-o:Class. }
```

## Activity definition:

**Who:** *ontology engineer* as responsible;

**What-Input:** Operational Ontology prepared in activity Document Operational Ontology (DOC-OPER);

**What-Output:** operational ontology made available.

**Running Example:**

For OOC-O, the operational ontology was published on the project website and is accessible at `https://nemo.inf.ufes.br/projetos/sfwon/`.

# Glossary

## Phases and Activities

| Phase | Activity | Page |
|---|---|---|
| Requirements Phase | Define Purpose (REQ-PURP) | 9 |
| | Capture Ontology Purpose (KNO-PURP) | 10 |
| | Identify and Size Domain (REQ-DOMN) | 10 |
| | Understanding Ontology Domain (KNO-DOMU) | 11 |
| | Size Ontology Domain (KNO-DOMS) | 11 |
| | Elicit Requirements (REQ-ELIC) | 11 |
| | Capture Ontology Requirements (KNO-REQI) | 12 |
| | Negotiate Ontology Requirements (KNO-REQN) | 12 |
| | Identify Subdomains (REQ-SUBD) | 13 |
| | Document Specification (DOC-SPEC) | 14 |
| | Control Specification (MAN-SPEC) | 16 |
| | Evaluate Specification (EVA-SPEC) | 16 |
| Setup Phase | Define Modeling Language (SET-LANG) | 18 |
| | Adopt Modeling Language (REU-LANG) | 19 |
| | Define Foundational Ontology (SET-FOUN) | 19 |
| | Adopt Foundational Ontology (REU-FOUN) | 20 |
| | Adopt Ontology Patterns (REU-PATT) | 20 |
| | Define Concept Criteria (SET-CRIT) | 20 |
| | Define Ontologies to Reuse (SET-REUS) | 21 |
| | Adopt Reused Ontology (REU-ONTO) | 21 |
| | Document Premise of Reference Ontology (DOC-REFE) | 23 |
| | Control Premise of Reference Ontology (MAN-REFE) | 24 |
| | Evaluate Premise of Reference Ontology (EVA-REFE) | 24 |
| Capture Phase | Identify Concepts (CAP-CONC) | 27 |
| | Catalog Concepts (CAP-CATA) | 27 |

# Bibliography

[AFS19] Camila Zacché de Aguiar, Ricardo de Almeida Falbo, and Vítor E. S. Souza. OOC-O: A Reference Ontology on Object-Oriented Code. In *Proc. of the 38th International Conference on Conceptual Modeling (ER 2019)*, pages 13–27, Salvador, BA, Brazil, 2019. Springer.

[Agu21] Camila Zacche de Aguiar. *Interoperabilidade Semântica entre Códigos-Fonte baseada em Ontologia.* Phd thesis, Federal University of Espírito Santo, Brazil, 2021.

[d'A12] Mathieu d'Aquin. *Modularizing ontologies*, pages 213–233. Springer, 2012.

[DSHS07] Mathieu D'Aquin, Anne Schlicht, Stuckenschmidt Heiner, and Marta Sabou. Ontology Modularization for Knowledge Selection: Experiments and Evaluations. In *Proc. of the 18th International Conference on Database and Expert Systems Applications*, pages 874–883, 2007.

[Eif06] ECMA Eiffel. Analysis, design and programming language. Technical report, ECMA Standard ECMA-367, ECMA, 2006.

[Fal14] Ricardo A. Falbo. SABiO: Systematic Approach for Building Ontologies. In Giancarlo Guizzardi, Oscar Pastor, Yair Wand, Sergio de Cesare, Frederik Gailly, Mark Lycett, and Chris Partridge, editors, *Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering.* CEUR, 2014.

[FGGP13] Ricardo A. Falbo, Giancarlo Guizzardi, Aldo Gangemi, and Valentina Presutti. Ontology patterns: Clarifying concepts and terminology. In *Proc. of the 4th International Conference on Ontology and Semantic Web Patterns (WOP'13)*, volume 1188, pages 14–26. CEUR, 2013.

[FLGP02] Mariano Fernández-López and Asunción Gómez-Pérez. Overview and analysis of methodologies for building ontologies. *Knowledge Engineering Review*, 17(2):129–156, 2002.

[GBBF$^+$22] Giancarlo Guizzardi, Alessander Botti Benevides, Claudenir M. Fonseca, Daniele Porello, João Paulo A. Almeida, and Tiago Prince Sales. UFO: Unified Foundational Ontology. *Applied Ontology*, 17(1):167–210, 2022.

[GDD06] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Architecture and Ontology Development*. Springer, 2006.

[GF95] Michael Grüninger and Mark S. Fox. Methodology for the Design and Evaluation of Ontologies. In *Proc. of the Workshop on Basic Ontological Issues in Knowledge Sharing at the 1995 International Joint Conference on Artificial Inteligence (IJCAI)*, pages 1–10, 1995.

[GFB$^+$18] Giancarlo Guizzardi, Claudenir M. Fonseca, Alessander B. Benevides, João Paulo A. Almeida, Daniele Porello, and Tiago P. Sales. Endurant Types in Ontology-Driven Conceptual Modeling: Towards OntoUML 2.0. In *Proc. of the 37th International Conference on Conceptual Modeling (ER 2018)*, pages 136–150. Springer, 2018.

[GJS$^+$18] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, and Daniel Smith. The Java language specification: Java SE 10 edition, February 20, 2018. Technical report, Oracle, 2018.

[GPRA99] Asunción Gómez-Pérez and Mª Dolores Rojas-Amaya. Ontological reengineering for reuse. In *Proc. of the 11th International Conference on Knowledge Engineering and Knowledge Management (EKAW*, pages 139–156, 1999.

[Gui07] Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. In *Databases and Information Systems IV – Selected Papers from the 7th International Baltic Conference, DB&IS 2006*, pages 18–39, 2007.

[Hun12] John Hunt. *Smalltalk and object orientation: an introduction*. Springer Science & Business Media, 2012.

[Laf97] Robert Lafore. *Object-oriented programming in C++*. Pearson Education, 1997.

[MH04] Deborah L. McGuiness and Frank Van Harmelen. OWL Web Ontology Language Overview. Technical report, W3C recommendation, 2004.

[MW04] Prasenjit Mitra and Gio Wiederhold. *An Ontology-Composition Algebra*, pages 93–113. Springer, 2004.

[NM16] Antonio Nicola and Michele Missikoff. A lightweight methodology for rapid ontology engineering. *Communications of the ACM*, 59(3):79–86, 2016.

[Phi10] Dusty Phillips. *Python 3 object oriented programming*. Packt Publishing Ltd, 2010.

[SFGP12] Mari Carmen Suárez-Figueroa and Asunción Gómez-Pérez. *Ontology requirements specification*, pages 93–106. Springer, 2012.

[Sli15] Thabet Slimani. A Study Investigating Knowledge-based Engineering Methodologies Analysis. *International Journal of Computer Applications*, 128(10):6–14, 2015.

[SS20] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. ScrumGuides.org, 2020.

[UK95] Mike Uschold and Martin King. Towards a Methodology for Building Ontologies. In *Prof. of the 1995 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.