Integration of constraint satisfaction problems and ontologies for the formalization and exploitation of knowledge in system configuration

Maryam Mohammadamini

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Genie industriel

Présentée et soutenue par :

Mme MARYAM MOHAMMADAMINI

le lundi 4 décembre 2023

Titre :

Intégration de problèmes de satisfaction de contraintes et d'ontologies pour la formalisation et l'exploitation de connaissances dans la configuration de systèmes

---

Ecole doctorale :

Systèmes (EDSYS)

Unité de recherche :

Laboratoire Génie de Production de l'ENIT (E.N.I.T-L.G.P.)

Directeur(s) de Thèse :

M. THIERRY COUDERT

MME ELISE VAREILLES

Rapporteurs :

M. ALI SIADAT, ENSAM - ARTS ET METIERS PARISTECH

M. CHRISTOPHE MERLO, ESTIA BIDART

Membre(s) du jury :

MME CATHERINE DA CUNHA, ECOLE CENTRALE DE NANTES, Président

M. ABDOURAHIM SYLLA, INP DE GRENOBLE, Membre

MME ELISE VAREILLES, ISAE-SUPAERO, Membre

M. MICHEL ALDANONDO, ECOLE NLE SUP DES MINES ALBI CARMAUX, Membre

M. THIERRY COUDERT, ECOLE NATIONALE D'INGENIEURS DE TARBES, Membre

Acknowledgements

First of all, I would like to express my deep gratitude to my Ph.D. supervisors, Professors Thierry Coudert, Élise Vareilles, and Michel Aldanondo, for their continuous support and guidance. I also want to thank them for their understanding, and patience. They gave me a multitude of academic life lessons and kept me on track during these three years of my Ph.D.

I am grateful to ENIT and ISAE for providing me the resources and infrastructure I required for my research. I would also like to express my gratitude to Eliane, Marie, Cécile, and Caroline, the administrative staff, for their help and companionship. Furthermore, I would like to especially thank Professor Laurent Geneste for his support and guidance during this thesis.

Working in the LGP research laboratory and interacting with incredible researchers has been a real pleasure. My friends in the laboratory made my stay in Tarbes an exciting journey. They include, but are not limited to: Hajar, Faheem, Piere, Mabrouk, Serge, Sadegh, Majid, Rida, Virag, Maryam, Mona, Océane, Kholoud, Rogers.

I would like to express my gratitude to my amazing husband Seyed Reza Hosseini for his tolerance, selflessness, and love. Moreover, I would like to thank my parents, especially my mother, and my in-laws, for their prayers, support, and encouragement.

Additionally, I would like to express my gratitude to Professors Ali Siadat, Christophe Merlo, Catherine Da Cunha, and Abdourahim Sylla for their acceptance to be part of my jury and for all of their suggestions.

Finally, I would like to give all praise to God, the greatest of benefactors, whose support has enabled me to complete this thesis.

List of Contents

List of Figures

List of Tables

List of acronyms

BOM	Bill Of Material
CBR	Case-Based Reasoning
CSP	Constraint Satisfaction Problem
CTO	Configure-To-Order
CTO.GADI$^{(i)}$	Generic Artifact Descriptive Instance (at level i) created during the CTO configuration activity
CTO.GADI$^{(i)}$(CSP)	Generic Artifact Descriptive Instance (at level i) created during the CTO configuration activity mapped into a CSP
CTO.GAI$^{(i)}$	Generic Artifact Instance (at level i) created during the CTO configuration activity
CTO.GASI$^{(i)}_{j}$	Generic Artifact Structural Instance (at level i and with version j) created during the CTO configuration activity
CTO.GASI$^{(i)}_{j}$(CSP)	Generic Artifact Structural Instance (at level i and with version j) created during the CTO configuration activity mapped into a CSP
CTO.rq	CTO requirement
CTO.solution	CTO solution obtained at the end of CTO configuration activity
EB	Experience Base
ETO	Engineer-To-Order
ETO.GADI$^{(i)}$	Generic Artifact Descriptive Instance (at level i) created during the ETO configuration activity
ETO.GADI$^{(i)}$(CSP)	Generic Artifact Descriptive Instance (at level i) created during the ETO configuration activity mapped into a CSP
ETO.GAI$^{(i)}$	Generic Artifact Instance (at level i) created during the ETO configuration activity
ETO.GASI$^{(i)}_{j}$	Generic Artifact Structural Instance (at level i and with version j) created during the ETO configuration activity
ETO.GASI$^{(i)}_{j}$(CSP)	Generic Artifact Structural Instance (at level i and with version j) created during the ETO configuration activity mapped into a CSP
ETO.rq	ETO requirement
ETO.solution	ETO solution obtained at the end of ETO configuration activity
GA$^{(i)}$	Generic Artifact (at level i)

$GADI^{(i)}$	Generic Artifact Descriptive Instance (at level i)
$GADM^{(i)}$	Generic Artifact Descriptive Model (at level i)
$GADM^{(i)}$(CSP)	Generic Artifact Descriptive Model (at level i) mapped into a CSP
$GAI^{(i)}$	Generic Artifact Instance (at level i)
$GASI^{(i)}_{j}$	Generic Artifact Structural Instance (at level i and with version j)
$GASM^{(i)}_{j}$	Generic Artifact Structural Model (at level i and with version j)
$GASM^{(i)}_{j}$(CSP)	Generic Artifact Structural Model (at level i and with version j) mapped into a CSP
GMB	Generic Model Base
KFC	Knowledge Formalization Criteria
KPI	Key Performance Indicator
KRC	Knowledge Reuse Criteria
SysML	Systems Modeling Language
UML	Unified Modeling Language

1. General introduction

In this chapter, we present the general introduction. In section 1.1, we address the context of this Ph.D. thesis. Then, in section 1.2, we define our research questions and scientific problems. In section 1.3, we explain the contributions of this thesis. Finally, in section 1.4, we end up with the outline of the thesis.

1.1. Context

Mass customization has emerged as a powerful concept for competitive advantage in the manufacturing industry. It allows companies to provide customized products that fit each user's requirements at a competitive price while remaining productive (Pine II et al., 1993), (Kotha & Pine, 1994). Additionally, in the fourth industrial revolution, also known as Industry 4.0, companies must respond rapidly to the wide-ranging requirements of many users and provide solutions to meet their requirements (Rüßmann et al., 2015). To achieve this, configuration tools can be employed.

This thesis focuses on the context of system configuration. System configuration can be considered as a type of design activity where systems are defined from a set of predefined subsystems and components while considering a set of restrictions on how the subsystems and components can be combined (Soininen et al., 1998), (Felfernig et al., 2014). The systems considered in this thesis are technical and physical (or tangible) systems (e.g., cars, computers, bicycles, …). Furthermore, we consider systems, subsystems, or components as artifacts. An artifact can be defined as an object that is intentionally produced for a certain purpose (Hilpinen, 1992, 1999). As presented in (Guillon, Ayachi, et al., 2021), the term artifact refers to a result of human activity.

To support the configuration activity and find solutions that satisfy users' requirements, configuration tools called Configurators (Tiihonen & Soininen, 1997) or Product Configuration Software (PCS) (Myrodia et al., 2017) are employed. A configurator is composed of two parts: 1) a Knowledge Base and 2) a Processing Unit. On the one hand, the Knowledge Base stores generic models, each representing a family of artifacts with all possible options and variants. To address the system configuration problem, generic models must first be created by experts and then exploited to fulfill requirements of users. On the other hand, the Processing Unit assists the user in the configuration activity and thus in finding the specific solution that is consistent with both the generic model and the user's needs. In this thesis, the two phases of formalization, leading to the populating of the knowledge base with generic models, and knowledge reuse, using the process unit to support the configuration activity, are studied.

In the knowledge formalization phase, the experts are responsible for collecting, validating, formalizing and updating the knowledge about the artifact families, always including the different options and variants, as well as the different relationships and sometimes their structure, in order to create generic models that are consistent with the diversity of possible solutions in the catalog. This thesis considers two views of the same artifact: a descriptive and a structural view (Erens & Wortman, 1996), (Aldanondo & Vareilles, 2008). These two views allow the same artifact to be configured either by a novice user, who only needs to consider the functional and operational requirements, or by a more expert user, who also needs to consider the technical requirements of the system. The descriptive view explains what the artifact is by giving information about its key attributes and performance indicators. In descriptive view, the artifact is considered as a black box without revealing its internal workings or components. On the contrary, the structural view explains how the artifact is composed of, i.e. its bill of materials. In the structural view, the artifact can be considered as a white box where the list and quantity of each of its artifacts (subsystems and components) appear.

For experts, it is not that easy to collect, validate, formalize and update knowledge for system configuration, as they may have to develop the attributes or structure of existing artifact families, or even design entirely new families with novel attributes and structures. As a result, it is necessary to create, maintain and update generic models (J. Zhang et al., 2005) with sometimes some commonality between them. When several artifacts have commonality, the corresponding knowledge can be gathered into generic models in order to facilitate creation, maintenance and update. Moreover, it is necessary for experts to be able to check the consistency of all the formalized knowledge in order to validate it and authorize its reuse.

In the knowledge reuse phase, generic models are exploited to configure systems and meet users' requirements. Knowledge reuse enables reasoning on generic models whether it represents a descriptive view or a structural view. In this thesis, we consider interactive configuration (Janota et al., 2010), (Vareilles, 2015), which is an iterative process of removing solutions from solution space or eliminating options that are no longer consistent with the choices made by the user and the generic model. Through an iterative process, the user progressively specifies his requirements and converges toward a solution. This configuration activity is called Configure-To-Order (CTO).

In the era of mass customization and Industry 4.0, different user requirements must be met. However, these requirements can sometimes go beyond the scope of formalized knowledge. It means that these out-of-standard requirements, called non-standard requirements (Sylla, Guillon, Vareilles, et al., 2018), cannot be fulfilled by the formalized generic models. Therefore, to meet these non-standard requirements, an engineering activity has to be carried out, during the configuration activity. This engineering activity is called ETO (Engineer-To-Order). These types of requirements are then called ETO requirements and they have to be formalized, and capitalized. In such a context, the configuration activity has to be modified in order to provide solutions that meet the ETO requirements.

1.2. Research questions and scientific problems

According to what we explained in the context, in this thesis, the following Research Questions (RQ) and Scientific Problems (SP) are specifically studied and discussed. The first two are related to the knowledge formalization phase and the last two are related to the knowledge reuse phase.

RQ 1. Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?

- **SP 1.** How can we benefit from the association of ontologies, constraint satisfaction problems, commonality and inheritance principles to better formalize knowledge and define generic models for system configuration?

RQ 2. How can ETO requirements be processed during configuration activity?

- **SP 2.** How can the configuration activity be adapted to formalize and capitalize ETO requirements?

1.3. Contributions

The contributions of this thesis can be summarized as follows:

- **Contribution 1:** To answer the first research question, we propose to associate ontologies, CSP approaches, and inheritance principles to create an ontology of generic models. This association allows us to better manage knowledge, by creating generic models at different levels of abstraction, following their commonality and mainly thanks to the generalization/specialization relation. The use of the CSP approach allows to formalize relations between the characteristics of the families of artifacts, to check their consistency and to facilitate their reuse using reasoning mechanisms such as constraint filtering. This approach facilitates the maintenance, creation and updating of the generic models, as they are organized in a hierarchical structure. This ontology of generic models is based on the principles of commonality: common attributes between several models are aggregated at the top level. In this way, the model at the highest level contains only the knowledge that is common to all the models, while the lower levels contain only the knowledge that is specific to them. This ontology exists for artifact families, and a clear distinction is made between descriptive and structural views. At the end of the knowledge formalization step, an ontology of generic models can be reached, where consistent generic models are structured from the most general to the most specialized.

- **Contribution 2:** To answer the second research question, we propose to adapt Configure-To-Order (CTO) configuration activity to ETO configuration activity in order to fulfill ETO requirements. Our proposal allows us to first select generic models then build instances of generic models, related to descriptive view and/or structural view at a certain level of abstraction. Then, using a generic process for knowledge reuse in CTO, interactively configure instances to fulfill requirements of user. For each artifact, it is possible, if structural view exists, to dive or not into its bill-of-material in order to configure it more finely. At the end of the

knowledge reuse in CTO, a CTO solution is found and then it is capitalized in an experience base allowing recommendations for future configurations. Following this, we propose another generic process dedicated to the reuse of knowledge to meet the ETO requirements. In this way, we propose an adaptation of CTO to ETO by 1) building completely new instances or 2) modifying existing instances. These modifications of instances during ETO configuration are represented as different cases, relevant to either the descriptive or structural view. At the end of the configuration, an ETO solution is proposed to the user and then it is capitalized in an experience base allowing for future reuse and updates of formalized generic models. For a system to deliver, as some requirements can be fulfilled by a CTO activity and others by an ETO activity, the integration of both processes is also studied. This situation results in solutions comprising CTO and ETO artifacts.

1.4. Thesis outline

As illustrated in Figure 1, the rest of the thesis is organized as follows:

- In Chapter 2, a literature review about related issues to our work such as system and system configuration, knowledge formalization for system configuration, and different approaches for system configuration are presented. We have highlighted the CTO and ETO requirements as well as eight criteria necessary for effective formalization and reuse of generic models. Each approach identified is analyzed in the light of these eight criteria. Scientific gaps are identified and research questions are presented accordingly.

- In Chapter 3, our first research question (RQ 1: Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?) is answered which focuses on formalizing knowledge for system configuration. Our first contribution is presented, which entails defining generic models at various abstraction levels, notated $^{(i)}$. Models of Generic Artifacts, notated $GA^{(i)}$ are firstly defined along with their classification in a taxonomy. To provide a clear representation of the artifacts families and of the relations between their characteristics, descriptive and structural views are distinguished. The descriptive view is represented by a Generic Artifact Descriptive Model, notated $GADM^{(i)}$ while the structural view is represented by a Generic Artifact Structural Model, notated $GASM^{(i)}_j$, where j denotes the possible alternatives for the structural views. CSP formalism is used to represent relationships and check consistency of the generic models. Based on commonality, generalization/specialization and inheritance principles, processes which allow to generalize and specialize these models are defined. Then, they are classified in their respective taxonomy. The three taxonomies (GA, GADM and GASM) constitutes the whole ontology in our approach: i.e. consistent, well described and structured pieces of knowledge which can be reused to configure systems.

- In Chapter 4, our second research question (RQ 2: How can ETO requirements be processed during configuration activity?) is addressed which deals with knowledge reuse for system configuration. Our second contribution involves proposing a knowledge reuse process aimed at meeting ETO requirements of users in the ETO situation. First, the CTO situation is taken into account. A generic process allows to create instances of generic models and to configure them in order to fulfill standard requirements, i.e, requirements which can be met

using formalized knowledge without any modifications. The configuration is done interactively with the user using the CSP and filtering mechanisms and leads to a CTO solution. Based on the CTO configuration process, an ETO configuration process which allows to configure systems in ETO situation is proposed. Reusing formalized generic models chosen in the taxonomies, and modifying them in order to meet non-standard ETO requirements, the process allows to reach an ETO solution for the system configuration. Finally, as the system configuration is a mix between CTO situation and ETO situation, the integration of both processes is proposed and leads to solutions where some artifacts have been configured with respect to the generic models (CTO configuration) and others are defined after a design phase (ETO configuration).

- In Chapter 5, a simple but realistic case study of bicycle families is presented to illustrate some of our proposals for the two phases of knowledge formalization and reuse. It is implemented on the OPERA software which has been developed in the context of the ANR[1] project. This case study allows to verify that all our proposals are consistent and feasible.

- Finally, in Chapter 6, the conclusions of this research work and also scientific perspectives are presented.

[1] Project n° ANR-16-CE10-0010

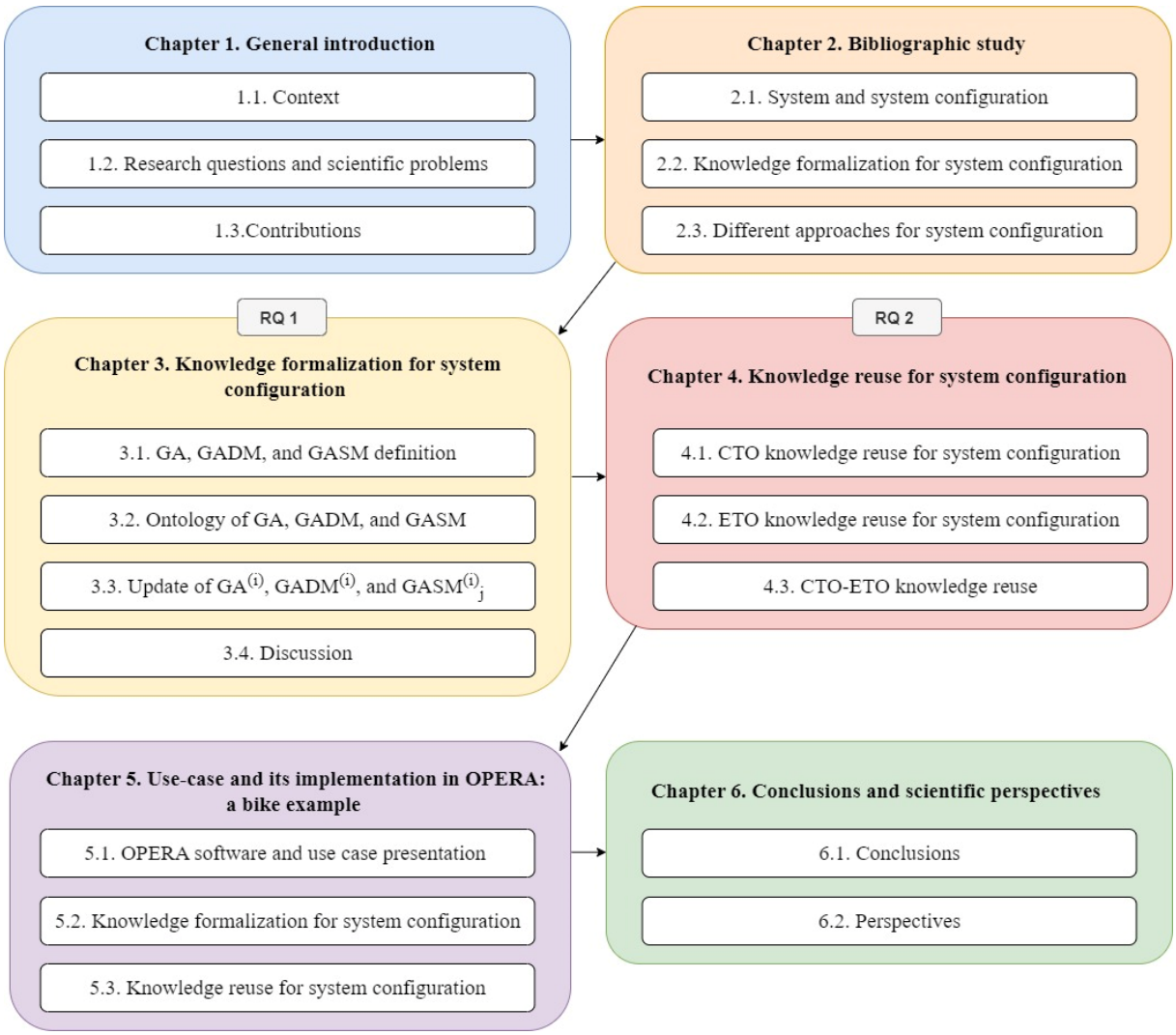

Figure 1. Thesis outline

2. Bibliographic study

In this chapter, we have conducted a literature review to find clues to answer our research questions. In this way, first, we present relevant articles on system and system configuration in section 2.1 to provide an understanding of system configuration which is the context of this thesis, and the definitions that exist in this domain. This section specifically describes the concept of ETO and CTO requirements. Then, we provide the state of the art on the knowledge formalization for system configuration in section 2.2 to discuss the topics that are important in modeling knowledge for system configuration and in answering our first research question. We argue for the need to have at least two corresponding views of an artifact: a descriptive and a structural one, on the notion of commonality, as well as the principles of abstraction, generalization/specialization and inheritance. We then discuss and analyze different approaches to system configuration in section 2.3, based on eight important criteria for knowledge formalization and reuse. Finally, we summarize the elements presented in this chapter in section 2.4 and we justify our research questions.

2.1. System and system configuration

In this section, we first study system definition in section 2.1.1 and then, system configuration and configuration activity in section 2.1.2. Subsequently, we study aiding or assisting configuration with a configurator in section 2.1.3. Finally, we conclude the section in 2.1.4.

2.1.1. System definition

In this section, we clarify the notion of systems following the definitions found in the literature. (Kauffman, 1980) mentioned that “a system is a collection of parts which interact with each other to function as a whole.” (Kossiakoff et al., 2011) stated that a system is defined as “a set of interrelated components working together toward some common objectives.” (Kim, 1999) presented a system as any collection or group of parts that interact, interrelate, or depend on each other to form a unified whole that serves a specific purpose. Keeping in mind that all the parts are somehow related and interdependent. (McLucas & Ryan, 2005) mentioned that a system is made up of subsystems, while subsystems are made up of further components. A subsystem is part of a larger system that performs a specific function within the larger system. A component is a part or element of a larger system that contributes to the overall function of the system without any decomposition.

Systems can be technical (e.g., airplanes, robots, cars, computers) or non-technical (e.g., economic system, societal systems). In this thesis, we focus our work on technical systems. Technical systems refer to a collection of sub-systems and components that work together to perform a specific task or achieve a particular goal. In (Guillon, Ayachi, et al., 2021), systems, subsystems, components, services, and modules of a system are considered as artifacts. An artifact is an object that is intentionally made for a specific goal (Hilpinen, 1992, 1999). It refers

to an outcome of human activity. It can be either tangible, such as physical component, intangible, such as service, or mixed, such as servitized products, regarding the concept it represents. An artifact can be decomposed into other artifacts (Guillon, Ayachi, et al., 2021).

In this thesis, we consider systems, subsystems, or components as technical and physical artifacts, as illustrated in Figure 2. Systems can consist of subsystems and components. Subsystems can consist of additional subsystems or components, whereas a component itself cannot consist of other elements.

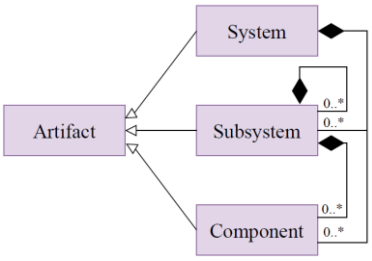

Figure 2. Artifact

2.1.2. System configuration and configuration activity

In this section, we delve into the literature on system configuration, including the definitions of system configuration, configuration activity, and configuration model. Then, we discuss customer requirements and strategies for achieving system configuration. The section concludes by mentioning configuration solutions as outputs of configuration activity.

(Mittal & Frayman, 1989) presented the first definition of product configuration, in which a product is designed after assembling some components taking into account that their connection is only possible in specific ways. (Sabin & Weigel, 1998) mentioned that product configuration can be considered a specific type of design activity that involves choosing and organizing components in a specific way to meet certain specifications. These components are usually part of a predefined set and interact with each other in predetermined ways. The act of selecting and arranging the right combination of parts is a key aspect of the configuration activity. (Soininen et al., 1998) stated that product configuration can be defined "as the problem of designing a product using a set of predefined components while considering a set of restrictions on how the components can be combined".

(Felfernig, Friedrich, et al., 2000) stated that a configuration activity can be described by specifying a set of components and their properties, such as attributes and their allowed values, connection points (ports) between them, and any constraints on the possible configurations of these components. These details provide a comprehensive understanding of the components and the conditions under which they can be legally configured to achieve the desired result. (Oddsson & Ladeby, 2014) provided a review article on product configuration which stated that a configuration activity involves combining predefined entities, which may be physical or non-physical, and determining their properties in a manner that satisfies specified requirements while also respecting constraints and following legal combinations of interfaces.

The configuration model is one of the foundations of configuration problems. According to (Soininen et al., 1998), a configuration model "specifies the entities that may appear in a configuration, their properties, and the rules on how the entities and their properties can be combined." (Forza & Salvador, 2008) defined a configuration model or product model as "a formal representation of the links between the characteristics of a product and the documents that describe each product variant". (Oddsson & Ladeby, 2014) defined a product configuration model as "an abstract representation or description, describing the structure of the product, the entities the product consists of, and the rules on how the entities and their properties can be combined." (Männistö et al., 2001) explained that configuration models are also known as generic models. The term 'generic' is used because a single model represents multiple product variants. (Männistö et al., 2001) proposed a generic product configuration model by factorizing all the information shared by all the products in a production line. (Erens & Wortman, 1996) stated that a generic product model is used to describe product families with well-defined relations between components and that these components themselves can be described as product families.

In this thesis, all the definitions outlined in papers related to product configuration are applicable in the context of system configuration. However, in system configuration, a product can be interchanged with a tangible system comprising subsystems and components, which are considered as tangible artifacts. In this thesis, we mainly use the term "generic model" instead of "configuration model". Following (Erens & Wortman, 1996), a generic model encompasses knowledge about a family of artifacts including their structure, all possible options, variants, and various relations (that will be explained in section 2.2.1).

In the configuration problem, in addition to the generic models, there are other foundations described in the following sections. The next section is dedicated to user requirements.

2.1.2.1. User requirements

User requirements are another foundation of the configuration problem. They refer to the specific needs or preferences of a user that a system must meet in order to be considered as suitable for their use. User requirements are gathered through an interactive process where the user can progressively input their preferences or needs into a configurator. The configurator then uses this information to reach a configuration solution that meets the user's requirements (i.e. a desirable configuration). There are a number of ways in which we classify user requirements. There are several ways of classifying user requirements, depending on the type of requirement (functional or technical), whether it is standard or not (included in the catalog or not), and whether it is open to negotiation or not.

Following the typology proposed in systems engineering, user needs can be either functional and operational needs or technical needs (Sage, 1992). Functional requirements define what the system must do. Operational requirements concern the way the system is operated. Technical requirements concern the way the system is built. These three needs make it possible to recognize that, depending on the type of user, novice or expert, different needs may be expressed for all or part of the system. Therefore, at least two views, a descriptive view (for

functional and operational requirements) and a functional view (for technical requirements) of the same system, must be considered.

(Pitiot et al., 2014) proposed to distinguish the most important requirements for a user, which are called "non-negotiable requirements" while the remaining requirements are called "negotiable requirements." Non-negotiable requirements are critical to the user and cannot be questioned. They must be met by the solution system to fully satisfy the user. They can be either functional, operational or technical requirements, or performance criteria such as the price or weight of a system. Negotiable requirements are not critical to the user and can be discussed. Similar to non-negotiable requirements, they can be either functional, operational or technical requirements, or performance criteria such as the price or weight of a system. This negotiable nature of requirements has to be taken into account.

(Sylla, Guillon, Vareilles, et al., 2018) defined "standard requirements" and "non-standard requirements". Standard requirements are those that are consistent with the catalog and the generic model. Taken together, these requirements result in a system that can be implemented using all the proposed components and subsystems. In contrast, non-standard requirements are not covered by the catalog and the generic model. They result from a combination of choices that have never been implemented before, or from a very specific need. This standardization of requirements must be taken into account in our proposals.

In line with the previous definitions, we propose, in this thesis, to distinguish between two types of requirements, defined as follows: CTO requirements and ETO requirements.

Definition 1: CTO Requirements
CTO requirements are standard requirements (or requirements expressed as such), whether negotiable or non-negotiable. They relate to a functional, operational or technical aspect of the system to be configured. They are therefore systematically considered during the configuration process.

Definition 2: ETO Requirements
ETO requirements are non-negotiable, non-standard requirements that affect a functional, operational or technical aspect of the system to be configured. They are therefore specifically addressed in the configuration activity by a specific design activity.

Up to this point, we have explained two foundations of configuration problems: generic models and user requirements. In the subsequent section, we will discuss various strategies that can be employed to accomplish system configuration.

2.1.2.2. Configuration strategies

System configuration can be achieved through various production strategies such as Make-To-Stock (MTS), Assemble-To-Order (ATO), Make-To-Order (MTO), Engineer-To-Order (ETO), and Configure-To-Order (CTO) which are explained in the following.

- **Make-To-Stock (MTS)** Make-To-Stock (MTS) is a production process that typically involves producing the ready-to-use system before receiving the customer's order. The customer's orders are usually fulfilled from an inventory of ready-to-use systems, which are replenished through production orders (Rudberg & Wikner, 2004).

- **Assemble-To-Order (ATO)** In Assemble-To-Order (ATO) (Wortmann et al., 1997), (Brière-Côté et al., 2010), components are kept in stock and there is a Bill of Materials (BOM) for each potential system, which is a list of subsystems and components, including their quantities, used to build that system (Hegge & Wortmann, 1991). When a customer's order is received for a particular system, the supplier initiates the assembly operations and provides the customer with a ready-to-use system, such as a computer, a car, etc.

- **Make-To-Order (MTO)** In Make-To-Order (MTO) (Wortmann et al., 1997), (Brière-Côté et al., 2010), raw materials are kept in stock and a BOM and routing exist for each potential system. Routing refers to the sequence of operations and steps required to build a system. Upon receiving an order from a customer for a specific system, the supplier can utilize the corresponding BOM and routing to initiate the manufacturing and assembly operations of the required system. This process allows the supplier to provide the customer with a ready-to-use system, such as windows, doors, etc. For such systems, the length and width of the plates are cut according to the customer's requirements.

- **Engineering-to-order (ETO)** In situations where it is not possible to meet all the customer's requirements using predefined systems, it may be necessary to develop a new system or adapt an existing one, leading to an Engineering-To-Order (ETO) situation (Wortmann et al., 1997), (Sylla, Guillon, Ayachi, et al., 2018), (Johnsen & Hvam, 2019). ETO involves the creation of customized systems based on customers' specific requirements (Brière-Côté et al., 2010), (Sylla, Guillon, Vareilles, et al., 2018). In many cases, both the finished system and its components are unique and have not been previously designed. Therefore, in ETO, the engineering phase can be partially or completely performed.

  (Sylla, Guillon, Vareilles, et al., 2018) distinguished between light and heavy ETO. The term "light" ETO is used by some companies or software providers when the requirements can be almost completely met and only minor adaptations are required. On the other hand, when many adaptations must be made to existing solutions and/or new solutions must be entirely defined, this activity is called "heavy" ETO. ETO demands significant collaboration with the customer and, like CTO, proposes the BOM and routing at the end of the configuration. There are fewer studies on ETO as for example (Brière-Côté et al., 2010), (Elgh, 2011). However, (Sylla, Guillon, Vareilles, et al., 2018) focused on the bridge between CTO and ETO. They studied the extension of configuration models usually used in CTO to ETO situations.

- **Configure-To-Order (CTO)** Configure-To-Order (CTO) is a flexible manufacturing strategy that can accommodate or support various production methods, including MTS, ATO, MTO, and partially ETO (Sylla, Guillon, Vareilles, et al., 2018). CTO allows

companies to manufacture products based on customer demand, while still maintaining a level of standardization.

Through CTO, customers can choose from a range of pre-designed system options and configurations that are then assembled or manufactured to meet their specific requirements. This strategy can enhance production efficiency, decrease lead times, and increase customer satisfaction. By leveraging CTO, companies can also gain greater flexibility and agility in their manufacturing processes, enabling them to quickly adapt to changing customer requirements and market demands.

As illustrated in Figure 3, the MTS strategy is characterized by short lead times, low product variety or diversity, and high product volume. The ATO production process is characterized by short lead times, a low product variety, and high product volume. The MTO strategy is characterized by rather long lead times, a rather high product variety, and a rather low product volume. CTO can cover MTS, ATO and MTO, and partially ETO. ETO is characterized by long lead times, a high product variety and a low product volume.

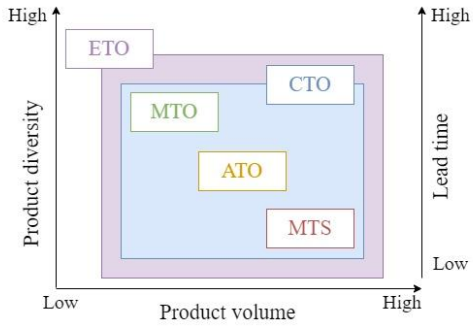

Figure 3. MTS, ATO, MTO, CTO and ETO

2.1.2.3. Configuration solutions

A configuration solution is the outcome of the configuration activity. It is a set of components and a detailed description of how they should be connected to create a product that meets all the requirements (Mittal & Frayman, 1989). According to (Sabin & Weigel, 1998), a solution must provide a list of selected components, i.e. a bill-of-materials or BOM, as well as the product's structure and arrangement. (Soininen et al., 1998) represented that a configuration specifies precisely what a real-world product instance should be like. (Oddsson & Ladeby, 2014) stated that a configuration is the result of the configuration activity, which describes the entity structure of the product and the connections between the entities in the set, meeting the given requirements.

In this thesis, we distinct two types of solutions: 1) CTO solutions which can be achieved at the end of configuration in CTO situations and 2) ETO solutions that can be reached at the end of configuration in ETO situations.

To assist or aid the users during configuration activity, it is essential to utilize a configurator. Hence, the following section is dedicated to this subject.

2.1.3. Aiding configuration with a configurator

Two types of configuration processes exist: autonomous configuration and interactive configuration (Yang & Dong, 2012), (Monge, 2019). Autonomous configuration or batch configuration involves the user providing all requirements at once and requesting a solution. Then a decision support system automatically carries out the configuration process and checks if all the requirements are consistent. Interactive configuration, on the other hand, involves the user inputting requirements one by one. After each user requirement, a specific routine processes them and removes the solutions that are no longer possible. The solution is built step by step by selecting requirements that are consistent with the set of remaining solutions. Regardless of whether the configuration is automatic or interactive, the result is a CTO solution or a statement indicating that there is no CTO solution. In this thesis, we mainly focus on interactive configuration.

According to (Janota et al., 2010) and (Vareilles, 2015), interactive configuration is a process that involves iteratively removing solutions from the solution space until reaching a solution that meets the user requirements. (Van Hertum et al., 2016), and (Falkner et al., 2020) mentioned that interactive configuration involves both a user and a configurator. The user's objective is to configure a system that meets all their requirements, while the configurator is a digital tool that assists the configuration activity by deducing the effects of the user's choices.

Configurators (Tiihonen & Soininen, 1997) are utilized to aid or assist the configuration activity and find solutions that meet the requirements of users. (Aldanondo & Vareilles, 2008), defined a configurator as a software tool that assists the person responsible for the configuration activity. It consists of a Knowledge Base (KB) that stores generic models and a Processing Unit (PU) that assists the user in the configuration activity to find a solution. The KB serves as a repository for generic models, allowing users to retrieve a relevant generic model. The PU is responsible to assist the user in the configuration activity. It may present the user with a set of allowed values (or choices) based on the generic models stored in the KB. As the user makes choices, the PU may update its allowed values to ensure that the user’s choices are consistent with the generic model.

The aim of a configurator is to ensure that the configured system is consistent with the generic model (all constraints are met) and the requirements (Aldanondo & Vareilles, 2008). (Oddsson & Ladeby, 2014) mentioned that in the literature, the three terms configurator, product configuration system, and configuration system are frequently used interchangeably and they refer to the software application. It was mentioned that a product configurator can be defined as “a configuration system, which is a software-based system that supports the user in the creation of product specifications by restricting how predefined entities (physical or non-physical) and their properties (fixed or variable) may be combined.” In this thesis, we only use the term "configurator".

Following (Janota et al., 2010) and (Vareilles, 2015), we consider interactive configuration in which a user progressively defines her/his requirements (i.e. restriction of the characteristics of systems, subsystems, or components), then the configurator removes inconsistent values with the generic model and proposes allowed ones to the user. This process continues until a unique

solution is proposed to the user. It should be noticed that in this thesis, the user is an actor of the company who need to configure systems by reusing formalized knowledge. She/he can be a designer, a representative of the customer, an expert or any person involved in system configuration.

2.1.4. Synthesis

In this section, to understand our first research question " Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?", we first established a literature review on system configuration and we defined generic models. The context of this thesis was related to technical and tangible system configuration. We considered that all definitions given for the product configuration in the literature can also be applied to the system configuration. From now on, whenever we use the term "artifacts", it refers to technical and physical (or tangible) systems, subsystems and components. Moreover, to understand our second research question "How can ETO requirements be processed during configuration activity?", we provided a literature review on configuration activity. We defined interactive configuration and different types of users' requirements. We also explained which types of requirements may arise in CTO and ETO situations.

Since in this thesis, we aim to first formalize knowledge for system configuration, the following section is dedicated to this subject.

2.2. Knowledge formalization for system configuration

In this section, we delve into the literature review on formalizing knowledge for system configuration. However, in section 2.2.1, we study the knowledge management process by briefly representing its different steps. Since knowledge formalization is one of the crucial phases in our thesis, we explore relevant topics that aid in the formalization of knowledge for system configuration. Thus, in section 2.2.2, we study the descriptive view and structural view of a system, and then we study the notation of "commonality of models" in section 2.2.3. Subsequently, we present a literature review on the principles of abstraction, generalization and specialization in section 2.2.4. Finally, we provide a synthesis of the section in 2.2.5.

2.2.1. Knowledge management process

Following (Alavi & Leidner, 2001) and (Venkatraman & Venkatraman, 2018), Knowledge management is a systematic process for extracting, formalizing, validating, storing, sharing, and utilizing knowledge within a company as illustrated in Figure 4.

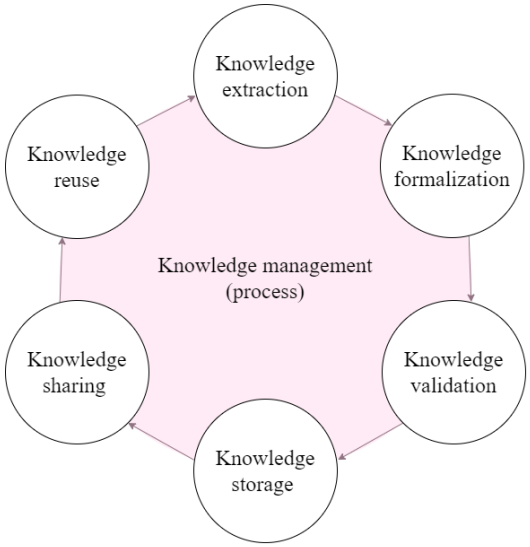

Figure 4. Knowledge management process

First of all, let's focus on the four knowledge formalization steps and see how our thesis fits in:

- Knowledge extraction is the process of identifying and capturing knowledge from various sources such as documents, databases, experts and experiments. In this thesis we assume that the available knowledge has already been identified and extracted by experts. This process is therefore naturally outside the scope of this thesis.
- Knowledge formalization is the process of transforming available knowledge into explicit, structured, and standardized forms such as rules, models, ontologies, taxonomies that can be easily managed and reused. In this thesis, knowledge formalization is the core of the first research question (RQ 1) and leads to generic model ontologies.
- Knowledge validation is the process of verifying the accuracy, completeness, relevance, and consistency of formalized knowledge through expert reviews, testing, simulations, and can involve assessing their applicability in real-world situations (verifying their reality). In this thesis, knowledge validation consists of checking the consistency of generic models.
- Knowledge storage is the process of storing formalized and validated knowledge in a central repository or database that can be easily accessed, searched, updated, and retrieved by authorized users. In this thesis, we store formalized generic models in a generic model base.

Now, let's focus on the two steps of knowledge reuse and see how our thesis fits into them:

- Knowledge sharing is the process of disseminating formalized and validated knowledge to individuals or groups who need it for their tasks or decision-making processes, through various channels such as training, mentoring, communities of practice, forums, wikis, or social media (Abdullah et al., 2008). In this thesis, the existence a generic model base and

of generic model ontologies implies that knowledge can be shared and reused to create, or enrich, new or existing generic models. In this thesis, knowledge sharing is part of the answer of our first research question (RQ 1).

– Knowledge reuse is the process of leveraging existing formalized and validated knowledge to solve new problems, by adapting, combining, or refining it as needed. In this thesis, knowledge reuse consists of the use of formalized generic models to configure systems, taking into account both CTO and ETO requirements. This process is the core of the second research question (RQ 2).

As we need to differentiate between descriptive and structural views in order to formalize knowledge, the subsequent section is dedicated to this topic.

2.2.2. Descriptive view versus structural view of a system

In the literature, researchers defined products from different views. For instance, (Jiao & Tseng, 1999) identified three views for a product family: functional, behavioral, and structural. Functional view represents the functions that the product family perform, behavioral view represents its behavior, and structural view represents its physical structure. Meanwhile, (Arana, 2007) defined three views for a generic product model: functional, technological, and physical. Functional view presents the product's main features. Technological view shows the design solutions used to meet the requirements, and physical view gives a detailed breakdown of the product's structure, like a bill of materials. (Aldanondo & Vareilles, 2008) proposed generic models based on both descriptive and physical views. The descriptive view defines the product's properties, while the physical view defines the product's physical components and their quantities. For users who lack expertise in product composition, configuring using a physical view can be challenging. To make configuration possible for these users, a descriptive view of the product is incorporated alongside the physical view. Moreover, it allows the user to make choice on configuring descriptive view or structural view. With regards to the literature, in this thesis, we define artifact families from two views: descriptive view and structural view (Figure 5). In the following, a detailed explanation of each of these views are presented.

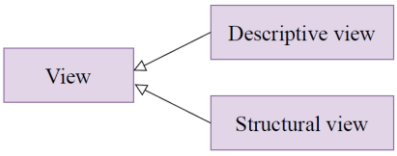

Figure 5. Different views

Descriptive view which is interesting for users focuses on the characteristics rather than physical structure (Aldanondo et al., 2003). Descriptive view only shows the key features or attributes and indicators of a family of artifacts. Each attribute and indicator have therefore a definition domain and relations are needed to describe the possible solutions. Here are their definitions:

– (Soininen et al., 1998) stated that attributes represent the characteristics of a concept or object. (Yang et al., 2012) mentioned that attributes are parametric properties of a component. (Bettman & Park, 1980) used concrete descriptive attributes**,** meaning that they

were based on reality, as well as their domains to represent an attribute-based evaluation approach. In this thesis, we follow the definition of (Soininen et al., 1998) and (Bettman & Park, 1980), and we define attributes as one of the characteristics that are used to describe the family of artifacts. Attributes can be symbolic, continuous, or discrete and their domain, which can be either a list or a range of values. Symbolic attributes are represented by a list of symbols. They are discrete as they can only take on specific numbers of distinct values. For instance, the color of a bike is a symbolic attribute with a discrete domain comprising a list of values such as {Red, Blue, Black}. Continuous attributes can take any value within a specific range or interval. They are typically represented by real numbers and have an infinite number of possible values within the defined range. For example, the weight of a bike is a continuous attribute with a domain that's an interval, such as between 10 and 30 kg. Any weight within this range is possible. Discrete attributes are similar to symbolic ones in that they have a finite domain, but they are represented by integers rather than symbols. They can be a list of integers or a set of intervals. For example, the wheel diameter of a bike is a discrete attribute, with a domain composed either of a list of integers such as {20, 21, 22, 26, 27, 28} inches, or a set of intervals such as {[20,22], [26,28]} inches.

- In addition to attributes, there are Key Performance Indicators (KPIs). (Guillon, Ayachi, et al., 2021) mentioned that KPIs can be used to compare commercial offers, and more specifically to compare systems. They divided the relevant KPIs into economic indicators, time indicators, and confidence indicators. Although KPIs are a specific type of continuous attributes with specific semantic, in this thesis, we distinguish them and we use attributes to describe the family of artifacts while we use KPIs to assess the family of artifacts. Both attributes and KPIs have domains that specify their possible values.
- In the descriptive view, in addition to the attributes and KPIs, there are relationships between them that enable the set of possible solutions to be described and evaluated. (Aldanondo et al., 2003). These relations are defined within a family of artifacts, signifying that they are internal relations linking only values of attributes and KPI (Guillon, 2019).

On the other hand, structural view which is interesting for experts focuses on the physical structure (Aldanondo et al., 2003). The structural view shows the bill-of-materials or BOM of a family of systems, i.e. the exact quantity of each item (sub-system or component) that makes up the top system. Since each item has indicators the methods to aggregate these indicators is needed to compute the ones of the top system. Moreover, relations between the items or between their attributes are required to present the allowed solutions. In follows, their definitions are presented:

- Following (Hegge & Wortmann, 1991), a BOM represents a quantified list of components, used to build a product family. These components are a family themselves. The most common approach in the manufacturing industry for modeling product structure is using a BOM, as stated by (Cao & Hall, 2020). A BOM can be single-level or multiple-level (Andersen, 1993). A single-level BOM represents one level of composition, while a multi-level BOM represents several levels of composition. In this thesis, we only focus on a single level of composition for each artifact.

- Every item within BOM has a KPI (Djefel et al., 2008). To determine the KPIs of a system, the KPIs of all items composing it must be aggregated. Each KPI requires a unique method for aggregation from a single level of composition to the system. Different functions, such as SUM, AVERAGE, MIN, MAX, etc., can be used to define an aggregation method of KPIs. In this thesis, we limit ourselves to two KPIs of Weight and Cost. Following (Djefel et al., 2008) and (Guillon et al., 2017) for these two KPIs, the aggregation method chosen is the function SUM.

- In the structural view, the main relation between items is through composition or aggregation, although other types of relations between them can exist (Arana, 2007) such as require, exclude, incompatibility, and compatibility relations. (Blecker & Friedrich, 2006) identified four categories of relations: 1) relations between distinct items, 2) relations between an item and attributes of different items, 3) relations between attributes of different items, 4) relations between attributes within an item. (Blecker & Friedrich, 2006) stated that compositional relations contain parts with assigned minimum and maximum cardinalities (to specify their numbers). The composition relations specify mandatory or optional components in the system structure (Cao & Hall, 2020). (Yang & Dong, 2013) defined three configuration rules: inclusion rules, exclusion rules, and resource rules. An inclusion rule specifies that, for a component to be included in the configuration, another component must also be present. A requisition rule is directional (Yang et al., 2012). Exclusion rules prohibit two components from existing in the same configuration. Resource relation specifies that the amount of resource consumed by components in a configuration must be less than or equal to that of the resources offered by components in the same configuration. (Felfernig et al., 2014) stated that usually compatibilities relations are used in cases where the number of allowed combinations of components is low. It represents two components needs to be in a configuration. However, in incompatibility relation they cannot be in a configuration. (Yang et al., 2012) mentioned that a port relation enforces that the corresponding ports of two components should be physically connected in a configuration. In this thesis, according to (Blecker & Friedrich, 2006), we consider two categories of relations: 1) relations between different artifacts, and 2) relations between attributes and/or KPIs of different artifacts. As illustrated in Figure 6, in this thesis, system configuration involves various types of relations including require relation, exclude relation and compatibility relation.

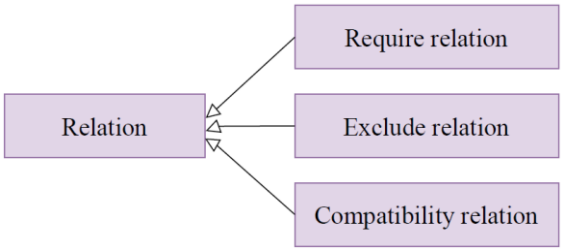

Figure 6. Different relations in system configuration

In addition to descriptive and structural views of a system, the concept of commonality of models is significant in formalizing knowledge for system and creating an ontology of generic models. The following section is dedicated to this matter.

2.2.3. Commonality of models

The purpose of this section is to provide a clear understanding of the concept of commonality using definitions sourced from literature. Our discussion begins by explaining the reasons for utilizing this particular concept.

According to (Baker, 1985), commonality in product design refers to using standardized components in designing and manufacturing multiple products within a product family or product line. By using common components, manufacturers can reduce costs, improve quality, and speed up production time, as well as enhance product flexibility and customization by allowing customers to choose different options while still using common components. Similarly, (Blecker & Friedrich, 2006) stated that through commonality, components are standardized and shared while maintaining the variety of the end products.

Several studies, such as (Siddique et al., 1998) and (Thevenot & Simpson, 2006), have focused on measuring commonality. They introduced the concept of Percent Commonality (%C), which is an index for quantifying platform commonality. According to (Siddique et al., 1998), %C is calculated by dividing the number of common components between two platforms by the total number of components in both platforms. The resulting value is then multiplied by 100 to obtain a percentage. It can be used to assess the level of commonality between different platforms. According to (Thevenot & Simpson, 2006), %C is based on three main viewpoints: (1) component, (2) component-component connections, and (3) assembly. Component viewpoint measures the percentage of components that are common between the products in the family. Component-component connections viewpoint measures the percentage of common connections between components. Assembly viewpoint measures the percentage of common assemblies between the products in the family. Each of these viewpoints results in a percentage of commonality, which can then be combined to determine an overall measurement of commonality for a platform by using appropriate weights for each item.

In this thesis, the term "commonality" is defined as the quantity of shared or common knowledge among multiple generic models whether it is their characteristics, or structure. This concept of commonality is used to create a hierarchy of generic models with different levels of abstraction.

Figure 7 illustrates the commonality of two independent generic models (number one and two). These generic models are presented by circles. In Figure 7, rectangles are used to present artifacts, curves to indicate relations between different artifacts or within an artifact, and straight lines between artifacts to indicate composition relations. The commonality of two models are identified: four common artifacts and two common relations. Each model possesses two specific artifacts exclusively dedicated to itself, along with a specific relation within an artifact (shown in bold).

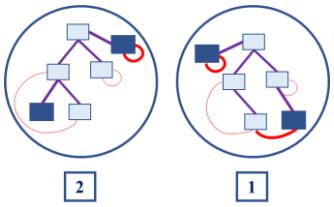

Figure 7. Example of commonality of models

To define models at different levels of abstraction we need to use generalization or specialization relationships between generic models, so the following section is dedicated to this topic.

2.2.4. Abstraction, generalization, specialization, inheritance principles

In this section, based on the literature, first, we represent the concept of abstraction levels, the principles of generalization and specialization. Then, we present the principles of inheritance which directly linked to the one of specialization. After that, we study taxonomy.

Abstraction, in general, is a fundamental concept in computer science and software engineering (Ben-Ari, 1998). The process of abstraction can also be referred to as modeling, and is closely related to the concepts of theory and design (Comer et al., 1989). Models can also be considered a type of abstraction due to their generalization of aspects of reality. In software engineering and computer science, abstraction is:

- the process of removing or generalizing physical, spatial, or temporal details (Colburn & Shute, 2007) or attributes in the study of systems in order to focus attention on details of greater importance; (Kramer, 2007) it is similar in nature to the process of generalization;
- the creation of abstract concept objects by reflecting common features or attributes of various non-abstract objects or systems under study (Kramer, 2007) - the result of the process of abstraction.

Computer science commonly presents levels of abstraction, each representing a different model of the same information and processes, but with different levels of detail. Each level uses an expressive system with a unique set of objects and compositions that apply only to a particular domain.

The idea of generalization was initially introduced in simulation programming languages, as noted by (Pedersen, 1989). (Khoshafian et al., 1991) described generalization as a bottom-up approach, wherein a set of similar classes is treated as a generic class. In generalization, numerous individual differences between classes are disregarded (Ohira et al., 2011).

Specialization, which was also first introduced in simulation programming languages, has two types: single and multiple specializations, according to (Pedersen, 1989) and (Taivalsaari, 1996). Single specialization implies that a child class has all the properties of its parent class and additional specific properties. Multiple specializations imply that a child class has all the properties of its parent classes. Specialization is a top-down approach (Khoshafian et al., 1991) that creates a hierarchy of classes, enabling the reuse of existing class to generate new classes

downwards (Ohira et al., 2011). Classes that are further away from the general class are more specialized, according to (Kamsu Foguem et al., 2008).

(Cardelli, 1984) stated that the concept of inheritance was first introduced in simulation programming languages. (Frohlich, 2002) defined inheritance informally as a mechanism that transforms an “ancestor” class into a “descendent” class by adding new features. (Taivalsaari, 1996) and (Simons, 2004) represented that inheritance allows for the creation of new classes based on existing ones. This is done by specifying the properties or characteristics that differ from the properties of existing classes, while all other properties are automatically inherited from the existing classes and incorporated into the new class (Krótkiewicz, 2018). Inheritance has several properties: transitivity, non-reflectivity, non-symmetry, and without cycles. The inheritance relationship is transitive, meaning that a parent or superclass can also be a child or subclass of another class, thus inheriting all the properties of its ancestors. Inheritance is non-reflective, meaning that if class A inherits from class B, then class B does not inherit from class A. Inheritance is non-symmetrical, meaning that if class A inherits from class B, it does not imply that class B inherits from class A. Inheritance is without cycles, meaning that there can be no circular dependencies in the inheritance hierarchy. This is also known as acyclic inheritance. An important feature of the concept of inheritance is its polymorphism. Polymorphism comes from the Greek and means that it can take several forms. Inheritance polymorphism, also known as specialization, is the ability to redefine a method in classes inheriting from a base class. It is then possible to call an object's method without worrying about its intrinsic type. This makes it possible to abstract the details of the specialized classes of an object family, masking them with a common interface (which is the base class).

(Sciore, 1989), mentioned that a set of "is a" relationships between classes forms a class hierarchy. Taxonomy is a form of classification and a fundamental mechanism for organizing knowledge (Wand et al., 1995). (Gartner, 2016) stated that taxonomy is a classification of terms or concepts of a domain organized in a hierarchical structure. (Van Heijst et al., 1997) explained that an ontology for a specific domain describes a taxonomy of concepts that define the semantic interpretation of the knowledge. (Kamsu Foguem et al., 2008) noted that since ontology is the heart of any knowledge description, ontological objects are typically described as a set of concepts and a set of relations between them. These sets can be ordered to form a taxonomy of concepts. (Nickerson et al., 2013) proposed a method to develop a taxonomy in an information system domain, emphasizing the importance of taxonomies in understanding and analyzing complex domains. (Guillon, Villeneuve, et al., 2021) defined a taxonomy as a specific type of ontology because it allows for structuring the knowledge. (Tumnark et al., 2019) mentioned taxonomy only includes "is a" relationships, while ontology includes cardinality and other restrictions.

In this thesis we combine the principles of generalization, specialization, inheritance, taxonomy and commonality to structure and formalize knowledge at different levels of detail or abstraction. This formalized knowledge can then be reused to generalize, specialize and create a taxonomy of generic models. By organizing knowledge at different levels of abstraction, the process of formalizing knowledge does not require starting from scratch. In addition, any change made to a generic model at a particular level of abstraction in the taxonomy is

propagated to all generic models at lower levels of abstraction through the principles of inheritance and specialization. By creating generic models at a lower level of abstraction, it is possible to focus only on their specific knowledge. The principle of inheritance polymorphism allows experts to specialize high-level knowledge at the level in question.

For instance, as illustrated in Figure 8, generic models at varying levels of abstraction are created through generalization and specialization. Initially, two independent generic models (number one and two) are created. As explained in Figure 7, their commonality is identified (i.e. four artifacts and two relations). Then, using generalization, a new generic model is created at a higher abstraction level (number three) which only possess the commonality of models. In addition, using specialization, a new model (number four) is created at a lower abstraction level. This new model comprises both the inherited knowledge from model number three, as well as its own specific knowledge (i.e. three artifacts and three relations shown in bold).

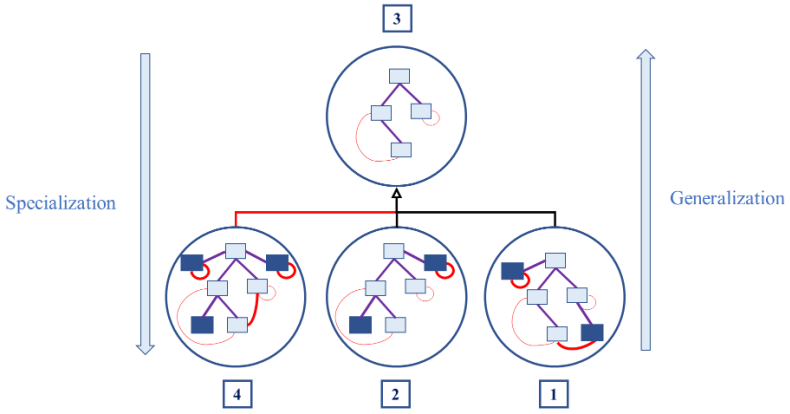

Figure 8. Example of using generalization and specialization

In the following, the synthesis of this section is presented.

2.2.5. Synthesis

In this section, we began by introducing the knowledge management process, followed by an outline of two essential phases: knowledge formalization and knowledge reuse. We mainly focused on concepts related to our first research question "Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?". Therefore, we provided a literature review on descriptive and structural views. Additionally, we delved into the concepts of: commonality of models, abstraction, generalization and specialization which play important role in knowledge formalization.

2.3. Different approaches for system configuration

This section is dedicated to the various approaches that can be used for system configuration, including the ontology approach, Unified Modeling Language (UML), Systems Modeling Language (SysML), Constraint Satisfaction Problem (CSP), Case-based reasoning (CBR), and hybrid approaches. In order to compare these different approaches, we have developed a list of eight criteria necessary for effective formalization and reuse of generic models. For each approach, we provide a definition and an explanation of its different elements. We then evaluate

them against our eight criteria and conclude with their advantages and disadvantages, distinguishing between knowledge formalization and knowledge reuse.

We therefore begin by defining our eight evaluation criteria, then present the ontology-based approaches, the UML and SysML approach, the CSP, the CBR and then the methods that hybridize the previous approaches. We conclude this section with a table summarizing our analysis.

2.3.1. Criteria for comparing different system configuration approaches

Based on the literature on system configuration and the needs expressed by experts and users, we have drawn up the following list of eight essential criteria for establishing, maintaining and updating generic models, and for using them effectively. Four criteria are dedicated to the knowledge modeling phase and four to the knowledge reuse phase, as synthesized in Table 1. For each criterion, the response options are as follows: Yes, No or Difficult.

In knowledge formalization, the criteria are:

- **KFC1: Clearly distinguish between descriptive and structural views:** The studied approach allows for a clear formalization of descriptive view and structural view. As explained in section 2.2.2, the descriptive view details the identity of the family of artifacts by providing an overview of its key attributes and KPIs, regardless of its structure. On the other hand, the structural view reveals the composition of the family of artifacts. The need to separate descriptive view from structural view arises from different interests and levels of expertise of the person who is in charge of configuration in knowledge reuse step. It is therefore essential to create this distinction between descriptive and structural views in knowledge formalization for system configuration.

- **KFC2: Better structure knowledge for later reuse in modeling:** The studied approach makes it possible to easily reuse the formalized generic models either partially or entirely in order to formalize new ones. As explained in section 2.2.4, we need to structure knowledge for system configuration to avoid the repetitive and time-consuming process of knowledge formalization, knowledge maintenance and knowledge update. A taxonomy of knowledge seems to be a good way to reduce the workload of experts. The different approaches will therefore be evaluated on this particular point.

- **KFC3: Explicitly formalize generic models at different levels of abstraction:** The studied approach enables to take advantage of the commonality of generic models in order to formalize generic models at a higher level of abstraction using generalization. Reversely, the studied approach enables to use inheritance principles in order to formalize generic models at a lower level of abstraction using specialization. Consequently, the approach makes it possible to formalize generic models at different levels of abstraction. As mentioned in section 2.2.4, we need to formalize generic models at different levels of abstraction in order to better manage knowledge for system configuration. Any modifications made at a generic model at

a more abstraction level is propagated to all generic models at lower abstraction level. It therefore enables distributing changes throughout the model hierarchy.

- **KFC4: Unquestionably validate the consistency of the knowledge:** The studied approach allows to validate the generic models during their formalization and guarantee the consistency of the solution space (i.e. the set of all possible solutions that these models can generate). As mentioned in section 2.2.1, we need to validate the consistency of the knowledge since it ensures that the knowledge embedded in our models is reliable, and does not contain contradictory information. This is necessary, as inconsistencies can threaten the integrity of the models and lead to no results.

In knowledge reuse phase, the criteria are:

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements:** The studied approach makes it possible to configure an artifact according to either its descriptive view or its structural view. Consequently, the studied approach enables to make this choice during the configuration activity (configuring only descriptive view or configuring both descriptive view and structural view). As explained in section 2.2.2, this need is required since it enables the users who are interested only in the attributes of the artifact, to configure its descriptive view and it enables those who are interested in the structure of the artifact to configure its structural view. Therefore, configuring both views allows the model to be applied more broadly, to meet a wide range of user requirements.

- **KRC2: Formalize and capitalize ETO requirements during configuration:** The studied approach enables to formalize ETO requirements during configuration activities to configure artifacts and capitalize solutions for further reuse. As explained in sections 2.1.2.1 and 2.1.2.2, the need to formalize and capitalize ETO requirements during configuration arises from the necessity to address ETO requirements. The different approaches will therefore be evaluated on this particular point.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements:** The studied approach makes it possible to configure an artifact with its sub-systems and/or components at different levels of abstraction. It enables to decide at the beginning of the configuration activity at which level of abstraction the artifact has to be configured. As mentioned in KFC3 and regarding the explanations of section 2.2.4, generic models are created at different levels of abstraction. Therefore, the need of configuring according to several abstraction level arises from the complex nature of knowledge models and the varying requirements of users. Not all users require or benefit from the same level of detail. Some may require an artifact at a high-level of abstraction, focusing on common knowledge. Other users may need an artifact at a lower level of abstraction, containing much more specific and detailed knowledge.

- **KRC4: Interactively configure a system:** The studied approach makes it possible to interactively configure an artifact. Consequently, the studied approach enables to define a requirement then remove inconsistent solutions from solution space and repeating it iteratively until obtaining a solution. As explained in section 2.1.3, this need is required since interactive configuration allows users to actively shape the solution, making decisions throughout the process. Through the iterative refinement of the solution space, where inconsistent solutions are progressively eliminated, the system converges towards a solution that aligns closely with the defined requirements. This gives dynamism and adaptability to the configuration activity and enables us to respond to changing requirements.

Table 1. Requirements

Requirements for system configuration
KFC1: Clearly distinguish between descriptive and structural views
KFC2: Better structure knowledge for later reuse in modelling
KFC3: Explicitly formalize generic models at different levels of abstraction
KFC4: Unquestionably validate the consistency of the knowledge
KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements
KRC2: Formalize and capitalize ETO requirements during configuration
KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements
KRC4: Interactively configure a system

In the upcoming section, we will discuss various approaches that can be utilized to meet the needs for formalizing and reusing knowledge in system configuration, which we have presented earlier.

2.3.2. Ontology and system configuration

In this section, we begin by reviewing the ontology used in systems configuration. Next, we will explore the application of this ontology in the context of systems configuration. Finally, we will evaluate this approach on the basis of our predefined criteria.

2.3.2.1. Ontology definition

According to (Gruber, 1993), an ontology is "*an explicit specification of a conceptualization*". Other researchers, such as (Gruber, 1995), and (Borst et al., 1997) have expanded upon this definition. Then, (Studer et al., 1998) defined an ontology as "a *formal, explicit specification of a shared conceptualization*".

Several key elements are typically included in an ontology such as classes (or concepts), individuals or objects, and relations.

Ontology Classes: Classes are the primary formalized elements of the domain (Reyes-Peña & Tovar-Vidal, 2019). (Noy & McGuinness, 2001) mentioned that for modeling class hierarchies, there are three approaches: top-down, bottom-up, and middle-out. The top-down approach involves starting with the most general class and then refining it to more specific classes. In contrast, the Bottom-up approach is starting with the most specific classes and moves towards general ones. The middle-out approach involves starting with the most crucial classes and then proceeding to either more general or more specific classes.

Ontology Individuals: Individuals are the representation of the main objects within the domain (Reyes-Peña & Tovar-Vidal, 2019).

Ontology Relations: Relations are links between the classes (Reyes-Peña & Tovar-Vidal, 2019). For instance, generalization or specialization relations between classes and sub-classes help to organize the classes in a hierarchical structure (Hadzic et al., 2009), known as a taxonomy. In addition, relations between two classes allow modeling aggregation or composition. While other types of relations in an ontology include Associates with, Instance of, etc. (Hanafi et al., 2018).

Ontology Development Process: (Noy & McGuinness, 2001) proposed that the process of developing an ontology involves seven steps: (1) determining the domain and scope, (2) reusing previous ontologies if they exist, (3) developing a terminology, (4) developing classes and class hierarchies, (5) defining properties, (6) defining property constraints, and (7) defining individuals. After completing these steps, it is necessary to evaluate the ontology to ensure it works properly through testing reasoning algorithms, checking the consistency of classes and properties, and validating inferences.

Reasoning refers to the process of automatically inferring or determining new knowledge that has not been explicitly stated from the existing information represented in the ontology (Riboni & Bettini, 2011). It helps to automatically deduce new conclusions from the formalized knowledge expressed in the ontology. Reasoning in the ontology can be used for tasks such as determining the relationships between different concepts in the ontology, checking

inconsistencies in the ontology (Riboni & Bettini, 2011), and answering queries about the knowledge represented in the ontology.

The specific reasoning capabilities of an ontology depend on the formalism used to represent the ontology and the reasoning tools that are available for that formalism. Rule-based reasoning involves using a set of rules to infer new knowledge from the information represented in the ontology (Zhai et al., 2018). The relations are typically specified in a logical language, such as SWRL[2] (Semantic Web Rule Language), and can be used to infer relations between classes, perform classifications, and identify inconsistencies in the ontology.

SWRL provides powerful deductive reasoning capabilities (Zhai et al., 2018). A SWRL rule consists of two parts: an antecedent (body) and a consequent (head). Both parts consist of zero or several atoms separated by an arrow "→". For instance, (Atom ^ Atom ^ …) → (Atom ^ Atom ^...). The consequent of an SWRL rule is triggered if and only if every atom in the antecedent is satisfied. Because the antecedent can be satisfied multiple times, SWRL rules support iteration and fire for every combination of values that satisfy the antecedent. It means that we cannot change values or remove inconsistent ones, we can only add new properties. Therefore, SWRL does not support non-monotonic reasoning (DeBellis, 2021). For more information refer to (Van Harmelen et al., 2008).

2.3.2.2. Ontologies for system configuration

Ontologies have been used in the domain of system configuration. (Soininen et al., 1998) proposed a general ontology that includes modeling concepts to represent knowledge in the domain of configuration. This ontology allows structuring and representing knowledge in this domain. Two review articles on product configuration, (L. L. Zhang, 2014) and (Oddsson & Ladeby, 2014), presented various definitions in this field and discussed prospects. (Ming et al., 2017) mentioned that ontology not only captures and documents information but also facilitates reusing the previous information to effect new decisions when requirements change. (Lyu et al., 2017) is a review paper that mentioned that research on ontology-based product modeling mainly focuses on how product knowledge in a domain can be represented and extracted in a formalized way, and then interpreted and reused in other domains.

(Yang et al., 2008) and (Yang et al., 2009) proposed an ontology-based approach to formalize knowledge for product configuration. They used Web Ontology Language (OWL) to define knowledge about the product and its components and Semantic Web Rule Language (SWRL) to define the constraints. Then, the configuration system is implemented using Java Expert System Shell (JESS). The proposed approach was applied to the configuration of a ranger drilling machine in the first paper and it was applied to a case for the personal computer in the second paper. Similarly, (Dong et al., 2011) and (Shen et al., 2012) developed an ontology-based approach to model knowledge for service product configuration. They used OWL, and SWRL to model knowledge and then employed the JESS rule engine to implement configuration processes.

[2] https://www.w3.org/Submission/SWRL/

(Xuanyuan et al., 2016) proposed a rule-based ontological formalism to represent the product structure and constraints of a product configuration. In this way, OWL and SWRL were used; as well as, the Jess reasoner to validate the consistency of the ontology and to develop valid product configurations. (Zhou et al., 2017) proposed a method to improve cutting tool configuration while reducing carbon emissions for the machining processes. The method used an ontology-based approach considering carbon emissions, SWRL language to create rules to reason feasible cutting tool configurations, and an evaluation method to find the optimal configuration and applied it on a vortex shell workpiece.

(Cao & Hall, 2020) proposed an ontology-based method to configure modular buildings. It used SWRL and Semantic Query Enhanced Web Rule Language (SQWRL) to formalize configuration constraints such as composition, cardinality, compatibility, and dependency. (Esheiba et al., 2021) proposed a hybrid knowledge-based recommender system that makes use of ontologies to capture knowledge about customer requirements, products, services, and production. As well as constraint programming for encoding the variants of product-service systems. The case study focused on the laser machine domain.

2.3.2.3. Advantages and drawbacks of using ontology

In this section, we assess the ontology based on our eight criteria. For each criterion, there are three possible outcomes: Yes, No and Difficult. We justify our answers by referring to the literature review.

In knowledge formalization, the criteria for ontology are evaluated as follow:

- **KFC1: Clearly distinguish between descriptive and structural views: YES**

In essence, the key to distinguishing between descriptive and structural views in an ontology lies in the systematic and structured definition of classes, attributes, relations, and hierarchies. Ontologies allow modeling knowledge about different artifacts, their characteristics (such as attributes), KPIs and the relation between them (descriptive view). Moreover, ontology allows to formalize relationships (such as composition). Other relations such as require and exclude as well as KPIs aggregation methods can be formalized within an ontology by means of rules (structural view). However, ontologies have difficulties in representing all relations required for formalizing knowledge in system configuration, such as compatibility between artifacts or their characteristics.

In (Soininen et al., 1998), (Yang et al., 2008), (Yang et al., 2009), (Dong et al., 2011) and (Shen et al., 2012), they formalized knowledge in the domain of configuration, separating descriptive and structural views. First, they used OWL to define knowledge about the product and its components and then SWRL to define the constraints.

- **KFC2: Better structure knowledge for later reuse in modeling: YES**

Ontologies are designed to structure knowledge in a standardized and reusable format. The formal definitions and hierarchies established, such as taxonomies, make them well-suited for modeling across various contexts. Their inherent taxonomic structure enables systematic organization and ease of reuse.

In the study by (Yang et al., 2008), a methodology was introduced that organizes configuration models hierarchically. This structure enables deriving domain-specific configuration knowledge by reusing or inheriting classes. The base of this methodology is a configuration meta-model from which specific domain knowledge can be derived.

Another study by (Yang et al., 2009) presented a comprehensive configuration ontology model that defines the fundamental terminologies and relationships relating to the product configuration domain. The paper demonstrates the derivation of domain-specific configuration models through ontology inheritance from the general model, enabling the accurate representation of specific product configuration knowledge.

- **KFC3: Explicitly formalize generic models at different levels of abstraction: YES**

Hierarchical nature of ontology allows for different levels of abstraction. Ontologies allow modeling knowledge about artifacts at different levels of abstraction. Using ontology, the generalization or specialization relations between classes and sub-classes can be defined in order to formalize general to specific classes, allowing for varying levels of detail (Hadzic et al., 2009). These class hierarchies referred to as taxonomies can be modeled using top-down or bottom-up approaches (Noy & McGuinness, 2001), (Polenghi et al., 2022). The top-down approach consists in starting with the most general class, then refining it to obtain more specific classes. The bottom-up approach, on the other hand, starts with the most specific classes and works towards the general classes.

- **KFC4: Unquestionably validate the consistency of the knowledge: DIFFICULT**

To validate the consistency of knowledge within an ontology, reasoning tools are employed. These tools can infer new knowledge and highlight inconsistencies based on the established logical rules and relationships. Using OWL, which is based on Description Logics, configuration models gain clear logical semantics, as highlighted by both (Yang et al., 2008) and (Yang et al., 2009). This permits reasoning and the detection of inconsistencies within knowledge bases, including checking for class subsumption and concept inconsistency. However, ensuring the unquestionable validation of ontology consistency can be challenging. As explained previously, ontologies have difficulties in representing compatibility relations between artifacts or their characteristics. Some rules can be formalized within an ontology and applied using a reasoner. However, that does not allow to prohibit inconsistent attribute values and thus, it is not that easy to check the consistency of the formalized knowledge using ontology.

In knowledge reuse, the criteria for ontology are evaluated as follow:

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements: YES**

Using ontologies, it is possible to reuse and configure formalized configuration models. For instance, (Yang et al., 2008) and (Yang et al., 2009) used an inference engine (i.e. JESS rule engine) in their study to carry out configuration process. They defined user requirements in the form of constraints, such as constraints on characteristics (or attributes) of a component. At the

end of configuration, solutions were obtained consists of the component individuals, the assignment of values to properties of these individuals and the connection relations among components. These solutions satisfied all constraints and customer requirements. To the best of our knowledge, no study in the literature has addressed configuring both descriptive and structural views and providing solutions for each.

- **KRC2: Formalize and capitalize ETO requirements during configuration: YES**

While ontologies are adept at capturing general knowledge structures, ETO requirements can be intricate and highly specific. This might necessitate ontology manipulation or extensions. We did not find any paper related to this criterion. However, to our knowledge, after creating an instance of model according to what the user wants, we can add attributes and values within the instance manually. Therefore, we can formalize and capitalize ETO requirements during configuration.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements: YES**

As explained before, ontologies allow modeling knowledge about artifacts at different levels of abstraction. It enables to decide at which level of abstraction the artifact has to be configured. Therefore, given the hierarchical structure of ontologies, it is possible to focus on different levels of abstraction. Parent classes offer higher abstraction levels, while child classes provide lower abstraction level containing detailed knowledge. We did not find any paper in the literature which directly addresses this criterion. However, as far as we know, a class can be selected either at higher abstraction levels or lower abstraction levels, then its instance can be created. Then, it can be configured using ontologies.

- **KRC4: Interactively configure a system: DIFFICULT**

Interactive configuration requires rapid interaction with user. While ontologies provide structured knowledge, rapid interactions, especially with vast ontologies to fulfill different requirements, is not that easy. Therefore, it is difficult to reason on the formalized knowledge and solve the configuration problem interactively using ontology. Although some rules can be formalized within an ontology and applied using a reasoner. However, that does not allow to prohibit inconsistent attribute values and thus, it is not possible to configure systems.

In conclusion, ontology seems to be particularly adept at formalizing knowledge, distinguishing between descriptive and structural views, and facilitating knowledge structuring for future reuse in modeling. Its hierarchical nature allows for flexibility in levels of abstraction, making it suitable for various modeling purposes. While ontologies are intrinsically structured, they may face challenges in areas like interactive configurations.

The following section is devoted to the Unified Modeling Language and Systems Modeling Language.

2.3.3. UML, SysML and system configuration

In this section, we commence with an examination of the ontology employed in systems configuration. Following that, we delve into the utilization of this ontology within the realm of systems configuration. Lastly, we assess the effectiveness of this approach based on our predetermined criteria.

"In this section, we initiate our examination by assessing the ontology applied in systems configuration. Subsequently, we delve into the practical implementation of this ontology within the framework of systems configuration. Lastly, we assess the effectiveness of this approach against our predetermined criteria."

In this section, we commence by examining the ontology employed in systems configuration. Following that, we will delve into the utilization of this ontology within the domain of systems configuration. Ultimately, we will assess this approach based on our predetermined criteria.

2.3.3.1. UML definition

The definition of Modeling Language (ML) has been represented by (Rumbaugh et al., 1999) as a language that focuses on the conceptual and physical representation of a system. Conceptual representation provides an abstract or high-level representation of a system's components, relationships, and interactions to understand its functionality and behavior without getting into implementation details. Physical representation, on the other hand, is the concrete or low-level representation of a system's components. Unified Modeling Language (UML) is the most widely used object-oriented modeling language, as noted by (Petre, 2013), (Hutchinson et al., 2014). (Clark & Evans, 1997) defined UML as a collection of graphical models that express the properties of an object-oriented design. (Rumbaugh et al., 1999) stated that UML is "*a graphical language for visualizing, specifying, constructing, and documenting the artifacts*[3] *of a system*".

UML diagrams: UML provides various diagrams that represent a system using 14 diagrams, grouped into two categories, allowing for the modeling of both its structure and behavior (Clark & Evans, 1997), (Language et al., 1997), (Rumbaugh et al., 1999), and (Mkhinini et al., 2020). Structural diagrams, such as class diagrams, depict the static elements of a system and their relationships (Clark & Evans, 1997). Meanwhile, behavior diagrams show the dynamic behavior of objects in a system and changes over time (Clark & Evans, 1997), (Rumbaugh et al., 1999). The UML class diagram is commonly used (Dobing & Parsons, 2006), and it is the most important structural model (Clark & Evans, 1997). UML use case diagrams offer a high-level overview of the system's functionality from the user's perspective, with each use case representing a specific functionality that the system must deliver to its users[4].

A UML class diagram has different elements including classes, attributes, and relationships. A class is a group of objects that share attributes, operations, relationships, and semantics (Wesley, 2015). An attribute is a property or characteristic of a class. UML has three types of

[3] An artifact in software development is an item created or collected during the development process. Example of artifacts includes use cases, requirements, design, code, executable files, etc.
[4] https://en.wikipedia.org/wiki/Use_case_diagram

relations: association, generalization, and aggregation; Association represents a structural relationship between two classes, showing how one class is related to another one. Generalization represents a "is a" relationship between classes, where a child inherits the structure and behavior of a parent (Clark & Evans, 1997), (Rumbaugh et al., 1999). Aggregation is a special type of association, representing a structural relationship between a whole and its components or parts (Rumbaugh et al., 1999).

UML is a modeling language, not a reasoning or inference engine. UML does not support automated reasoning or inference. However, it can be used in combination with other tools that do support monotonic reasoning, such as Object Constraint Language (OCL) (Omg, 2012) which is a modeling language to specify and validate constraints on the model. These constraints can then be used to check the consistency and correctness of the model. This enables the modeler to reason about the system being modeled, identify potential issues, and ensure that the model adheres to specified requirements and constraints. (Queralt & Teniente, 2006) proposed a method for reasoning on structural conceptual schemas specified in UML with OCL integrity constraints. This approach contains two steps: 1) translating the UML class diagram and the OCL constraints into a first-order logic representation, and 2) using the CQC Method, which performs constraint-satisfiability checks, to carry out the reasoning and validation tasks. (Berardi et al., 2005) addressed the challenge of reasoning on UML class diagrams and showed that it can be quite a complex task. (Pérez & Porres, 2019) proposed a framework for reasoning based on Constraint Logic Programming.

2.3.3.2. SYsML Definition

The Systems Modeling Language (SysML) is another modeling language particularly designed for systems engineering applications. It is a standard from the Object Management Group (OMG), developed in March 2003. It allows the representation of systems and product architecture, their behavior, and functionalities (Balmelli, 2007). SysML supports the specification, analysis, design, verification, and validation of complex systems (Hause, 2006).

SysML diagrams: SysML offers a wider variety of diagrams than UML, including diagrams such as requirements, parametric, and allocations diagrams, which are not available in UML (Balmelli, 2007). SysML diagrams can define system requirements, behavior, structure and parametric relationships (Hause, 2006). (Guillon, 2019) mentioned the system structure can be represented by block definition diagrams and internal block diagrams, while the behavior diagrams include the use case diagram, activity diagram, sequence diagram and state machine diagram. The requirement diagram captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships, while the parametric diagram represents constraints on system parameter values such as performance, reliability and mass properties to support engineering analysis. As pointed out by (Fiorèse et al., 2012), the functional block diagram of SysML shows the set of functions that the product to be designed must fulfill. There are four behavioral diagrams formalized in the SysML: the use case diagram, the sequence diagram, the activity diagram and the state diagram. Modeling the behavioral view of the product is particularly relevant during the complete design of the product, if the system is accepted by the customer. Physical block diagram of SysML shows the internal structure of the block, i.e. its decomposition into physical components (Guillon, 2019).

2.3.3.3. UML and SysML for system configuration

UML and OCL can be employed in the domain of system configuration. (Felfernig, Jannach, et al., 2000) proposed an approach using UML to define a product configuration knowledge base. (Felfernig, Friedrich, et al., 2000) defined logic-based formal semantics for UML constructs, allowing to generate logical sentences and to process them by a problem solver. (Felfernig, 2007) represented configuration knowledge and built configuration models using UML and OCL as standard configuration knowledge representation languages. This standard representation of configuration knowledge facilitates the integration of configuration technologies into software environments managing complex products and services. Configuration systems supporting this standard representation are easier to integrate and improve the technological support for implementing a company's mass customization strategy. (Felfernig et al., 2014) used a graphical configuration knowledge representations approach that is UML to formalize configuration models relying on the notation of class diagrams; without the need for additional notations such as component or sequence diagrams. (Rigger et al., 2021) proposed a method to formalize knowledge for product configuration during the development of an engineering system using SysML.

2.3.3.4. Advantages and drawbacks of using UML and SysML

In this section, we provide the advantages and drawbacks of utilizing UML and SysML for knowledge formalization, as well as for knowledge reuse.

- **KFC1: Clearly distinguish between descriptive and structural views: YES**

UML and SysML are a powerful approach for formalizing knowledge about systems. They provide a set of diagrams to illustrate various aspects of systems, from their architecture and structure to their behavior and interactions.

In a study by (Felfernig et al., 2014), they used UML to create a model for configuring personal computers (PCs). They used class diagrams to define the PC's structure, including its parts and how they fit together. Some basic relations about how these parts can be combined were directly included in the model, others were represented textually.

In another study by (Rigger et al., 2021), they employed SysML to build a configuration model. They showed how to create product designs and define relations within these designs. They did this by adding variables and constraints directly into the product design using diagrams.

- **KFC2: Better structure knowledge for later reuse in modeling: YES**

In (Felfernig et al., 2014), through UML's class diagrams, one can create a taxonomy by hierarchically organizing classes, thus better structure knowledge that can be reused in different models.

In (Rigger et al., 2021), a hierarchical component structures is created using SysML which facilitate the reuse of components for formalization of configuration models. In their work, they created a package named “product architecture” that holds details about the system's hierarchy and the blocks that are used.

- **KFC3: Explicitly formalize generic models at different levels of abstraction: YES**

In the context of modeling, UML and SysML provide a valuable framework for creating generic models at various levels of abstraction. This flexibility allows modelers to begin with high-level system concepts and gradually refine them into more detailed concepts.

In (Felfernig et al., 2014), UML is used to formalizing specialization and generalization relationships. They defined a class as a child of another class, with the child class inheriting all attributes and behaviors from its parent class. Additionally, (Rigger et al., 2021) illustrate how SysML offers generalization relationships between blocks and allows for the definition of abstract elements.

- **KFC4: Unquestionably validate the consistency of the knowledge: DIFFICULT**

UML intrinsically lacks reasoning capabilities essential for validating the consistency of a model. To add constraints to UML models and reasoning on them, auxiliary tools like the Object Constraint Language (OCL) can be employed.

(Felfernig, 2007) explains how OCL helps represent relations in customizable products and services using UML and then check the consistency. Similar to UML, SysML does not inherently possess capabilities to validate the consistency or accuracy of the represented knowledge.

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements: NO**

UML and SysML have limitations when it comes to configuration of models based on specific requirements. Using them alone, configuration process cannot be performed directly. The models created using UML and SysML are static in nature, meaning that while they can illustrate different variants, or configurations, they cannot be configured based on requirements.

- **KRC2: Formalize and capitalize ETO requirements during configuration: YES**

There is no paper in the literature about this criterion. However, to our knowledge, it is possible to formalize ETO requirements using UML and SysML. After creating an instance of the formalized configuration model, a new attribute, a new value of attribute or a new relation between the classes can be added manually.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements: NO**

While UML and SysML enable experts to formalize knowledge about systems at varying levels of abstraction, they inherently cannot configuration the model based on varying configuration requirements. There is no paper in the literature about this criterion.

- **KRC4: Interactively configure a system: NO**

UML and SysML do not inherently support interactive configuration. When using UML and SysML, systems can be modeled to show different configurations, but these formalized models cannot be interactively configured based on user inputs.

In conclusion, UML and SysML appear to be powerful approaches for formalizing knowledge. Their intrinsic ability to distinguish between descriptive and structural views, creating models at different levels of abstraction, and structuring knowledge in a taxonomy makes them suitable for modeling systems. However, challenges arise when it comes to checking consistency of formalized knowledge, configuring models at different levels of abstraction (either descriptive or structural) and specifically interactive configurations. In this PhD, both knowledge formalization and knowledge reuse are considered, therefore UML and SysML are not selected since they cannot fulfill our requirements related to knowledge reuse.

The following section is dedicated to the constraint satisfaction problem approach.

2.3.4. CSP and system configuration

In this subsection, we first recall the constraint satisfaction problems commonly used in system configuration (Felfernig et al., 2014). Second, we see how CSP can be used for system configuration. We then go on to evaluate this approach with respect to our criteria.

2.3.4.1. CSP definition

A Constraint Satisfaction Problem, notated CSP, is an approach to formalize knowledge and reason on it to find solutions compatible with the problem. The definition of constraint satisfaction problem was firstly proposed by (Montanari, 1974), in which, a CSP is defined as a triplet {X, D, C} where:

- $X = \{x_1, x_2, \ldots, x_k\}$ is a finite set of variables,
- $D = \{d_1, d_2, \ldots, d_k\}$ is a finite set of domains - one for each variable of X,
- $C = \{c_1, c_2, \ldots, c_m\}$ is a finite set of constraints on variables. Constraints represent restrictions on the combination of variables values.

CSP Variables: Variables are objects that can take on a variety of values (Tsang, 1993). The domain of a variable is a set of all possible values that can be assigned to the variable (Tsang, 1993). The variables can vary in their types, including symbolic or numerical, discrete or continuous (Vareilles, 2005). :

- Symbolic variables are represented by a list of symbols. Symbolic variables are discrete variables, since they have a finite domain.
- Numerical variables are represented by a list of integers, a list of reals, intervals of integer values, or intervals of reals. Numerical variables can be discrete (having a finite domain or a set of discontinuous intervals), or continuous (having a set of continuous intervals).

CSP Constraints: Constraints on a set of variables restrict the combination of values that these variables can take simultaneously (Tsang, 1993), (Ghedira, 2013). In a CSP, compatibility constraints and activation constraints can be distinct (Vareilles, 2005), (Rossi et al., 2006), as can global constraints (Tsang, 1993). They are explained respectively as follows.

Compatibility constraints define the possible combinations of values between several variables (Mittal & Falkenhainer, 1990), (Vareilles, 2005). (Tsang, 1993) also mentioned compatibility constraints are described by lists of combinations of allowed or forbidden values. Compatibility constraints can be formalized in the form of tables or mathematical (or numerical) functions (Vareilles, 2005). As explained in (Vareilles, 2005), (Rossi et al., 2006), (Monge, 2019), compatibility tables represented in tabular form, the explicit list of authorized values to consider and numerical functions represented in a mathematical form, the implicit allowed combinations of variable values.

Activation constraints modify the structure of the solution space by adding or removing variables or constraints from the current problem (Mittal & Falkenhainer, 1990). As proposed by (Mittal & Falkenhainer, 1990), there are four types of activation constraints: Require, Always require, Require not, and Always require not. (Mittal & Falkenhainer, 1990) defined them as follows. The fundamental activity constraint is "require" which establishes a variable's activity through the assignment of values to a set of active variables. The "always require" constraint expands on the require constraint by requiring a variable's activity based on the activity of other variables, regardless of their current value. The "require not" constraint specifies an inconsistency between an assignment of values to a set of active variables and the activity of another variable. Similarly, the "always require not" constraint extends the concept of the require not constraint by precluding a variable's activity based on the activity of other variables, regardless of their present value.

The term global constraint appeared in the late 1980s (Beldiceanu et al., 2007). Global constraints are considered as classes of constraints that are defined using a formula of arbitrary arity (a formula that involves any number of variables). On the other hand, (Van Harmelen et al., 2008) pointed out a global constraint is a constraint over a sequence of variables. The use of global constraints makes it easier to construct a CSP model. (Tsang, 1993) highlighted that the all-different constraint, which requires that all variables in the constraint must be unique, is a classic example of a global constraint. (Beldiceanu et al., 2005) presented a catalog of global constraints. (Simonis, 2007) stated that global constraints help model complex problems as they provide high-level constraint abstractions that simplify the creation of large-scale models.

CSP Solution: A solution of a CSP is a complete instantiation of all the active variables satisfying all the constraints (Barták et al., 2010). An instantiation is an assignment of values to the variables, i.e. each variable as one and only one value. There are two types of instantiation: partial and complete. In a partial instantiation, only a subset of variables is instantiated, the solution is under construction.

CSP processing: Resolving a CSP involves various solving techniques and filtering techniques. Solving techniques are capable of systematically exploring the entire search space and producing providing all solutions to a problem. The Backtrack algorithm is the most widely

used basic search technique, first introduced by (Golomb & Baumert, 1965). This technique employs a depth-first strategy, utilizing a backtracking technique that returns to the previous state if the current partial assignment is found to be inconsistent. In contrast, incomplete solving techniques do not thoroughly explore the search space and instead rely on an opportunistic exploration of the set of complete assignments. These techniques only produce a subset of solutions and require an assignment evaluation and comparison function. Examples include the Tabu search technique by (Glover & Laguna, 1993), and Simulated annealing by (Kirkpatrick et al., 1983). These incomplete techniques are often used to tackle larger problems.

Filtering techniques utilize constraints actively to make inferences about the problem (Van Beek & Dechter, 1997). The main goal of filtering techniques is to detect locally or completely inconsistent partial assignments (Van Beek & Dechter, 1997), (Debruyne & Bessiere, 1997). (Rossi et al., 2006) stated that one of the most widely used techniques is the arc consistency, first introduced by (Montanari, 1974) for discrete CSP. (Debruyne & Bessiere, 1997) extended arc consistency to numerical CSP by the mean of Bound-consistency. This method checks the consistency of each value in the domain of a variable with each constraint separately (Bessière, 1994), (Vareilles, 2005) and (Rossi et al., 2006). (Bessiere, 1991) mentioned the arc consistency filters locally, but does not guarantee a problem has a solution.

There are several degrees of filtering, each corresponding to the number of variables involved in the local consistency check. The higher the degree, the more effective the detection of inconsistent combinations, but the longer the detection process takes (Vareilles, 2005). Filtering methods include node consistency (Mackworth, 1977), arc consistency (Mackworth, 1977), path consistency (Mackworth, 1977), and k-consistency (Freuder, 1978). These filtering techniques allow reflecting choices on the current problem by eliminating inconsistent values. The search for solutions becomes interactive, relying on a sequence of coherent choices that lead to one or multiple solutions.

When comparing solving and filtering techniques, filtering techniques improve solving techniques. For example, forward checking, developed by (Haralick & Elliott, 1980), combines arc consistency filtering with a backtracking algorithm.

2.3.4.2. CSP for system configuration

The CSP approach has been used to formalize knowledge and reason on it in configuration problems (Felfernig et al., 2014). The utilization of CSPs in system configuration problems has been so extensive that it has earned a dedicated chapter in the Handbook of Constraint Programming (Junker, 2006).

(Tsang, 1993) stated that configuration problems can be formulated as constraint problems. The application of CSP in system configuration involved defining a system as a fixed and finite set of component variables. Each of these component variables held property variables and port variables to represent connections with other components. The domains of variables were limited to a finite and discrete set. Restrictions on which component combinations were valid were represented using compatibility constraints. The solution of CSP involved assigning a value to each variable and determining the final configuration.

(Xie et al., 2005) employed CSP to model and solve an engineering product configuration problem. The goal was to define a methodology for a generic configurator that can handle complex constraints, in an engineering product configuration problem.

(Aldanondo & Vareilles, 2008) proposed a constraint-based approach in the domain of product configuration and showed how it can be extended to upstream Requirements Configuration and downstream Process Configuration. The CSP modeling framework was used to identify configuration elements and formalize these elements. In their work they formalized descriptive and then physical view of the product after that they associated these two views.

(Tidstam et al., 2016) proposed the modeling and solving of the CSP as an aid during the inspection of configuration rules. Their objective was to develop CSP variations that could automate manual tasks involved in the development of vehicle configuration rules.

(Männistö et al., 2001) defined a novel mechanism based on generic models of product individuals organized into a specialization hierarchy to support multiple abstraction levels. For creating such hierarchies, they defined a set of transformation operations on models.

2.3.4.3. Advantages and drawbacks of using CSP

This section evaluates the constraint satisfaction problems using our eight criteria. For each of them, three modalities are possible: Yes, No and Difficult. We argue our answers in the light of the literature review.

In knowledge formalization, the criteria for CSP are assessed as follow:

- **KFC1: Clearly distinguish between descriptive and structural views: YES**

Configuration problems naturally take into account different views of the family of systems to be configured. Different views, functional, physical or assembly, are considered. In (Aldanondo & Vareilles, 2008) and (L. L. Zhang et al., 2013), a descriptive and a structural view of a family of systems are clearly defined in a single generic configuration model, defined as a constraint satisfaction problem. The variables and constraints in the model are thus artificially grouped according to whether they belong to one view or the other. The two views are linked by a set of constraints that define the solution space.

- **KFC2: Better structure knowledge for later reuse in modeling: YES**

We have only found work by (Shen et al., 2012) and (Guillon, Ayachi, et al., 2021) that uses ontologies and taxonomies in system configuration. In their work, (Shen et al., 2012) proposes a PES (Product Extension Service) configuration model in the case of servicisation. The configuration is supported by a model called PESCO (Product Extension Services Configuration Ontologies), which consists of three sub-ontologies: (1) a service sub-ontology (SO), (2) a product sub-ontology (PO), and (3) a customer sub-ontology (CO). In (Guillon, Ayachi, et al., 2021)'s work, the business knowledge required to develop proposals is structured using a taxonomy in which each concept is associated with a CSP: variables are associated with each concept in the taxonomy, and business rules are formalized in the form of constraints. All elements of the model are thus formalized in the form of a tree structure of concepts associated to CSPs, which are then reused as appropriate to create generic models.

- **KFC3: Explicitly formalize generic models at different levels of abstraction: DIFFICULT**

We have only found the work of (Männistö et al., 2001), which refers to a level of abstraction in system configuration. In their work, (Männistö et al., 2001) proposed a complete generic model, including all levels of abstraction of all items of the model. The generic model therefore corresponds to the complete tree of abstraction and the authors conclude that the generic model can become very difficult to understand and manage.

- **KFC4: Unquestionably validate the consistency of the knowledge: YES**

Filtering and resolution methods can be used to validate the consistency of knowledge models. This makes it possible to (1) verify the completeness of the solution set, by a complete resolution of the CSP, which can be costly in terms of time and space, and (2) partially verify the solution space by more or less strong filtering.

In knowledge reuse, the criteria for CSP are assessed as follow:

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements: DIFFICULT**

In (Aldanondo & Vareilles, 2008) and (L. L. Zhang et al., 2013), the generic model with its different views is used to configure both the descriptive and the functional view of a family of systems. As the two views are intrinsically coupled, it is not possible to obtain a solution that only includes the descriptive part of the system (in the CSP sense). In fact, the assignment is only partial if only the descriptive view has been configured. The rest of the variables have therefore to be assigned to any consistent values.

- **KRC2: Formalize and capitalize ETO requirements during configuration: YES**

(Bonev & Hvam, 2013) and (Sylla, Guillon, Vareilles, et al., 2018) have studied how ETO requirements can be tackle during configuration activity. They have studied the impacts of ETO requirements on configuration activity and have proposed various ETO configuration requirements.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements: DIFFICULT**

In (Männistö et al., 2001), it is proposed to configure systems thanks to the complete generic model including all abstraction levels of all elements of the model. For each subsystem and component, the user has to choose between different variants and options before configuring it. Some branches of the tree are therefore cut during the configuration activity, and this process leads to the minimal BOM that is consistent with the model and the user's requirements.

- **KRC4: Interactively configure a system: YES**

Interactive configuration can be achieved using filtering techniques. Depending on the filtering algorithms used (arc consistency, path consistency, etc.), inconsistent values will be more or

less well removed from the space of possibilities. A compromise must therefore be found between filtering quality and response time.

In conclusion, CSPs seem to be a good candidate to partially answer our research questions, both in terms of knowledge formalization and reuse. Undoubtedly, CSPs can be used to formalize and reason about any kind of configuration knowledge in order to interactively configure systems. Unfortunately, CSPs suffer from their lack of structure and the difficulty of simply reusing parts of generic models.

The following section is devoted to the Case-Based Reasoning (CBR) approach.

2.3.5. CBR and system configuration

In this section, we will start by defining the CBR. Then, we'll delve into how this CBR is applied in the context of systems configuration. Lastly, we will assess this approach using the criteria we have defined.

2.3.5.1. CBR definition

The Case-Based Reasoning (CBR) approach (Kolodner, 1992), (Aamodt & Plaza, 1994) is an approach that is based on the idea that similar problems have similar solutions. This approach allows solving a new problem by finding similar problems solved in the past and reusing the knowledge and information of those problems. In other words, this approach solves new problems by finding similar past cases that have been solved previously and adapting their solutions to the new situation.

The main elements of CBR approaches are cases, and similarity functions. A case or an experience is described as a problem situation (Aamodt & Plaza, 1994). Each case is described by a set of attributes whose definition domain has been previously fixed. The set of attributes of a case allows to describe the problem encountered and the proposed solution. When a case is stored in a database, each of its attributes must be filled in and assigned a unique value. Moreover, (Xu et al., 2009) mentioned that typically, the case consists of two parts: 1) the problem and the description of the attributes of the case, and 2) the solution to the problem. A previous case or source case refers to a past problem situation that has been studied in a way that its related knowledge and information can be reused to solve a new similar problem in the future (Aamodt & Plaza, 1994). A case base is a database of previously solved problems and their solutions (Aamodt & Plaza, 1994).

A distance measure, called similarity, is used to classify the different cases in the case base (source cases) in relation to their similarity to the submitted problem (target case). There are two types of similarity measures: 1) local similarity measure (Bergmann, 2002) is the similarity between the values of the domains of the attributes taken two by two, 2) global similarity measure is the similarity, related to cases, aggregates the local similarities to determine the global similarity between two cases.

A CBR system is made up of five phases: Define, Retrieve, Reuse, Revise, and Retain (Kolodner, 1992), (Aamodt & Plaza, 1994). The Define phase involves describing a new problem to compare it with past problems stored in the case base. The Retrieve phase involves

searching for and selecting the most similar cases in the case base. In the Reuse phase, knowledge related to the retrieved cases is adapted by humans to propose an initial solution. In the Revise phase, the initial solution is tested and compared to the actual solution, and if necessary, revised to make it more suitable. Finally, in the Retain phase, the new problem and its related information are stored in the case base.

There are two types of uses for case-based reasoning: (1) interpretation and (2) problem-solving (Kolodner, 1992), (Marling et al., 2002). Interpretation involves using past cases to evaluate new cases. Problem-solving involves adapting a solution to a past problem, to meet the requirements of the new situation. (Shaharin et al., 2019) mentioned that the main advantage of the CBR, is that it allows to solve problems based on the experiences gained from previous cases that led to providing effective solutions. CBR can be used to support the reuse of knowledge by retrieving and adapting solutions from the case base to new product configurations. Cases help a reasoner to focus reasoning on important parts of a problem by pointing out what features of a problem are the important ones (Kolodner, 1992).

2.3.5.2. CBR for system configuration

(Tseng et al., 2005), applied the CBR approach to perform actual product configuration, aiming to reuse previous successful reasoning cases. (Lee & Lee, 2005) also used the CBR for product configuration. (Yang et al., 2008) employed the CBR approach to solve product configuration problems, but noted that it is useful when knowledge is incomplete and does not support the reuse of product structure knowledge and constraints. (Xu et al., 2009) introduced an extended object model for case-based reasoning in product configuration design. This model adopts various methods of knowledge expression such as constraints, rules, and objects. It supports all processes of case-based reasoning in product configuration design such as case representation, indexing, retrieving, and case revision. A metering pump product configuration design system was developed based on this model to support customized products.

2.3.5.3. Advantages and drawbacks of using CBR

In this section, we assess the CBR using our eight criteria, with three possible modalities for each: Yes, No, and Difficult. We substantiate our responses based on the insights from the literature review.

Regarding knowledge formalization, the assessment criteria for CBR are evaluated as follows:

- **KFC1: Clearly distinguish between descriptive and structural views: YES**

In (Sylla et al., 2021), they propose object-oriented case representation model which allows to evaluate systems performance. In their object-oriented representation, each case is defined as a class which is described by a set of attribute-value pairs. Classes are hierarchically organized and may consist of one or more sub classes.

- **KFC2: Better structure knowledge for later reuse in modeling: YES**

(Stahl & Bergmann, 2000) proposed the idea of recursive CBR and applied it for product configuration. The approach structures products hierarchically into sub-components and recursively applies CBR to find best-matching alternative sub-components, thereby avoiding

huge portions of the knowledge acquisition effort. The presented approach assumes a limited number of dependencies between the different sub-problems in the application domain to be efficient. This approach is particularly suited to the customization of structured products in Electronic Commerce applications.

- **KFC3: Explicitly formalize generic models at different levels of abstraction: YES**

(Bergmann & Wilke, 1996) develop a general framework for representing cases at different levels of abstraction, which is useful for analyzing existing and designing new approaches in case-based reasoning. The purpose of an abstraction hierarchy in case-based reasoning is to organize cases into levels of abstraction, with each level representing a different degree of detail.

- **KFC4: Unquestionably validate the consistency of the knowledge: NO**

In the existing literature, we have not identified any studies that specifically address this particular criterion. To the best of our knowledge, there is no intrinsic mechanism in CBR to validate the consistency of the knowledge.

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements: YES**

(Sylla et al., 2021) proposes a method for retrieving the most similar cases from the case base using CBR approach. At the first step, the structure of the case base is exploited to directly retrieve relevant previous cases to the target case. The case base structure contains general knowledge about which systems are of the same type and which systems are not of the same type. Once a new system must be evaluated, its properties are extracted to identify the relevant class in the case base class hierarchy. Only cases that are instances of the class of the target case and those that are instances of its parent classes are considered. Then, a similarity measure is used to rank the retrieved cases based on their similarity to the target case. The most similar cases are then selected and used to propose an initial solution.

- **KRC2: Formalize and capitalize ETO requirements during configuration: YES**

CBR can be employed to formalize and capitalize on ETO requirements during configuration. By using past cases or solutions, CBR allows for tailoring specific configurations to meet ETO requirements.

(Sylla et al., 2021) The paper proposes a CBR approach for the evaluation of complex systems in ETO industrial situations. The proposed CBR system can be used to evaluate both a complete system and a part of the system. The authors propose a generic approach that allows for the retrieval of the most similar cases in both situations.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements: DIFFICULT**

In their proposed approach (Bergmann & Wilke, 1996), represented cases at multiple levels of detail can be reused. It means their approach enables the retrieval of appropriate cases at the

same levels of detail when searching for similar cases to a new one. However, to our knowledge, if the cases are not formalized and stored in a structured way, reusing them to configure in this step will be difficult.

- **KRC4: Interactively configure a system: NO**

In CBR, the adaptation of case is done based on the user requirements. To the best of our knowledge there is no paper to discuss specifically interactive configuration using CBR, in which users provide input progressively converging to a solution.

In conclusion, CBR enables us to formalize knowledge at different levels of abstraction, in a structured way and at different levels of abstraction. However, it has difficulties when it comes to checking consistency and interactive configuration. Therefore, in this thesis, we will not consider CBR.

The following section is devoted to the hybrid approach.

2.3.6. Hybrid approach and system configuration

In this section, we start by the definition of hybrid approach. Next, we explore how this hybrid approach can be applied to system configuration. Finally, we assess this approach based on our criteria.

2.3.6.1. Hybrid approach definition

(Stumptner, 1997) defined hybrid approach as an approach that employs several representations and reasoning mechanisms.

2.3.6.2. Hybrid approach for system configuration

In the literature, a few papers worked on the hybrid approaches, such as the use of CSP and CBR approaches by (Vareilles et al., 2012), CBR and ontology by (Romero Bejarano et al., 2014).

(Vareilles et al., 2012) proposed an approach to aiding design decisions that combines the strengths of CSP and CBR. CSP is an approach that uses a constraint model and a constraint filtering mechanism to solve design problems, while CBR enables to solve problems by retrieving similar past cases and adapting them to the current problem. The authors propose to use both approaches together to take advantage of their strengths. They suggest that the knowledge base of CSP can be used to represent explicit knowledge in the form of constraints, while the case base of CBR can be used to represent contextual knowledge in the form of past cases. (Romero Bejarano et al., 2014) addressed the fulfillment of requirements for CBR processes in system design. The proposed method defines an integrated CBR process in line with system engineering principles and establishes an ontology to capture knowledge about the design. Based on the ontology, models are provided for requirements and solutions representation, followed by a recursive CBR process suitable for system design. (Guillon, Villeneuve, et al., 2021) proposed to build and deployed a knowledge-based system (KBS) combining the following approaches: ontology, CSP, and CBR in order to capture, formalize,

and reuse knowledge relevant to bids. A case study from a company building electrical parts of harbor lifting devices was illustrated.

2.3.6.3. Advantages and drawbacks of using hybrid approaches

In this section, we assess the hybrid approaches against our eight criteria. For each criterion, there are three potential modalities: Yes, No, and Difficult. Our justifications for these assessments are based on our review of the existing literature.

In knowledge formalization, the criteria for hybrid approaches are assessed as follow:

- **KFC1: Clearly distinguish between descriptive and structural views: YES**

In (Romero Bejarano et al., 2014), they defined an ontology to represent knowledge about system design, including characteristics, domains, and constraints. Moreover, to design a system, they break it into subsystem. In their study (Guillon, Villeneuve, et al., 2021), developed a knowledge-based system (KBS) with the aim of assisting companies in the bid development process. They identified several concepts (associating to components family) and formalized a BOM (at three levels of decomposition). They represented the knowledge using ontology. Moreover, they defined Business rules by means of constraints (using CSP).

- **KFC2: Better structure knowledge for later reuse in modeling: YES**

In (Romero Bejarano et al., 2014), using ontology, a hierarchical structure of concepts is defined representing a taxonomy. Moreover, the paper proposes an integrated approach that combines Recursive CBR (RCBR) with system engineering principles to improve the system design process. The paper explains that the RCBR approach involves decomposing the initial system into a hierarchy of subsystems, and retrieving a solution for each subsystem. If there are unsolved subproblems, the system is recursively carried out to find sub solutions. In (Guillon, Villeneuve, et al., 2021), concepts are structured in a taxonomy a taxonomy is created to structure concepts and group concepts together.

- **KFC3: Explicitly formalize generic models at different levels of abstraction: YES**

In (Romero Bejarano et al., 2014), they create an ontology. At the core of this ontology is the most general concept called "System" which doesn't have any parent concepts. The relationship between concepts in the ontology is established through edges that signify generalization and specialization. This means that any concept inherits the characteristics of its parent concepts.

- **KFC4: Unquestionably validate the consistency of the knowledge: YES**

In their study (Vareilles et al., 2012), the authors integrated CBR with constraint filtering, which narrows down variable domains based on constraints during the design process. By employing constraint filtering, inputs to the CBR system are of higher quality and more efficient, as constraints reduce possible input variations.

- **KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements: YES**

In the approach proposed by (Romero Bejarano et al., 2014), there are two scenarios to consider based on the designer's choices: 1) When the designer decides to decompose the system solution into multiple subsystems, the RCBR process is applied to develop each subsystem solution at a lower level. This entails the designer specifying dedicated subsystem requirements for each of them. Once these subsystem solutions are developed, they are integrated to form the final system solution, with values assigned to the variables. 2) Alternatively, if the solution is not comprised of subsystems, the development process concludes after the adaptation task.

- **KRC2: Formalize and capitalize ETO requirements during configuration: YES**

In their study (Romero Bejarano et al., 2014), three different scenarios are defined for the process of determining a solution:

Scenario 1: This scenario comes into play when there are no suitable existing solutions, and a solution needs to be created from scratch.

Scenario 2: In cases where a suitable solution already exists, it is selected for reuse. Then, a check is performed. If the chosen solution meets all the clear requirement constraints, there is no need for further adjustments, and the solution is ready.

Scenario 3: If the selected solution does not meet the requirement constraints, it is copied and adapted to align with the new requirements.

This adaptation process can range from making minor adjustments to completely restructuring the solution, depending on the complexity of the required changes.

- **KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements: YES**

In the study (Romero Bejarano et al., 2014), they focused on addressing the requirement of integrating the Case-Based Reasoning (CBR) process with modular and hierarchical engineering design processes. Their objective was to promote system reuse in a structured manner, adhering to established system engineering standards, where systems could be systematically reused across various levels.

- **KRC4: Interactively configure a system: YES**

The paper (Vareilles et al., 2012) propose an innovative approach to combine ontology and CBR, emphasizing their collaborative interaction rather than a sequential process. These approaches exchange and share knowledge to reach a solution. In the proposed approach, ontology and CBR are employed in an interactive way, where designers gradually input their requirements, progressively reducing the solution space until complete the design.

In conclusion, existing hybrid approaches in the literature can fulfill some of our criteria. For instance, the association of CSP and CBR enables to check the consistency of model and interactively configure. In contrast, the association of ontology and CBR enable to formalize

knowledge distinguishing descriptive and structural view, better structure knowledge and formalize at different levels of abstraction.

The following section is devoted to the synthesis of this section.

2.3.7. Synthesis

In this section, we have started by identifying the main criteria for formalizing and reusing knowledge for system configuration. Therefore, we have identified 4 criteria for knowledge formalization including KFC1: Clearly distinguish between descriptive and structural views, KFC2: Better structure knowledge for later reuse in modeling, KFC3: Explicitly formalize generic models at different levels of abstraction and KFC4: Unquestionably validate the consistency of the knowledge. Moreover, we have identified 4 criteria for knowledge reuse including KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements, KRC2: Formalize and capitalize ETO requirements during configuration, KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements and KRC4: Interactively configure a system.

Next, we have explained various approaches in the domain of system configuration, such as ontology, UML/SysML, CSP, CBR, and hybrid approaches. Following a comprehensive evaluation of each approach based on our eight criteria, we have drawn conclusions regarding their respective advantages and drawbacks, making a clear distinction between the phases of knowledge formalization and knowledge reuse. The result of this comparison is represented in Table 2. For each criterion, the response options are as follows: ‘Y’ represent Yes, ‘N’ shows No and ‘D’ means Difficult. As shown, using the ontology, it is possible to fulfill KFC1, KFC2, KFC3, KRC1, KRC2, KRC3 however it is difficult to fulfill KFC4 and KRC4. Using the UML and SysML, it is possible to fulfill KFC1, KFC2, KFC3, KRC2, however it is difficult to fulfill KFC4. It is not possible to fulfill KRC1, KRC3 and KRC4. Using CSP, it is possible to fulfill KFC1, KFC2, KFC4, KRC2, and KRC4 however it is difficult to fulfill KFC3, KRC1, KRC3. Using the CBR, it is possible to fulfill KFC1, KFC2, KFC3, KRC1, and KRC2 however it is difficult to fulfill KRC3. It is not possible to fulfill KFC4, and KRC4. Using the association of CSP and CBR, it is possible to fulfill KFC4, and KRC4. Using the association of Ontology and CBR, it is not possible to fulfill KFC4, and KRC4.

In this thesis, to fulfill all of our criteria for both knowledge formalization and knowledge reuse, the association of ontologies and CSPs approaches are selected. Ontology enable us to create generic models at higher level of abstraction or at lower level of abstraction. CSPs enable us to formalize different relations linking artifacts and their characteristics. Moreover, it allows to check the consistency of formalized knowledge and perform interactive configuration.

Table 2. Different approaches for system configuration

Requirements	Ontology	UML/ SysML	CSP	CBR	CSP + CBR	Ontolody + CBR	Ontolody + CSP + CBR
KFC1: Clearly distinguish between descriptive and structural views	Y	Y	Y	Y	N	Y	Y
KFC2: Better structure knowledge for later reuse in modeling	Y	Y	Y	Y	N	Y	Y
KFC3: Explicitly formalize generic models at different levels of abstraction	Y	Y	D	Y	N	Y	N
KFC4: Unquestionably validate the consistency of the knowledge	D	D	Y	N	Y	N	N
KRC1: Configure descriptive and/or structural views for the same model according to the configuration requirements	Y	N	D	Y	N	Y	N
KRC2: Formalize and capitalize ETO requirements during configuration	Y	Y	Y	Y	N	Y	N
KRC3: Configure according to several levels of abstraction for the same knowledge model according to the configuration requirements:	Y	N	D	D	N	Y	N
KRC4: Interactively configure a system	D	N	Y	N	Y	N	N

The following section is dedicated to the synthesis of the chapter.

2.4. Synthesis

This thesis is related to the configuration of technical and physical systems. Therefore, in section 2.1, we established a literature review on system configuration and configuration activity and we defined generic models. We defined interactive configuration and different types of customer requirements. We also explained which types of requirements are fulfilled in CTO and ETO situations.

In section 2.2, first, we briefly presented knowledge management process. Then, we mainly focused on knowledge formalization and the topics which are important to our first research

question. Therefore, we provided a literature review on descriptive and structural views. We explained why we need to distinguish these two views and each one is composed of which elements. Additionally, we delved into the concept of commonality of models, abstraction, generalization and specialization principles as well as inheritance, which allows us to create generic models at different levels of abstraction.

In section 2.3, we have started by identifying the main criteria for formalizing and reusing knowledge for system configuration. Then, we have presented and compared different approaches including ontology, UML/SysML, CSP, CBR, and hybrid approaches that can be used to formalize and reuse knowledge for system configuration. After assessing each approach against our eight criteria we have concluded with their advantages and disadvantages, while separating between knowledge formalization and knowledge reuse. Ultimately, we have chosen the association of ontologies and CSPs as the most suitable option, since it fulfils our criteria.

Based on the literature review, we found out there is lack of papers on the formalization of generic models at various abstraction level while separating two different views of descriptive and structural. To address this scientific gap, we must answer our first research question in chapter 3 "Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?". Therefore, in chapter 3, we will define descriptive view and structural view for a family of artifacts, then create an ontology of generic models after that explain the update of generic models. We will employ the associations of ontologies, CSPs, commonality and inheritance principles.

On the other hand, regarding the literature review, we identified lack of study on the reusing formalized generic models either descriptive or structural, and at different abstraction levels to fulfill ETO requirements during configuration. To address this scientific gap, we must answer our second research question in chapter 4 "How can ETO requirements be processed during configuration activity?". Thus, in chapter 4, we will define instances of descriptive view and structural view for a family of artifacts, then define three generic process for CTO knowledge reuse, ETO knowledge reuse and CTO-ETO knowledge reuse. We will adapt CTO configuration activity towards ETO configuration activity to meet ETO requirements.

Part of our literature review has been published in:

- Maryam Mohammad Amini, Michel Aldanondo, Élise Vareilles, Thierry Coudert. Twenty Years of Configuration Knowledge Modeling Research. Main Works, What To Do Next?. IEEM 2021 - International Conference on Industrial Engineering and Engineering Management, Dec 2021, Singapore. pp.1328-1332.

3. Knowledge formalization for system configuration

This chapter is devoted to our first research question: "Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?" In chapter 2, we have highlighted the absence of work on knowledge formalization for system configuration to create generic models at various levels of abstraction, separating descriptive and structural views.

In response to this gap, we propose to combine commonality, ontologies, CSP approaches, and inheritance principles to define the concept of Generic Artifact, notated $GA^{(i)}$ and its two views: the Generic Artifact Descriptive Model, notated $GADM^{(i)}$, and the Generic Artifact Structural Model, notated $GASM^{(i)}_j$. $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$ are the core of our first contribution and lead to the proposition of three taxonomies to better manage knowledge in system configuration. In this chapter, we will explore how commonality, ontologies, CSP approaches, and inheritance principles can help to define $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$. In section 3.1, we start with a formal definition of each of them, then in section 3.2 we discuss how they can be generalized, specialized and structured within an ontology. In section 3.3, we explain how they can be updated. After that, in section 3.4, we provide a discussion about the benefits of our proposal. Finally, in section 3.5 we synthesize the chapter.

We adopt the knowledge management process explained in chapter 2, for system configuration knowledge formalization. As illustrated in Figure 9, the process involves extracting knowledge (step 1), formalizing it into generic models representing both descriptive and structural views

at different levels of abstraction (step 2), validating the generic models by checking their consistency (step 3), and finally storing the ontologies of consistent generic models in a Generic Model Base (GMB) (step 4). The process is iterative, allowing experts to go back and forth between these four steps as needed (represented by arrows). In the following section, the formalization of knowledge for system configuration and checking its consistency is investigated. As explained in section 2.2, the first step about knowledge extraction is not considered in the thesis.

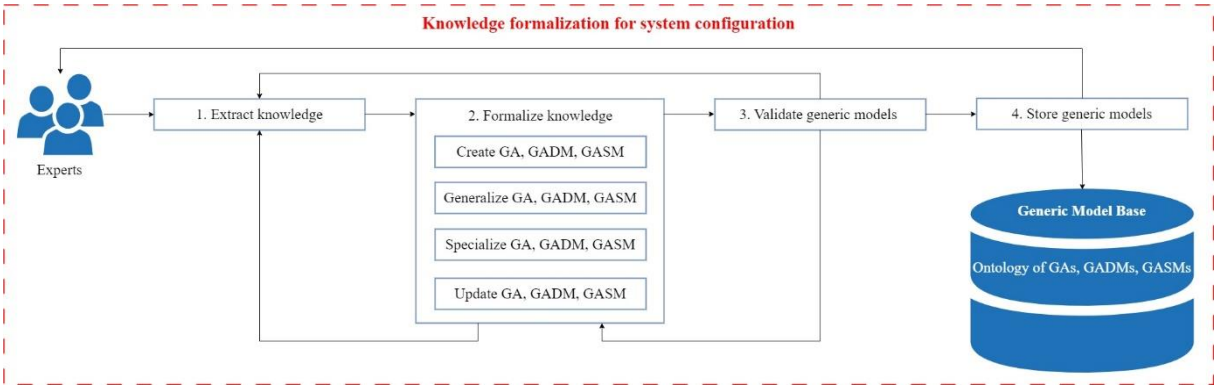

Figure 9. Knowledge formalization for system configuration

3.1. GA, GADM, and GASM definition

This section is devoted to the definition of GA, GADM, and GASM. First, in section 3.1.1, the definition of Generic Artifact, notated GA, is proposed, followed by the definition of its descriptive view, formalized by Generic Artifact Descriptive Model, notated as GADM in section 3.1.2. Then in section 3.1.3, the definition of its structural view which is formalized as Generic Artifact Structural Model notated as GASM is provided. Finally, the synthesis of this section is given in section 3.1.4.

3.1.1. Generic Artifact definition and example

As mentioned in chapter 2, we consider technical and tangible systems, subsystems, or components as Artifacts. Systems can be composed of a set of subsystems and components. Subsystems can be composed of a set of subsystems or components; a component cannot be composed. Subsystems and components correspond to tangible parts of systems to be configured (Guillon, Ayachi, et al., 2021).

Definition 3: Generic Artifact or GA
A Generic Artifact or a GA is a semantically unambiguous conceptual view of a family of artifacts. A GA is characterized by a prototypical name and a description of the family of artifacts it represents.

Following definition 3, a GA is defined by a name and a description. As shown in Figure 10, a generic artifact named A is denoted A.GA. For example, as represented in Figure 11, a GA representing the family of bikes is defined as follows:

– **Bike.GA:** the name **Bike** evokes a clear mental image of a bike.

- **Description:** A bike is a vehicle with two wheels that you ride by sitting on it and pushing two pedals with your feet (Collins, 1982).

A GA does not explicitly represent any knowledge about features of a relevant family of artifacts nor its structure. However, it provides a clear understanding of the family of artifacts it represents. Once the GA has been created, it is stored in a taxonomy of GA.

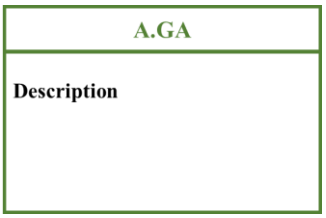

Figure 10. A.GA

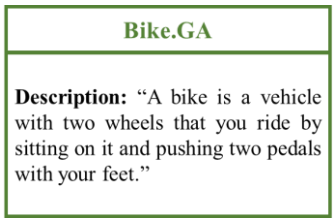

Figure 11. Bike.GA example

The next section is dedicated to the definition of the descriptive view of a GA.

3.1.2. Generic Artifact Descriptive Model definition

As explained in chapter 2 and in response to KFC1, a GA needs to be described by its key features or key attributes, and KPIs, without showing any information about its internal functioning. It is therefore considered as a black box where a set of configured attributes lead to a particular assessed solution. We, therefore, introduce this descriptive view as a GADM and define it as follows.

Definition 4: Generic Artifact Descriptive Model or GADM
The descriptive view of a GA, is named Generic Artifact Descriptive Model and notated GADM. The GADM consists of the key attributes of the family of artifacts with their domains, KPIs with their domains, and allowed or forbidden relations between attributes and/or KPIs values to describe the possible solutions of the artifact family without considering its structure. A GADM is associated to one and only one GA, and vice versa.

Following the definition, a GADM describes its associated GA thanks to:

- **Name:** we suggest that the GADM be given the same name as the GA to better establish the link between GA and GADM.
- **Description:** it is a statement that describes how the GADM currently looks. The description of a GADM can be similar to the description of its GA. It can contain an explanation of the functions or roles of the GA.
- **Attributes with their domains:** list of key descriptive attributes and their valid domain that describe the current GA,
- **KPIs with their domains:** list of key performance indicators and their valid domains that are needed to assess the current GA,
- **Relations between attributes and/or KPIs values:** list of relations between attributes and/or KPIs values that describe the solution space of the current GA. These relations allow

or forbid certain combinations of attributes and/or KPIs values, thus they define the possible characteristics of the GA with the associated performances.

To formalize and check the knowledge consistency of a GADM, we propose to map it within a CSP denoted *GADM(CSP)*:

- Attributes and domains are mapped to variables and domains of the CSP,
- KPIs and domains are also mapped to variables and their corresponding domains,
- Relations between attributes and/or KPIs of GADM are represented in terms of compatibility tables and numerical functions.

This formalization of GADM as a CSP allows us to use filtering techniques to ensure the local consistency of the GADM knowledge, as expressed in KFC4.

To create a GADM, first of all, the corresponding GA is required. Then, the same name as GA is given, and then description, attributes with their domains, KPIs with their domains, and relations between attributes and/or KPIs values must be defined. Once the GADM has been described, it is translated into a CSP. In this way, the consistency of the GDAM knowledge is checked. This process is iterative, meaning that the experts can modify the GADM knowledge at any time (adding attributes, modifying definition domains, relationships, etc.), then re-translate it into the CSP to check its accuracy. The consistency of the GADM can therefore be checked at each new iteration to validate the formalized knowledge. Once the GADM has been created and validated, it is stored in a taxonomy of GADM.

As illustrated in Figure 12, a GADM is the descriptive view of a GA, which is mapped into a CSP (notated *GADM(CSP)*). For a GA, one and only one GADM can be defined.

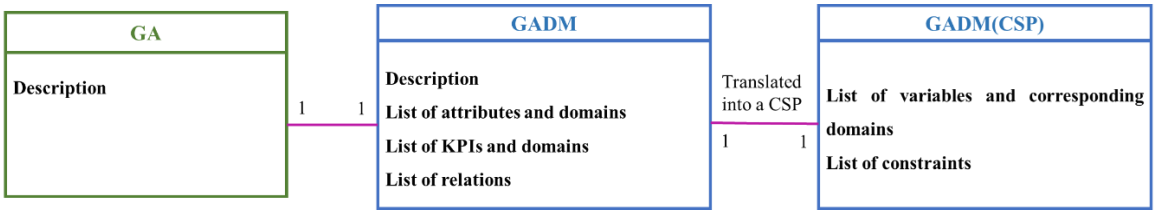

Figure 12. GADM and its translation into a CSP

For instance, a GADM of a bike family is represented in Figure 13 along with its CSP. In *Bike.GADM*, three attributes of Color, User and RingBell are defined. In *Bike.GADM*, two relations are defined. One relation represents that different colors are suitable for certain users and also cost and weight depends on them. The other relation represents that the quantity of ring bells depends on the user.

Three attributes and two KPIs in *Bike.GADM* corresponds to five variables in the CSP of *Bike.GADM*. Moreover, the relations in *Bike.GADM* are translated into compatibility tables linking the compatible values of variables. For example, the first relation in *Bike.GADM* is translated into a compatibility table that links the compatible values of four variables of User, Color, Weight, and Cost. The second relation is translated into a table of compatibility linking the compatible values of two variables of User and RingBellQty, containing two tuples.

Constraint filtering is then applied and the domains of variables are restricted. For instance, the initial domain of Weight is {[0, 35]} but after filtering the constraints, it becomes {[2, 35]}.

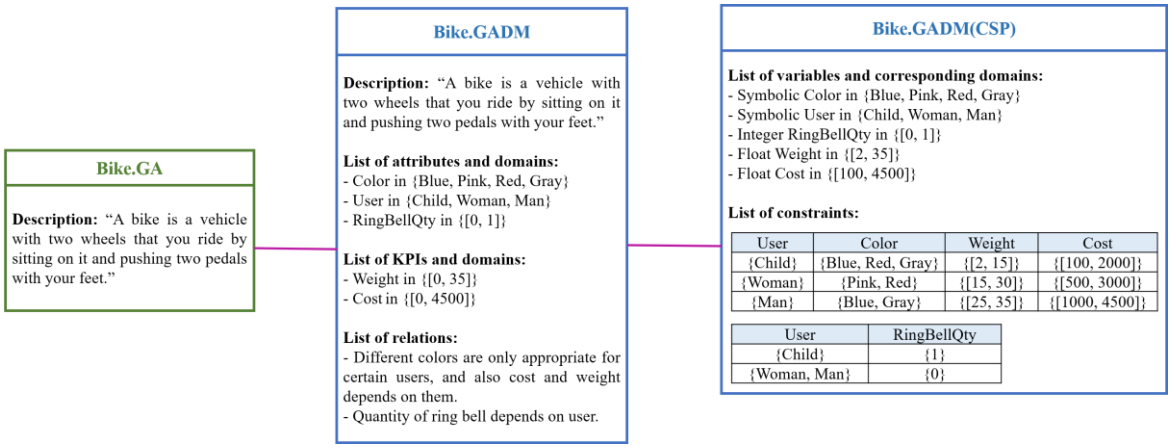

Figure 13. An example of a *Bike.GADM* and its translation into a CSP

The next section is devoted to the definition of the structural view of a GA.

3.1.3. Generic Artifact Structural Model definition

As explained in chapter 2 and in response to KFC1, a GA can also be seen by its internal structure and its internal functioning explanations. In the particular case of system configuration, it corresponds to the Bill of Material (BOM) of an artifact. We, therefore, introduce this white box of a GA as a complementary view of a GADM, named Generic Artifact Structural Model (GASM). We define it as follows.

Definition 5: Generic Artifact Structural Model or $GASM_j$
The structural view of a GA, is named Generic Artifact Structural Model and notated GASM. A GASM is always associated with a GA and describes its structure as a single-level BOM by a quantified list of its existing GAs and their relations (including KPIs assessment). A GA can be associated to several GASM presenting different versions of the BOM. We therefore identify each GASM in order to differentiate them in the knowledge model, by an index j. The notation of GASM is then $GASM_j$.

The GASM definition process is iterative, recursive and bottom-up. The process is bottom-up because the creation of a GASM requires the prior creation of a GA and its GADM. If a GA constituting a GASM also has a GASM, the latter is indirectly associated with the GASM being defined. A GA may have several associated GASMs with different details or versions on the single level nomenclature, noted as $GASM_j$.

Following the definition, a $GASM_j$ consists of:

– **Name:** The name of the $GASM_j$ must clearly indicate its content. As several $GASM_j$ may be associated with the same GA, this is important for better structuring of knowledge to differentiate them. We suggest naming a $GASM_j$ by concatenating the name of its associated GA and additional information describing its structure.

- **Description:** it is a statement describing how the $GASM_j$ currently looks, or what the current $GASM_j$ consists of.
- **$GASM_j$ composition (quantity, GA):** list of pairs of quantity and GAs from which the $GASM_j$ is composed at the first level of composition.
- **KPIs aggregation methods:** list of methods to evaluate the KPIs of the current corresponding GASM according to the GADM associated to the GA and their quantities.
- **Relations between GAs or between attributes and/or KPIs of GAs:** list of relations within the $GASM_j$ that describe all the solution space of the current $GASM_j$. The relations within $GASM_j$ can have different types including (Arana, 2007), (Blecker & Friedrich, 2006): Require relations between GAs, Exclude relations between GAs, and Compatibility relations between GAs, or between attributes and/or KPIs of different GAs.

As for GADM, to check the knowledge consistency (KFC4), we map a $GASM_j$ to a CSP, denoted $GASM_j(CSP)$, as follows:

- In $GASM_j(CSP)$, each GA is substituted by its $GADM(CSP)$, and the quantities correspond to variables and domains.
- KPIs aggregation methods are formalized as constraints linking all the relevant KPIs of the $GADM(CSP)$ composing the current $GASM_j(CSP)$.
- Relations are expressed in terms of compatibility tables and numerical functions on all relevant and available variables, coming either from the current $GASM_j(CSP)$ and all the variables of the $GADM(CSP)$.

To create a $GASM_j$, you first need the corresponding GA. A relevant and discriminating name and description must then be defined. The experts can then concentrate on its nomenclature by identifying the elements that make it up: the list of quantity / GA pairs is thus established. The relationships between the different GAs are also defined. Once the $GASM_j$ has been described, it is translated into a CSP or $GASM_j(CSP)$. The consistency of $GASM_j(CSP)$ is then checked using filtering algorithms. This process is iterative, meaning that the experts can modify the $GASM_j$ knowledge at any time (adding attributes, modifying definition domains, relationships, etc.), then re-translate it into the CSP to check its accuracy. The consistency of the $GASM_j$ can therefore be checked at each new iteration to validate the formalized knowledge. Once the $GASM_j$ has been created and validated, it is stored in a taxonomy of GASM.

As shown in Figure 14, a $GASM_j$ is the structural view of a GA, which is mapped into a CSP. For a GA, from zero to many $GASM_j$ can be defined. Since a GA can have several structures corresponding to several ways to represent its composition (i.e. different versions of the BOM), it can have many $GASM_j$. However, a GA can have no $GASM_j$ if the GA is related to a family of artifacts that is not composed (i.e. a component). Every $GASM_j$ is translated into a CSP (notated $GASM_j(CSP)$).

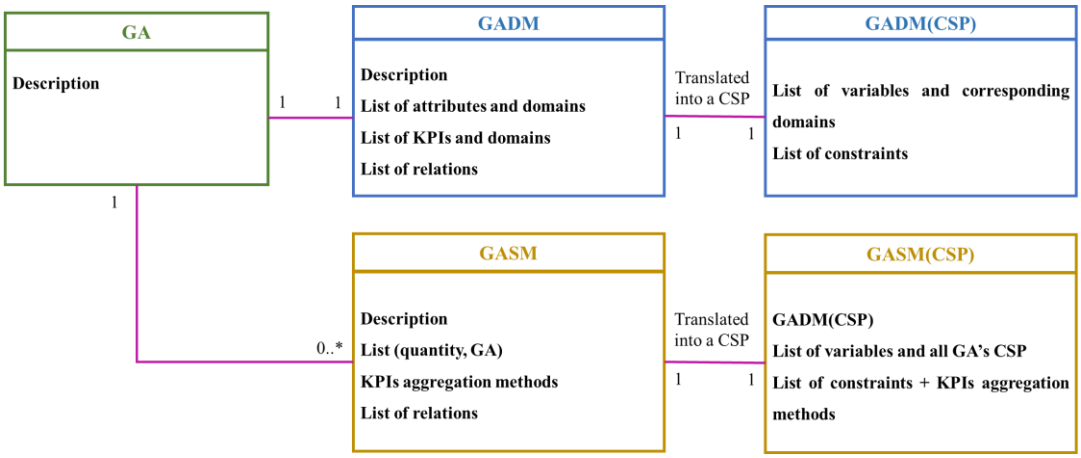

Figure 14. UML model of GA, GADM and GASM with its translation into a CSP

In the example of Figure 15, *Bike.GASM*$_j$, the jth GASM of a bike family (i.e. *Bike.GA*), composed of two or three *Wheel.GA* and one *Seat.GA*, is illustrated. Two KPIs aggregation methods related to Weight and Cost are defined. They represent respectively that the weight is the sum of the weights of its GAs and the cost is the sum of the costs of its GAs. This relation indicates that if the bike's user is a child, then the material of the seat must be plastic.

As represented, the CSP of *Bike.GASM*$_j$ contains the CSP of *Bike.GADM*, in addition, the GAs in *Bike.GASM*$_j$ are substituted by their CSP in the CSP of *Bike.GASM*$_j$. The quantities of GAs in *Bike.GASM*$_j$ corresponds to variables in its CSP. KPIs aggregation methods are translated into constraints (i.e. numerical functions). Moreover, a relation in *Bike.GASM*$_j$ is translated into a compatibility table in *Bike.GASM*$_j$*(CSP)* that links the compatible values of the variable User of *Bike.GADM* and the variable Material of *Seat.GADM*. This compatibility table consists of two tuples. Then, constraint filtering is applied to obtain a consistent piece of knowledge for the future configuration of bikes.

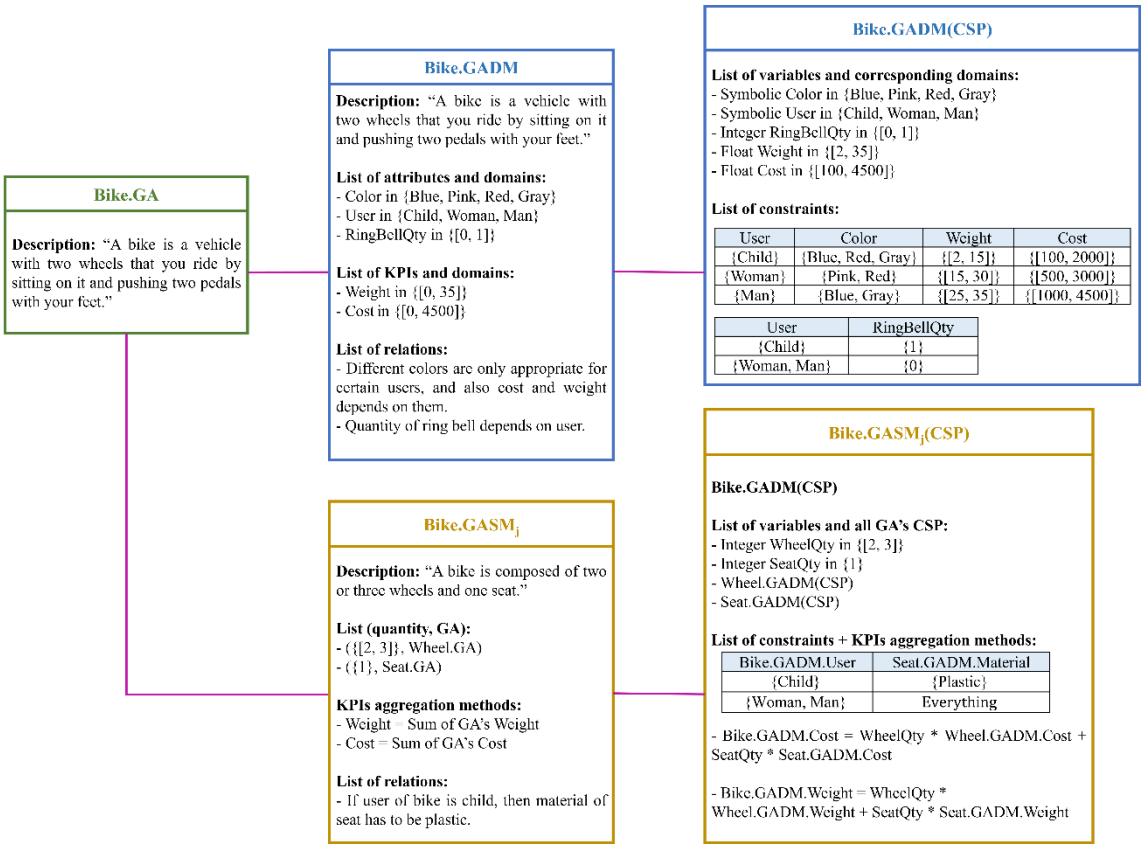

Figure 15. Example of a *Bike.GASM$_j$* and its translation into a CSP

In the following, the synthesis of this section is presented.

3.1.4. Synthesis

To address our initial research question, it is necessary to establish consistent generic models presenting descriptive and structural views of the same artifact. In this section, the concept of Generic Artifact, as well as its descriptive view or GADM and structural view or *GASM$_j$* has been introduced and defined.

After their definition, we have explained how these models are linked together by a family of artifacts and how they can be translated as a CSP. A simple but realistic bike example illustrates our proposals.

This proposal answers directly to the need of presenting two complementary views of an artifact (descriptive and structural) in the knowledge model (KFC1) and ensures the consistency of knowledge during its formalization (KFC4).

In the following section, we explain how we can use these views to formalize knowledge at different levels of abstraction (KFC3) and create an ontology (KFC2) that gathers GA, GADM, and GASM, by the use of generalization and specialization relations as well as commonality and inheritance principles.

3.2. Ontology of GA, GADM, and GASM

As explained in chapter 2, ontologies rely on generalization and specialization relations and composition of classes and allow several levels of abstraction. Based on that, we propose to use the concept of commonality, also defined in chapter 2, in addition to these properties of ontologies to create an ontology of families of artifacts and therefore better manage knowledge in system configuration (answers to KFC2 and KFC3).

This section is dedicated to the definition of the respective ontology of GA, GADM, and GASM. Therefore, in section 3.2.1, the taxonomy of Generic Artifact is proposed. Then, in section 3.2.2, the taxonomy of the Generic Artifact Descriptive Model is defined. After that, in section 3.2.3, the taxonomy of the Generic Artifact Structural Model is proposed. Finally, in section 3.2.4, the synthesis of the section is represented. All the proposals are illustrated on examples.

3.2.1. Taxonomy of Generic Artifacts

Whatever GA is considered, it can be generalized or specialized into another GA, respectively at a higher or lower level of abstraction. We note $GA^{(i)}$ a GA at level of abstraction i, $GA^{(i-1)}$ its generalization and $GA^{(i+1)}$ its specialization.

Definition 6: Generic Artifact Generalization or $GA^{(i-1)}$
GA generalization is the process of creating a more generalized GA (notated $GA^{(i-1)}$) based on the commonality of several $GA^{(i)}$ and the factorization of their common knowledge at a higher level of abstraction.

Based on commonality principle, this process involves two steps:

Step 1) Identify the commonality of at least two existing $GA^{(i)}$ (mainly based on the analysis of their descriptions),

Step 2) Define a relevant name and description of the $GA^{(i-1)}$.

For example, as illustrated in Figure 16, *MountainWheel.*$GA^{(3)}$ and *CityWheel.*$GA^{(3)}$ are generalized into *Wheel.*$GA^{(2)}$ because of their commonality. In the description of *MountainWheel.*$GA^{(3)}$, it is stated that *"A mountain wheel is a circular component that is intended to rotate on an axle bearing for mountain environment"*, while in the description of *CityWheel.*$GA^{(3)}$, it is mentioned that *"A city wheel is a circular component that is intended to rotate on an axle bearing for urban environment"*. To generalize *MountainWheel.*$GA^{(3)}$ and *CityWheel.*$GA^{(3)}$ into *Wheel.*$GA^{(2)}$, the description of *Wheel.*$GA^{(2)}$ must be written in a way that it is applicable in both environments. The *Wheel.*$GA^{(2)}$ is then created and is described as follows: *"A wheel is a circular component that is intended to rotate on an axle bearing"*.

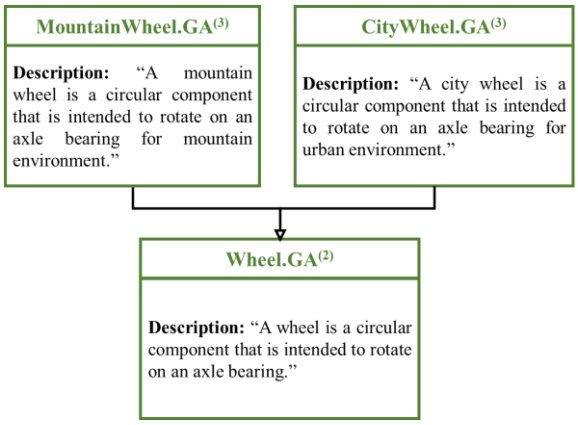

Figure 16. An example of *Wheel.GA*$^{(2)}$ generalization

Definition 7: Generic Artifact Specialization or *GA*$^{(i+1)}$
GA specialization is the process of creating a more specific GA (notated *GA*$^{(i+1)}$) based on an existing GA (*GA*$^{(i)}$) and its modification (restriction and enrichment) with specific knowledge at a lower level of abstraction (i+1).

Based on inheritance principle, this process involves two steps:

Step 1) Duplicate the knowledge of *GA*$^{(i)}$ into the knowledge of *GA*$^{(i+1)}$,

Step 2) Modify the inherited name and description of *GA*$^{(i+1)}$.

Figure 17 represents a partial taxonomy with three GA (*A.GA*$^{(i)}$ and *B*$_1$*.GA*$^{(i+1)}$ and *B*$_2$*.GA*$^{(i+1)}$). *A.GA*$^{(i)}$ is specialized into *B*$_1$*.GA*$^{(i+1)}$ and *B*$_2$*.GA*$^{(i+1)}$.

For example, as represented in Figure 18, *Bike.GA*$^{(2)}$ is specialized into a *CityBike.GA*$^{(3)}$ and *MountainBike.GA*$^{(3)}$. To specialize *Bike.GA* into *CityBike.GA*$^{(3)}$, its description must be modified and it must be mentioned that the city bikes are designed for urban areas. On the other hand, to specialize *Bike.GA*$^{(2)}$ into *MountainBike.GA*$^{(3)}$, its description needs to be modified and it must be emphasized that mountain bikes are suitable for mountain areas.

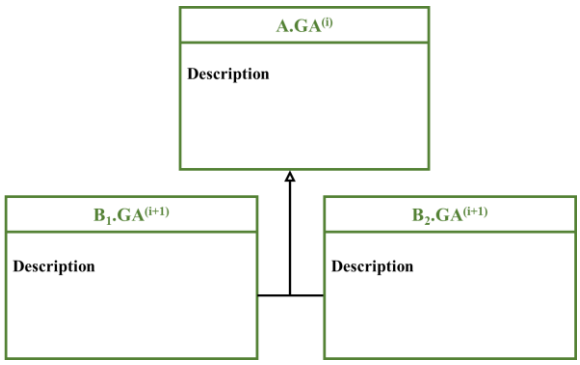

Figure 17. Specialization of *A.GA*$^{(i)}$

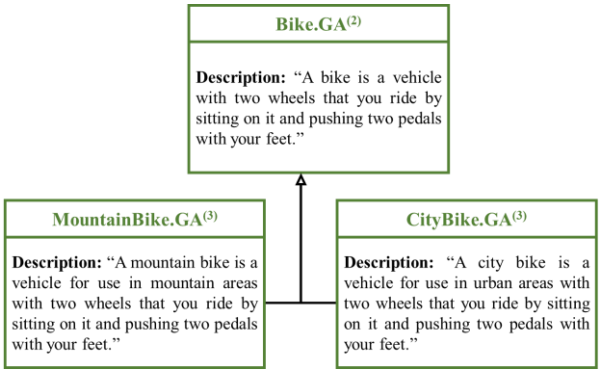

Figure 18. An example of *Bike.GA*$^{(2)}$ specialization

Therefore, the taxonomy of GAs is a partial ontology in this thesis. All the GAs are structured in a taxonomy that represents the "is a" relationship between GAs (Sciore, 1989). This taxonomy classifies GAs hierarchically, according to their generalization or specialization of

knowledge. For instance, Figure 19 illustrates two taxonomies of GAs. First, *System.GA*$^{(1)}$ is defined as the most general GA because we consider artifacts as tangible and technical systems. In other words, *Bike.GA*$^{(2)}$, *CityWheel.GA*$^{(2)}$, *MountainWheel.GA*$^{(2)}$, *Seat.GA*$^{(2)}$, *Light.GA*$^{(2)}$ and *Brake.GA*$^{(2)}$ are generalized into *System.GA*$^{(1)}$. Conversely, *System.GA*$^{(1)}$ is specialized into several children (i.e. *Bike.GA*$^{(2)}$, *CityWheel.GA*$^{(2)}$, *MountainWheel.GA*$^{(2)}$, *Seat.GA*$^{(2)}$, *Light.GA*$^{(2)}$, and *Brake.GA*$^{(2)}$). Then, as explained in Figure 16 and Figure 18, because of their commonality, *CityWheel.GA*$^{(3)}$, and *MountainWheel.GA*$^{(3)}$ are generalized into *Wheel.GA*$^{(2)}$. As *Bike.GA*$^{(2)}$ was not considered as sufficiently specific, it is specialized into *CityBike.GA*$^{(3)}$, and *MountainBike.GA*$^{(3)}$. Every descendant inherits the knowledge of his ancestors. Therefore, after the generalization and specialization *CityBike.GA*$^{(3)}$, *MountainBike.GA*$^{(3)}$ and also *Wheel.GA*$^{(2)}$ are added to the taxonomy of GAs.

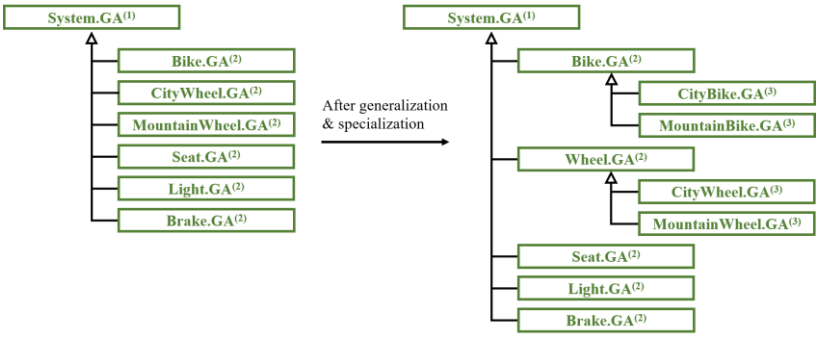

Figure 19. GAs taxonomy with specialization and generalization

The generalization or specialization of a *GA*$^{(i)}$ implies that its GADM (notated *GADM*$^{(i)}$) is also respectively generalized or specialized. The following section is dedicated to the taxonomy of GADM and the processes of generalization and specialization.

3.2.2. Taxonomy of Generic Artifact Descriptive Models

When a *GA*$^{(i)}$ is generalized or specialized, its associated *GADM*$^{(i)}$ has also to be generalized or specialized, defining different levels of abstraction for the descriptive views.

Definition 8: Generic Artifact Descriptive Model Generalization or *GADM*$^{(i-1)}$
GADM generalization is the process of creating a more general GADM (notated *GADM*$^{(i-1)}$) based on the commonality of several *GADM*$^{(i)}$ and the factorization of their common knowledge at a higher level of abstraction.

The process of GADM generalization can be described as follows:

Step 1) Identify the commonality of at least two existing *GADM*$^{(i)}$: since in GADM generalization, *GADM*$^{(i-1)}$ allows to gather the commonality of *GADM*$^{(i)}$, it is essential to first identify this commonality. The commonality of a set of *GADM*$^{(i)}$ is defined as the common knowledge among them including attributes with their domains, KPIs with their domains, and relations. Therefore, the expert identifies: 1) the attributes, and KPIs that are shared across all the *GADM*$^{(i)}$ as well as their domains, along with 2) the common relations between their values. The commonality is defined by:

- Common attributes and KPIs with their domains are identified: $GADM^{(i-1)}$ gathers the common attributes and KPIs among all $GADM^{(i)}$. For each common attribute or KPI, the union of their domains in all the $GADM^{(i)}$ has to be achieved in order to encompass all the specific domains.
- Common relations between attributes and/or KPIs values are identified: $GADM^{(i-1)}$ gathers the common relations among all $GADM^{(i)}$.

Step 2) Factorize the commonality into the knowledge of $GADM^{(i-1)}$: the identified commonality (i.e. common attributes and KPIs, union of their domains, and common relations) are factorized into the $GADM^{(i-1)}$.

In order to guarantee the consistency of knowledge of GADM ontology, we strongly recommend that factorized knowledge cannot be modified in any of the $GADM^{(i)}$ that has been considered. In each GADM, we therefore need to tag elements (attributes, domains, and relations) that are inherited or generalized as 'I' to distinguish them from specific knowledge tagged by 'S'. We strongly recommend that inherited elements can only be used to refine the common knowledge into specific knowledge.

Step 3) Define a relevant name and the description of the $GADM^{(i-1)}$: to generalize at least two $GADM^{(i)}$ into a $GADM^{(i-1)}$, after factorizing the commonality of the $GADM^{(i)}$ into the knowledge of $GADM^{(i-1)}$, a common name and description for the $GADM^{(i-1)}$ need to be defined to reflect its more generalized nature.

Since the GADMs are mapped to a CSP, the generalization follows exactly the same principle, notated $GADM^{(i-1)}(CSP)$:

- Attributes, KPI and their domains are mapped to variables and domains of the CSP, prefixed by I or S to know if attributes have been inherited or not. Domains on the generalized $GADM^{(i-1)}(CSP)$ are unified to cover all the knowledge,
- Relations between attributes and/or KPIs of GADM are represented in terms of compatibility tables and numerical functions, marked as I or S, as well as their tuples.

In Figure 20, $MountainWheel.GADM^{(2)}$ and $CityWheel.GADM^{(2)}$, that are two independent GADM as illustrated on the left part of Figure 19, are represented with their CSP.

$MountainWheel.GADM^{(2)}$ consists of two specific attributes of Diameter and Material, and two KPIs of Weight and Cost (inherited from $System.GADM^{(1)}$) as well as a relation indicating that any diameter of mountain wheels cannot be associated with any material and the cost and weight of mountain wheels depends on them. They are tagged by 'S' except Weight and Cost that are tagged by 'I' (inherited from $System.GADM^{(1)}$). $MountainWheel.GADM^{(2)}$ is then translated into a CSP. In $MountainWheel.GADM^{(2)}(CSP)$, attributes, and KPIs are represented by four variables. The relation is translated into a compatibility table linking compatible values of four variables of S: Diameter, S: Material, I: Weight, and I: Cost. It consists of three tuples.

$CityWheel.GADM^{(2)}$ consists of three attributes of Diameter, Material, and InnerTubeQty, two inherited KPIs of Weight and Cost as well as two relations. One represents that any diameter of the city wheel cannot be associated with any material and the cost and weight of the city

wheel depend on them. The other relation indicates that the quantity of the inner tube depends on the diameter of the city wheel. They are tagged by 'S' except Weight and Cost which are tagged by 'I'. *CityWheel.GADM*$^{(2)}$ is then translated into a CSP. In *CityWheel.GADM*$^{(2)}$*(CSP)*, attributes, and KPIs corresponds to five variables. The first relation is translated into a compatibility table linking compatible values of the variables S: Diameter, S: Material, I: Weight, and I: Cost. It contains three tuples. The second relation is also translated into a compatibility table that links compatible values of the variables S: Diameter and S: InnerTubeQty. It contains two tuples.

An instance of GADM generalization for families of wheels is shown in Figure 21. *MountainWheel.GADM*$^{(3)}$ and *CityWheel.GADM*$^{(3)}$ are generalized into *Wheel.GADM*$^{(2)}$. Then, it is translated into a CSP. In *Wheel.GADM*$^{(2)}$, two attributes of Diameter and Material that are common among attributes of *MountainWheel.GADM*$^{(3)}$ and *CityWheel.GADM*$^{(3)}$ are factorized and they are tagged by 'S'. Moreover, two KPIs of Weight and Cost that are common are factorized in *Wheel.GADM*$^{(2)}$. They are tagged by 'I' because they are inherited from *System.GADM*$^{(1)}$. In *Wheel.GADM*$^{(2)}$, the union of the domains of attributes and KPIs are considered. For example, in *Wheel.GADM*$^{(2)}$, the domain of attribute Diameter is {[16, 26]} which is the union of {[16, 26]} and {[17, 24]} which are respectively the domain of Diameter in *MountainWheel.GADM*$^{(3)}$ and in *CityWheel.GADM*$^{(3)}$. In *Wheel.GADM*$^{(3)}$, one relation that is common among the relations of *MountainWheel.GADM*$^{(3)}$ and *CityWheel.GADM*$^{(3)}$ is factorized. It represents that any diameter cannot be associated with any material, and also cost and weight depends on them. It is tagged by 'S' (because it is specific to *Wheel.GADM*$^{(2)}$).

Initially, in *MountainWheel.GADM*$^{(3)}$ and *CityWheel.GADM*$^{(3)}$, attributes and relations were tagged by 'S'. However, after creating *Wheel.GADM*$^{(2)}$, the tags of two attributes (Diameter and Material) and the tag of the first relation that are inherited from *Wheel.GADM*$^{(2)}$ has been changed from 'S' to 'I'.

In *Wheel.GADM*$^{(2)}$*(CSP)*, four variables of Diameter, Material, Cost and Weight that are common among variables of *MountainWheel.GADM*$^{(3)}$*(CSP)* and *CityWheel.GADM*$^{(3)}$*(CSP)* are factorized. For the variables, the union of their domains is considered in *Wheel.GADM*$^{(2)}$*(CSP)*. For instance, in *Wheel.GADM*$^{(2)}$*(CSP)*, the filtered domain of the attribute Diameter is {16, 18, 20, 22, 24, 26} which is the union of {16, 18, 20, 22, 24, 26} and {18, 20, 22, 24} that are respectively the filtered domain of Diameter in *MountainWheel.GADM*$^{(3)}$ and in *CityWheel.GADM*$^{(3)}$. Among the constraints (compatibility tables), those that link the same variables are taken into account, and among the tuples of these constraints, only the common tuples are taken into account. Therefore, the constraint linking the variables Diameter, Material, Weight and Cost is common. The first two tuples are common, but not the third one. If there are two compatibility tables linking different variables they remain specific to each GADM. Constraint filtering is then applied and the domains of variables are restricted. It is important to notice that the tags of variables (i.e. Diameter and Material) and a constraint inherited from *Wheel.GADM*$^{(2)}$ is changed from 'S' to 'I' due to generalization process. Therefore, in *MountainWheel.GADM*$^{(3)}$*(CSP)* and *CityWheel.GADM*$^{(3)}$*(CSP)*, the tags of the two first tuples are changed to 'I' while the third one remains 'S'.

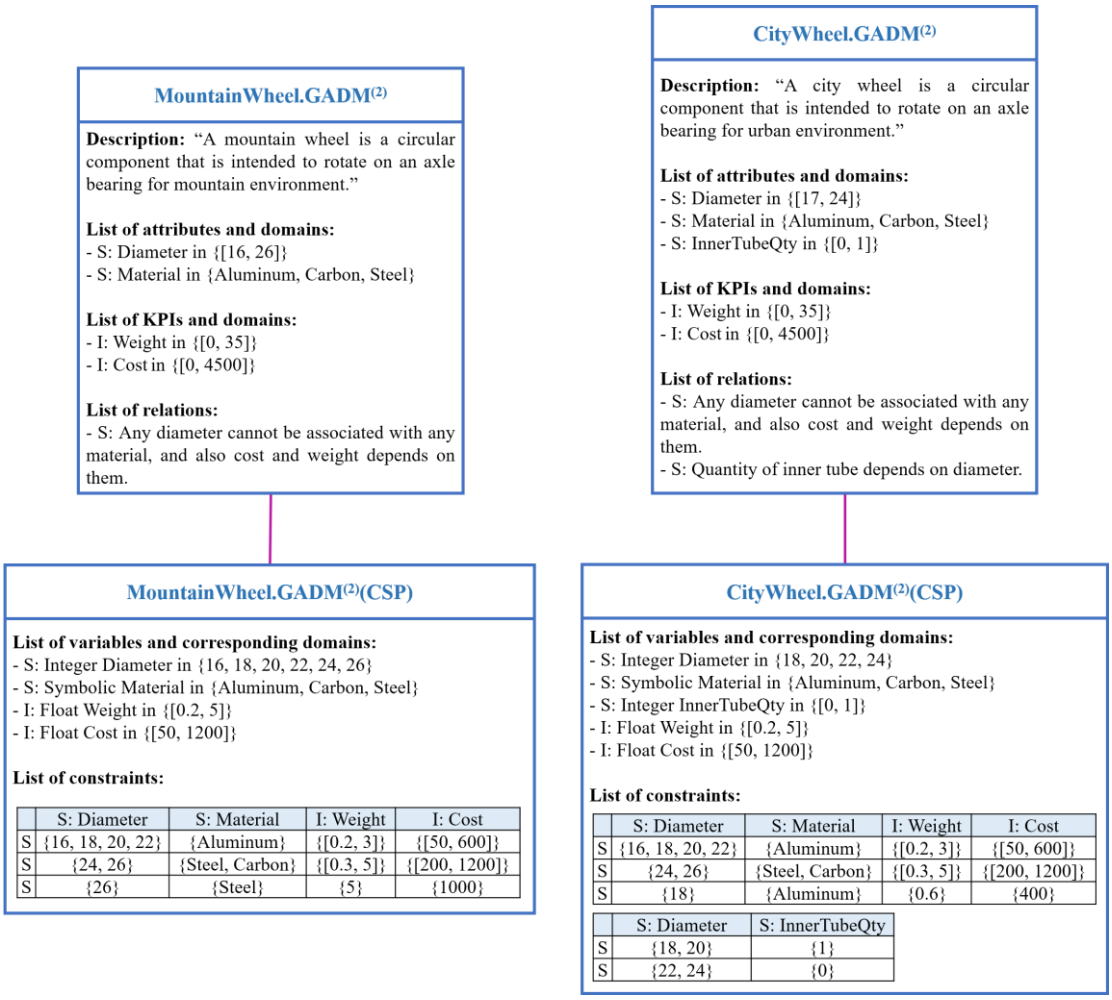

Figure 20. An example of *MountainWheel.GADM*$^{(2)}$ and *CityWheel.GADM*$^{(2)}$ with their CSP

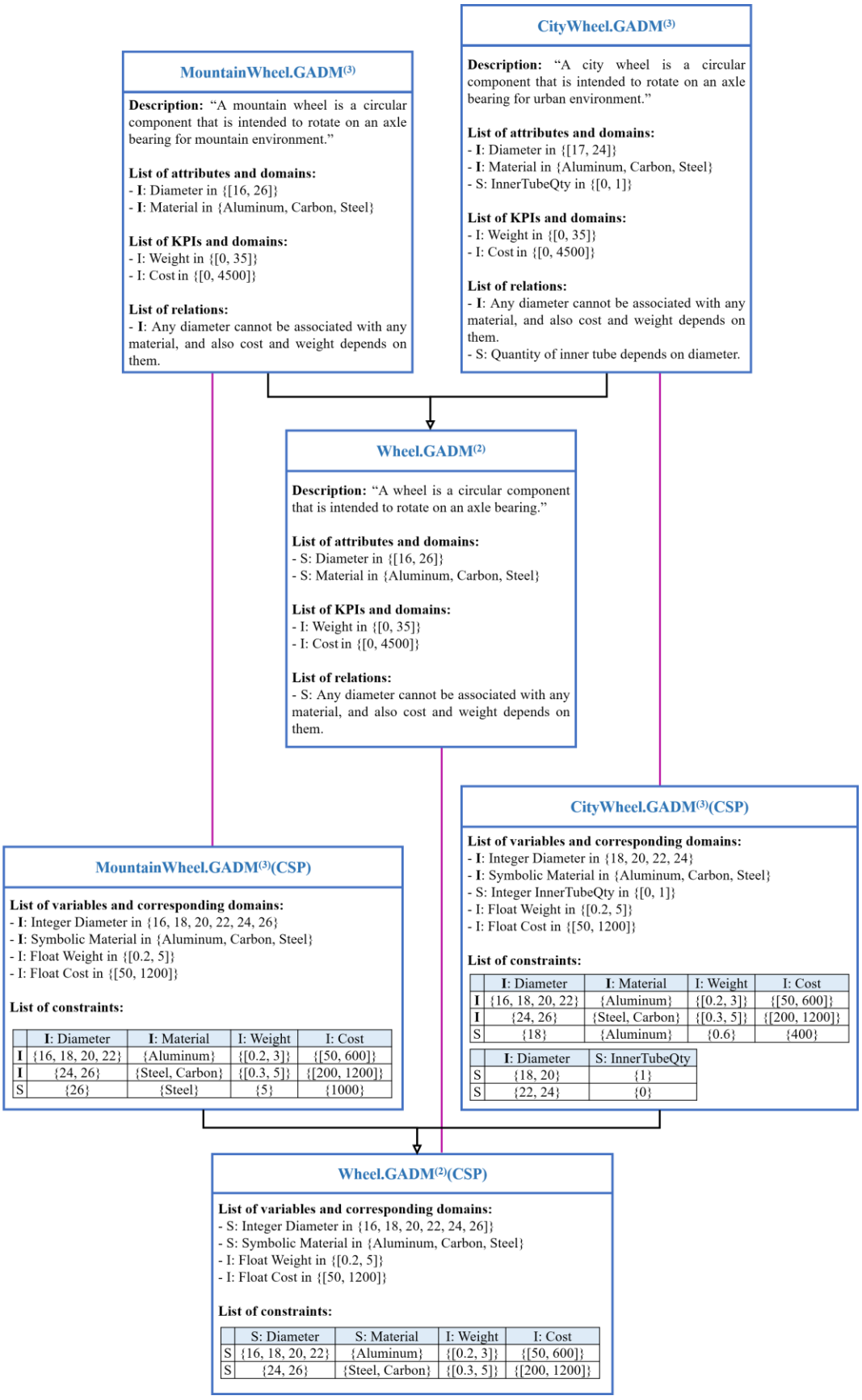

Figure 21. An example of GADM generalization with their CSP

In the following section, we give the definition of GADM specialization.

Definition 9: Generic Artifact Descriptive Model Specialization or $GADM^{(i+1)}$
GADM specialization is the process of creating a more specific GADM (notated $GADM^{(i+1)}$) based on an existing $GADM^{(i)}$ and its modification (restriction and enrichment) with specific knowledge at a lower level of abstraction.

A GADM ($GADM^{(i)}$) can be specialized into another specialized GADM ($GADM^{(i+1)}$) if $GA^{(i)}$ has been previously specialized into $GA^{(i+1)}$. The $GADM^{(i)}$ specialization is a four-step process:

Step 1) Duplicate the knowledge of $GADM^{(i)}$ into the knowledge of $GADM^{(i+1)}$: each $GADM^{(i+1)}$ inherits all the knowledge of its $GADM^{(i)}$ including attributes with their domains, KPIs with their domains, and relations between attributes and/or KPIs values. This first step of specialization provides the opportunity for experts in terms of modeling to reuse an existing GADM (here $GADM^{(i)}$) and create new ones ($GADM^{(i+1)}$) by reusing previously formalized knowledge and specializing it. In each $GADM^{(i+1)}$, all the knowledge (attributes, domains, and relations) is tagged by 'I' as inherited.

Step 2) Modify the inherited name and description of $GADM^{(i+1)}$: to specialize a $GADM^{(i)}$ into a $GADM^{(i+1)}$, after duplicating the knowledge of the $GADM^{(i)}$ into the $GADM^{(i+1)}$, we need to modify the inherited name and description from $GADM^{(i)}$. In addition, we modify the description of $GADM^{(i+1)}$ by adding more details compared to the description of $GADM^{(i)}$ to describe why the $GADM^{(i+1)}$ is more specialized than the $GADM^{(i)}$.

Step 3) Narrow the inherited knowledge of $GADM^{(i+1)}$: In order to guarantee the consistency of knowledge of GADM ontology, we strongly recommend that inherited knowledge can only be restricted. An expert can narrow (or specialize) the inherited knowledge of $GADM^{(i+1)}$ to restrict the solution space and to make the inherited knowledge more precise and accurate by taking two actions: 1) restricting the inherited domains of attributes and/or KPIs, 2) restricting the inherited relations between attributes and/or KPIs values.

- Restrict the inherited domains of attributes and/or KPIs: the inherited domains of attributes and KPIs can be restricted by the expert based on his knowledge about not allowed values. These values are then removed from the domains of attributes and/or KPIs in the $GADM^{(i+1)}$. It is important to notice that the expert is allowed to restrict the inherited domains of attributes and/or KPIs, but not allowed to delete the inherited attributes or KPIs because they are common characteristics that belong to $GADM^{(i)}$.

- Restrict the inherited relations between attributes and/or KPIs values: the inherited relations between attributes and/or KPIs values can also be restricted. The inherited relations can be restricted but they cannot be removed since they are part of the $GADM^{(i)}$ knowledge (common knowledge).

Step 4) Enrich the knowledge of $GADM^{(i+1)}$: after narrowing or restricting the inherited knowledge of $GADM^{(i+1)}$, in this step, the expert can enrich the inherited knowledge. By enriching the knowledge, we mean adding knowledge that is specific to the family of artifacts. There are two ways to enrich the knowledge of $GADM^{(i+1)}$: 1) defining new attributes with their domains, 2) defining new relations between attributes and/or KPIs values. In each $GADM^{(i+1)}$, all the specific knowledge (attributes, domains, and relations) is tagged by 'S' as specific to this particular $GADM^{(i+1)}$.

- Define new attributes (with their domains): the expert can add new attributes with their domains only dedicated to $GADM^{(i+1)}$, in addition to what it inherits from $GADM^{(i)}$. This allows the expert to create a $GADM^{(i+1)}$ with specific characteristics (i.e. attributes with their domains).
- Define new relations between attributes and/or KPIs values: the expert can add new relations only dedicated to $GADM^{(i+1)}$. These relations can be defined to link 1) values of inherited attributes or KPIs and new attributes, and 2) values of new attributes. They are dedicated to the $GADM^{(i+1)}$, in addition to all the relations that a $GADM^{(i+1)}$ inherits from its $GADM^{(i)}$.

Since the GADMs are mapped to a CSP, the specialization follows exactly the same principle, notated $GADM^{(i+1)}(CSP)$:

- Attributes, KPI and their domains are mapped to variables and domains of the CSP, prefixed by 'I' or 'S' to know if attributes have been inherited or not.
- Relations between attributes and/or KPIs of GADM are represented in terms of compatibility tables and numerical functions, marked as 'I' or 'S', as well as their tuples.

As represented in Figure 22, similar to the specialization of $A.GA^{(i)}$, the specialization of $A.GADM^{(i)}$ is created. $A.GADM^{(i)}$ is specialized into $B.GADM^{(i+1)}$. Then, $B.GADM^{(i+1)}$ is translated into a CSP. It means that $A.GADM^{(i)}(CSP)$ is specialized into $B.GADM^{(i+1)}(CSP)$.

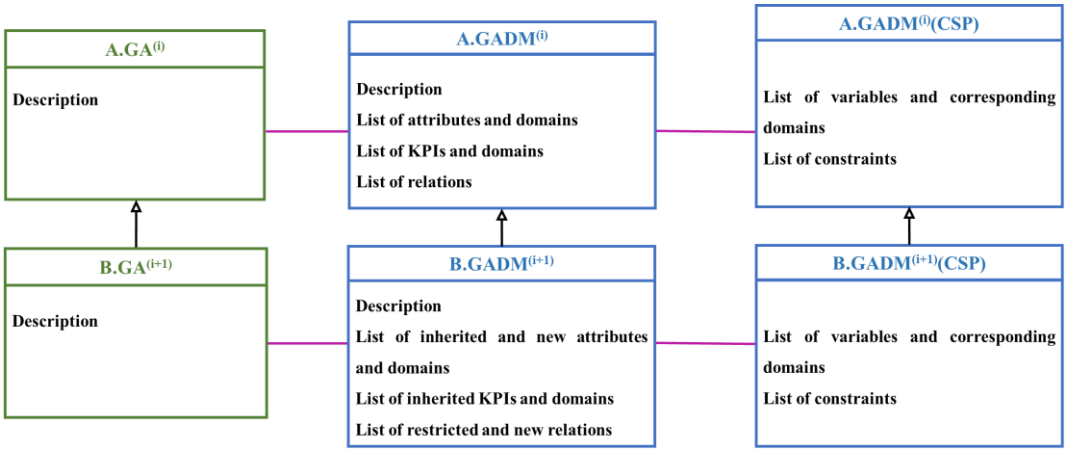

Figure 22. Specialization of $A.GADM^{(i)}$

An instance of GADM specialization for families of bikes is shown in Figure 23. As illustrated on the right part of Figure 19, $Bike.GADM^{(2)}$ is specialized into $CityBike.GADM^{(3)}$. Then, it is translated into a CSP ($Bike.GADM^{(2)}(CSP)$). In $CityBike.GADM^{(3)}$, attributes, KPIs and

relations that are inherited from *Bike.GADM*$^{(2)}$, are tagged by 'I' while the attributes and relations that are only dedicated to *CityBike.GADM*$^{(3)}$ are tagged by 'S'.

In *CityBike.GADM*$^{(3)}$, the domain of an attribute is restricted (the Pink color is removed in the domain of the attribute Color of *CityBike.GADM*$^{(3)}$), a new attribute with its domain is added (S: LightQty), a relation is restricted and a new relation is defined. Then, in the CSP of the *CityBike.GADM*$^{(3)}$, the forbidden value of the variable Color (Pink) is deleted from its domain, a new variable with its corresponding domain is added (S: LightQty), and existing constraints are restricted by adding tuples. The new tuple that is tagged by 'S' is added to indicate that the user 'Man' is only compatible with the color 'Blue', the weight '30' and the cost '4000'. It means that a man can only have a blue city bike. A new compatibility table is added. It indicates the relations between the user and the quantity of lights. A 'Child' bike user only requires one light but an adult bike user ('Man' or 'Woman') needs two lights. Note that the deleted value is removed from the constraints as well. Constraint filtering is applied and the domains of variables are restricted. All the new elements that has been added or the modified elements are represented in bold in Figure 23.

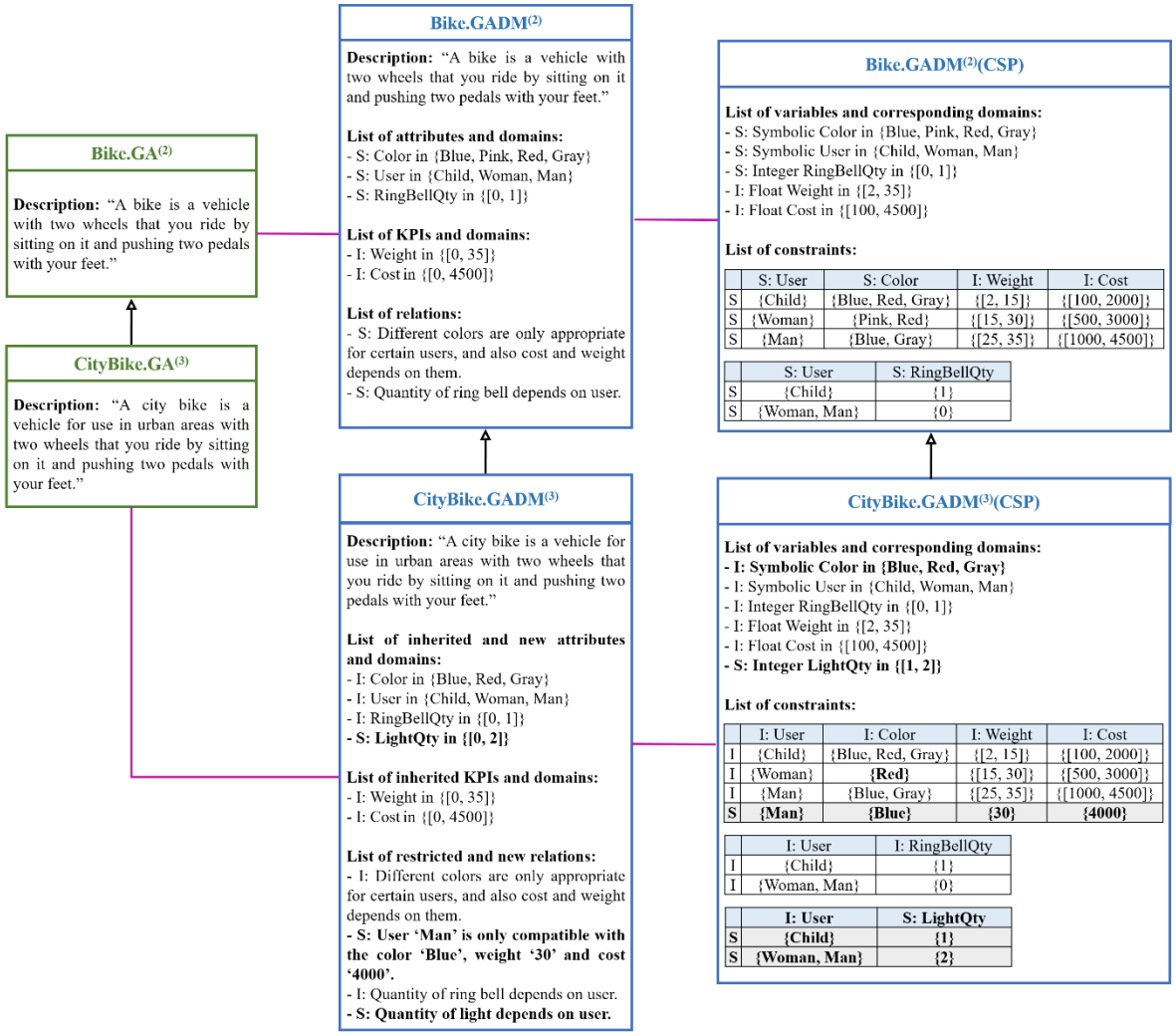

Figure 23. An example of *Bike.GADM*$^{(2)}$ specialization

A GADMs taxonomy is derived from the taxonomy of GAs (illustrated in Figure 19) to classify GADMs based on their common knowledge (i.e. characteristics). As an example, Figure 24 represents a taxonomy of GADMs which is compliant with the related taxonomy of GAs. The most general GADM, *System.GADM*$^{(1)}$, is defined by characteristics common to all GADMs (mainly KPIs). It is the parent of several specialized GADMs, such as *Bike.GADM*$^{(2)}$, *CityWheel.GADM*$^{(2)}$, *MountainWheel.GADM*$^{(2)}$, *Seat.GADM*$^{(2)}$, *Light.GADM*$^{(2)}$ and *Brake.GADM*$^{(2)}$. Corollary, *Bike.GADM*$^{(2)}$, *CityWheel.GADM*$^{(2)}$, *MountainWheel.GADM*$^{(2)}$, *Seat.GADM*$^{(2)}$, *Light.GADM*$^{(2)}$ and *Brake.GADM*$^{(2)}$ are generalized into *System.GADM*$^{(1)}$.

Additionally, *CityWheel.GADM*$^{(3)}$ and *MountainWheel.GADM*$^{(3)}$ are generalized into *Wheel.GADM*$^{(2)}$. *Bike.GADM*$^{(2)}$ is specialized into *CityBike.GADM*$^{(3)}$ and *MountainBike.GADM*$^{(3)}$. After generalization and specialization, the taxonomy of GADMs is represented in Figure 24. During GADM specialization, all specialized GADMs inherit the characteristics of their ancestors. Therefore, *Bike.GADM*$^{(2)}$, *Wheel.GADM*$^{(2)}$, *Seat.GADM*$^{(2)}$, *Light.GADM*$^{(2)}$ and *Brake.GADM*$^{(2)}$ inherit all the characteristics of *System.GADM*$^{(1)}$. *CityBike.GADM*$^{(3)}$ and *MountainBike.GADM*$^{(3)}$ inherits all the characteristics of *Bike.GADM*$^{(2)}$, which includes the characteristics of *System.GADM*$^{(1)}$ as well. Moreover, *CityWheel.GADM*$^{(3)}$ and *MountainWheel.GADM*$^{(3)}$ inherits all the characteristics of *Wheel.GADM*$^{(2)}$.

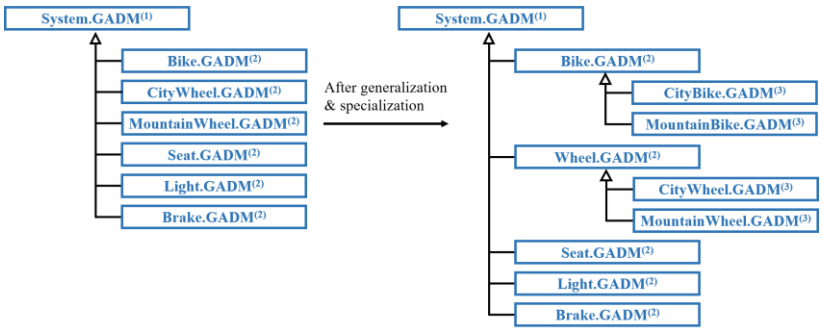

Figure 24. GADMs taxonomies

The generalization or specialization of a *GA*$^{(i)}$ implies that its *GASM*$^{(i)}$ is also respectively generalized or specialized. The following section is dedicated to the taxonomy of GASM.

3.2.3. Taxonomy of Generic Artifact Structural Models

When a *GA*$^{(i)}$ has been generalized or specialized, *GASM*$^{(i)}_j$ its also have to be generalized or specialized leading to different levels of abstraction for the structural views.

Definition 10: Generic Artifact Structural Model Generalization or *GASM*$^{(i-1)}_j$
GASM generalization is the process of creating a more general GASM, notated *GASM*$^{(i-1)}_j$, based on the commonality of several and *GASM*$^{(i)}_j$ the factorization of their common knowledge at a higher level of abstraction.

The process of $GASM^{(i)}_j$ generalization involves the following steps:

Step 1) Identify the commonality of at least two existing: since $GASM^{(i)}_j$ in GASM generalization, a $GASM^{(i-1)}_j$ is defined which gathers the commonality of all specialized , first $GASM^{(i)}_j$ of all, the commonality among the different $GASM^{(i)}_j$ must be identified. Commonality of several $GASM^{(i)}_j$ is defined as the quantity of common knowledge among them. Thus, the expert identifies 1) the GAs that are shared across all the $GASM^{(i)}_j$ and their quantities, 2) the common KPIs aggregation methods as well as 3) the common relations between GAs or between attributes and/or KPIs values of GAs.

- Common $GA^{(i)}$ with their quantities are identified: $GASM^{(i-1)}_j$ gathers the common GAs among all $GASM^{(i)}_j$. Therefore, any $GA^{(i)}$ that are specific to only one or a few of the $GASM^{(i)}_j$ are not considered. Moreover, among $GA^{(i)}$ from the same family but with different levels of abstraction, the most general one is considered. For each GA the union of their quantities in all the $GASM^{(i)}_j$ are considered.
- Common KPIs aggregation methods are identified: $GASM^{(i-1)}_j$ gathers the common KPIs aggregation methods among all $GASM^{(i)}_j$ in which only common $GA^{(i)}$ and the union of their quantities are considered.
- Common relations between $GA^{(i)}$ or between attributes and/or KPIs values of $GA^{(i)}$ are identified: $GASM^{(i-1)}_j$ gathers the common relations among all $GASM^{(i)}_j$. Thus, any relations that are specific to only one or a few of the are not considered $GASM^{(i)}_j$.

Step 2) Factorize the commonality into the knowledge of $GASM^{(i-1)}_j$: the identified commonality (i.e. common GAs, the union of their quantities, common KPIs aggregation methods, and common relations) are factorized into $GASM^{(i-1)}_j$. In order to guarantee the consistency of knowledge of GASM ontology, we strongly recommend that factorized knowledge cannot be modified in any of the that $GASM^{(i-1)}$ has been considered. In each GASM, we therefore need to tag elements (attributes, domains and relations) that are inherited or generalized as 'I' to distinguish them from specific knowledge tagged by 'S'. As for GADM, we strongly recommend that inherited elements can only be used to refine the common knowledge into specific knowledge

Step 3) Define a description of the $GASM^{(i-1)}_j$: to generalize (at least) two in $GASM^{(i)}_j$ to a $GASM^{(i-1)}_j$, after factorizing the commonality of the $GASM^{(i)}_j$ into the $GASM^{(i-1)}_j$, a common name and description for the $GASM^{(i-1)}_j$ must be defined which represents the more generalized nature of the $GASM^{(i-1)}_j$.

Since the GASMs are mapped to a CSP, the GASM generalization follows exactly the same principle, notated $GASM^{(i-1)}_j(CSP)$:

- Attributes, KPI and their domains are mapped to variables and domains of the CSP, prefixed by 'I' or 'S' to know if attributes have been inherited or not. Domains on the generalized $GASM^{(i-1)}_j(CSP)$ are unified to cover all the knowledge,

– Relations between attributes and/or KPIs of GADM are represented in terms of compatibility tables and numerical functions, marked as ‘I’ or ‘S’, as well as their tuples.

MountainWheel.GASM$^{(2)}_{j}$ and *CityWheel.GASM*$^{(2)}_{j}$ are two independent GASMs with their CSPs are illustrated in Figure 25. *MountainWheel.GASM*$^{(2)}_{j}$ consists of one *Rim.GA*$^{(2)}$, one *Tire.GA*$^{(2)}$, KPIs aggregation methods related to Weight and Cost (which have been inherited from the *System.GASM*$^{(1)}_{1}$), as well as a relation representing that the diameter of mountain wheel, rim and tire must be equal due to tire mounting. They are tagged by 'S' since they represent the knowledge that is specific to *MountainWheel.GASM*$^{(2)}_{j}$.

MountainWheel.GASM$^{(2)}_{j}$ is then translated into a CSP. In *MountainWheel.GASM*$^{(2)}_{j}$*(CSP)*, the CSP of *MountainWheel.GADM*$^{(3)}$ is embedded. GAs in *MountainWheel.GASM*$^{(2)}_{j}$ are substituted by their CSP in the CSP of *MountainWheel.GASM*$^{(2)}_{j}$. The quantities of GAs in *MountainWheel.GASM*$^{(2)}_{j}$ corresponds to variables in its CSP. KPIs aggregation methods are translated into two constraints (numerical functions). Moreover, a relation in *MountainWheel.GASM*$^{(2)}_{j}$ is translated into two numerical functions in *MountainWheel.GASM*$^{(2)}_{j}$*(CSP*) indicating that the Diameters of *MountainWheel.GADM*$^{(2)}$, *Rim.GADM*$^{(2)}$, and *Tire.GADM*$^{(2)}$ are equal. In *MountainWheel.GASM*$^{(2)}_{j}$*(CSP)*, KPI are tagged ‘I’ as they have been inherited and the rest is tagged by ‘S’ as they are specific.

As represented in Figure 25. *CityWheel.GASM*$^{(2)}_{j}$ consists of one *Rim.GA*$^{(2)}$, one *Tire.GA*$^{(2)}$, one *InnerTube.GA*$^{(2)}$, KPIs aggregation methods related to Weight and Cost (which have been inherited from the *System.GASM*$^{(1)}_{1}$), a relation representing that the diameter of the city wheel, rim and tire must be equal due to tire mounting as well as another relation representing that the diameter of the city wheel, and inner tube must be equal. KPI are tagged ‘I’ as they have been inherited and the rest is tagged by ‘S’ as they are specific.

An example of generalization is represented in Figure 26. *MountainWheel.GASM*$^{(3)}_{j}$ and *CityWheel.GASM*$^{(3)}_{j}$ are generalized into *Wheel.GASM*$^{(2)}_{j}$. Since *MountainWheel.GASM*$^{(3)}_{j}$ and *CityWheel.GASM*$^{(3)}_{j}$ are both composed of one *Rim.GA*$^{(2)}$, and one *Tire.GA*$^{(2)}$, these common GAs, and the union of their quantities are factorized into *Wheel.GASM*$^{(2)}_{j}$. For KPIs aggregation methods, only the same formulas can be generalized, otherwise they remain specific to the related GASM. Therefore, since the same formula is used in *MountainWheel.GASM*$^{(3)}_{j}$ and *CityWheel.GASM*$^{(3)}_{j}$ for Weight and Cost, they can be generalized. In *Wheel.GASM*$^{(2)}_{j}$, one relation that is common among the relations of *MountainWheel.GASM*$^{(3)}_{j}$ and *CityWheel.GASM*$^{(3)}_{j}$ is factorized. It represents the fact that, due to tire mounting, the diameter of wheel, rim and tire must be equal. In *Wheel.GASM*$^{(2)}_{j}$, everything, but KPI, is tagged by ‘S’.

Initially, in *MountainWheel.GASM*$^{(2)}_{j}$ and *CityWheel.GASM*$^{(2)}_{j}$, GAs, KPIs aggregation methods, and relations are tagged by ‘S’ as they are only dedicated to their specific GASM. However, after generalization, i.e. *Wheel.GASM*$^{(2)}_{j}$ is created, the knowledge inherited from *Wheel.GASM*$^{(2)}_{j}$ in *MountainWheel.GASM*$^{(3)}_{j}$ and *CityWheel.GASM*$^{(3)}_{j}$ will be tagged by ‘I’. It should be noted that for the KPIs aggregation methods if both GAs and their quantities are the same then they are tagged by ‘I’, otherwise they are tagged by ‘S’. For example, in *MountainWheel.GASM*$^{(3)}_{j}$, since the GAs and their quantities are similar to *Wheel.GASM*$^{(2)}_{j}$, KPIs aggregation methods are tagged by ‘I’. However, in *CityWheel.GASM*$^{(3)}_{j}$, in addition to

Rim.GA$^{(2)}$ and *Tire.GA*$^{(2)}$, there is *InnerTube.GA*$^{(2)}$ which is not common (it is only specific to *CityWheel.GASM*$^{(3)}_j$).

Wheel.GASM$^{(2)}_j$, *MountainWheel.GASM*$^{(3)}_j$ and *CityWheel.GASM*$^{(3)}_j$ are mapped into their respective CSP: *Wheel.GASM*$^{(2)}_j$*(CSP)*, *MountainWheel.GASM*$^{(3)}_j$*(CSP)* and *CityWheel.GASM*$^{(3)}_j$*(CSP)*. All their inherited variables are tagged 'I' and all their specific knowledge is tagged 'S'.

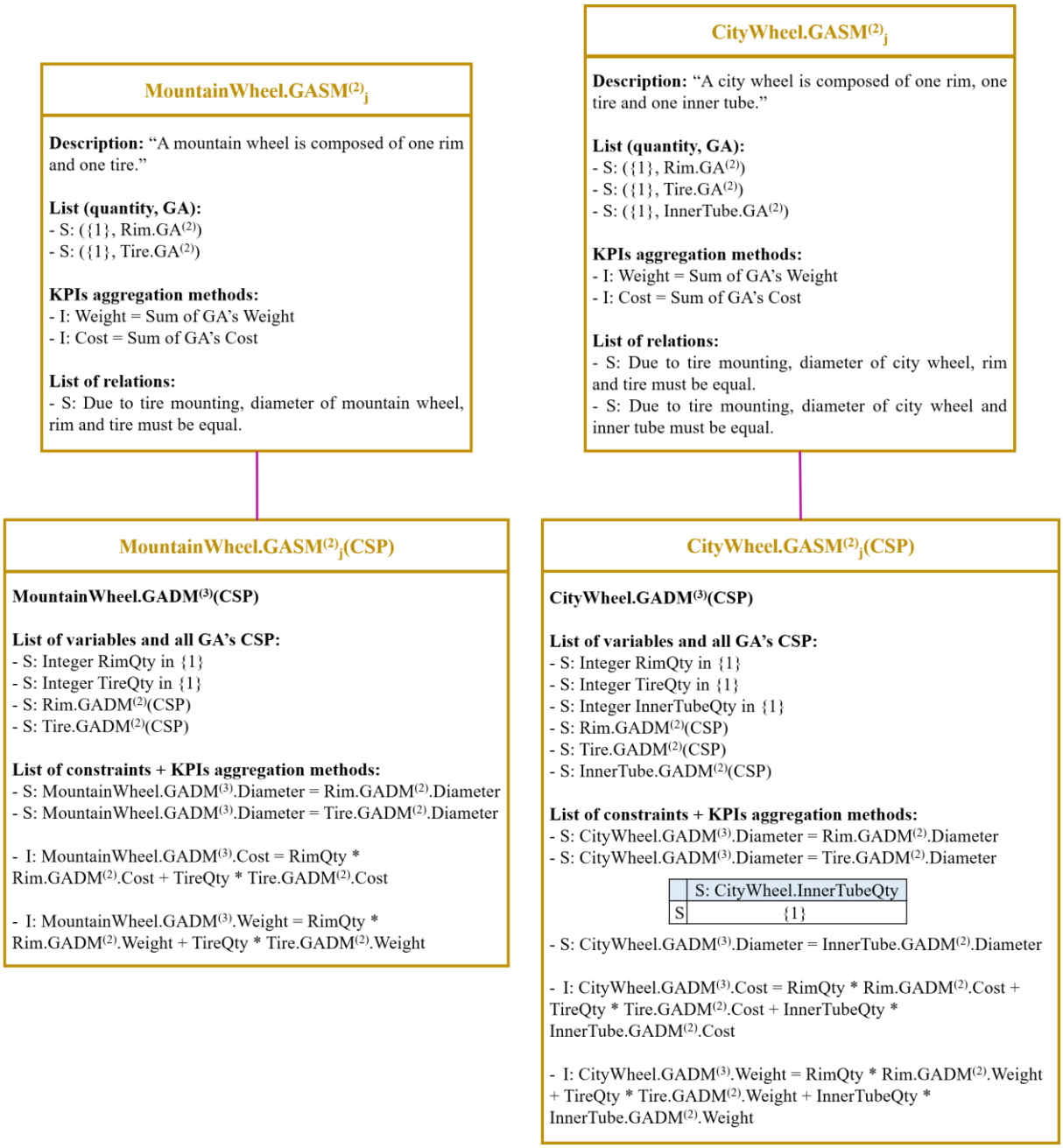

Figure 25. An example of *MountainWheel.GASM*$^{(2)}_j$ and *CityWheel.GASM*$^{(2)}_j$

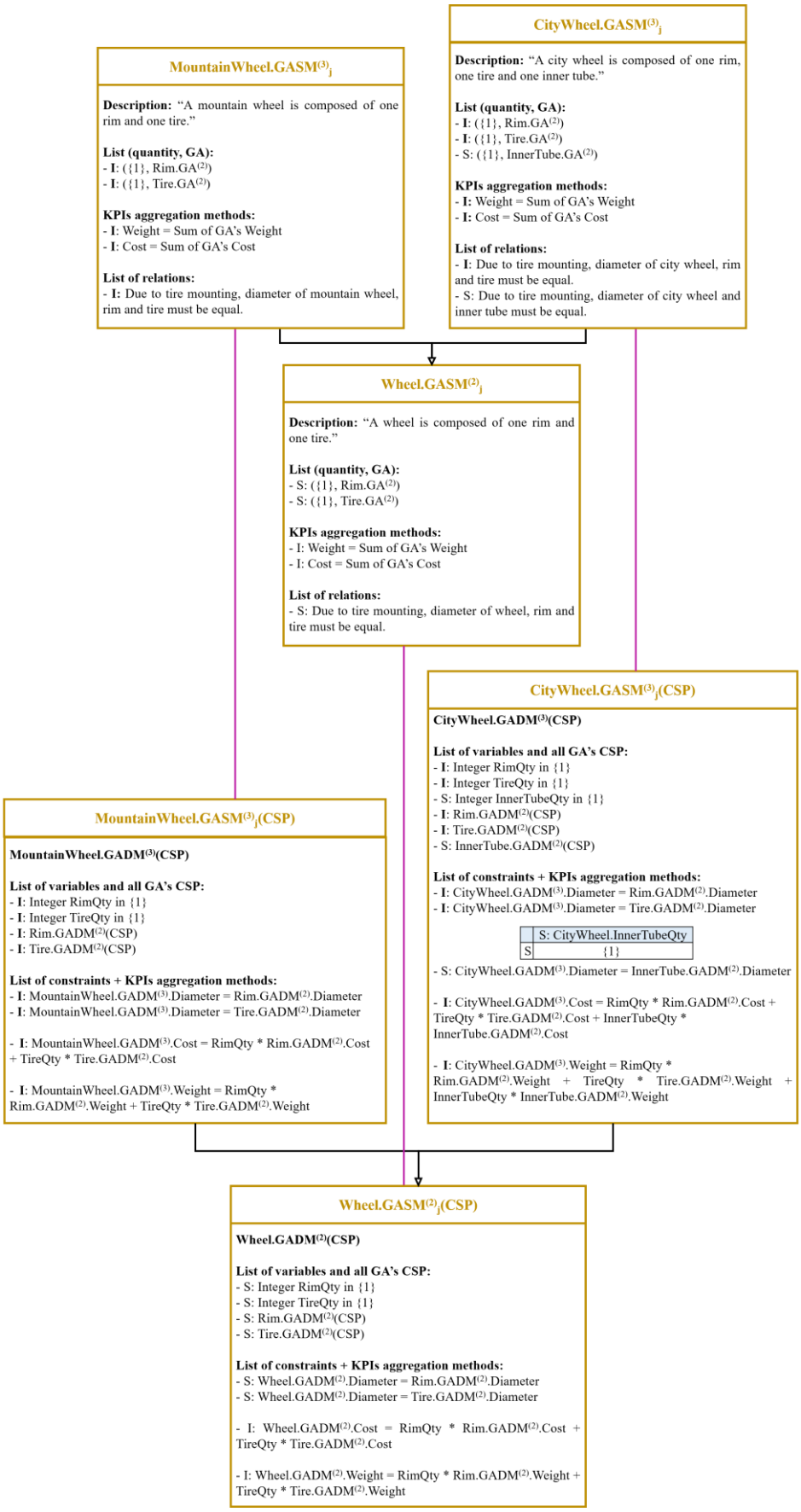

Figure 26. An example of *Wheel.GASM*$^{(2)}_{j}$ generalization

In the following section, the definition of GASM specialization is given.

Definition 11: Generic Artifact Structural Model Specialization or $GASM^{(i+1)}_j$
GASM specialization is the process of creating a more specific GASM, notated $GASM^{(i+1)}_j$, based on an existing $GASM^{(i)}_j$ and its modification (restriction and enrichment) with specific knowledge at a lower level of abstraction.

A general GASM $GASM^{(i)}_j$ can be specialized into another specialized GASM $GASM^{(i+1)}_j$ if $GA^{(i)}$ has already been specialized into $GA^{(i+1)}$. This GASM specialization is a four-step process:

Step 1) Duplicate the knowledge of $GASM^{(i)}_j$ into the knowledge of $GASM^{(i+1)}_j$: following the principles of specialization described in (Männistö *et al.*, 2001), we propose that $GASM^{(i+1)}_j$ inherits all the knowledge of its $GASM^{(i)}_j$, which includes all the GAs composing $GASM^{(i)}_j$ with their quantities, the methods to aggregate the KPIs of GAs, and all the relations linking GAs or attributes and/or KPIs of GAs. Consequently, in our proposed model, $GASM^{(i)}_j$ gathers all the common knowledge among all specialized $GASM^{(i+1)}_j$. This step facilitates the modeling process for experts as they can reuse the common knowledge capitalized in $GASM^{(i)}_j$ to build a new $GASM^{(i+1)}_j$.

Step 2) Modify the inherited name and description of $GASM^{(i+1)}_j$: after duplicating the knowledge of $GASM^{(i)}_j$ into $GASM^{(i+1)}_j$, we have to modify the name and description inherited from $GASM^{(i)}_j$. This requires changing the name of $GASM^{(i)}_j$ to $GASM^{(i+1)}_j$ anywhere within the $GASM^{(i+1)}_j$. Additionally, we modify the description of $GASM^{(i+1)}_j$ by providing more specific information than what was contained in the original description of $GASM^{(i)}_j$.

Step 3) Narrow the inherited knowledge of $GASM^{(i+1)}_j$: In order to guarantee the consistency of knowledge of GASM ontology, we strongly recommend that inherited knowledge can only be restricted. The expert can narrow or specialize the inherited knowledge of $GASM^{(i+1)}_j$ - GASM composition (quantity, $GA^{(i)}$), KPIs aggregation methods, and relations between $GA^{(i)}$, or between attributes and/or KPIs of different $GA^{(i)}$ - by 1) restricting the inherited quantities of $GA^{(i)}$, 2) replacing an inherited $GA^{(i)}$ by a more specialized $GA^{(i)}$ and 3) restricting the inherited relations between attributes and/or KPIs values of $GA^{(i)}$ which are explained as follows:

- Restrict the inherited quantities of $GA^{(i)}$: the expert can restrict the inherited quantities of GAs composing $GASM^{(i+1)}_j$. This allows them to i) remove an optional $GA^{(i)}$ (for example, change the quantity from {[0, 1]} to {0}), ii) consider an optional $GA^{(i)}$ as mandatory (e.g. change its quantity from {[0, 1]} to {1}), or iii) restrict the quantity of a mandatory $GA^{(i)}$ (for instance, from {[2, 4]} to {3}). Then, KPIs aggregation methods must be updated. Note that in the GASM specialization, enlarging the domains of quantities is not allowed.

- Replace inherited $GA^{(i)}$ by more specialized $GA^{(i)}$: in addition to restricting inherited quantities, the expert can also replace inherited GAs with more specialized ones in $GASM^{(i+1)}_j$. This specialization process is only allowed for GAs belonging to the same family line. For instance, $Wheel.GA^{(2)}$ can be replaced by $CityWheel.GA^{(3)}$ that is one of its descendants in the taxonomy of GAs. These replaced $GA^{(i)}$ must be considered in the KPIs aggregation methods. Replacing a $Wheel.GA^{(2)}$ by a $CityWheel.GA^{(3)}$ is allowed but replacing a $Seat.GA^{(2)}$ by a $Light.GA^{(2)}$ is not allowed.

- Restrict the inherited relations between $GA^{(i)}$ or between attributes and/or KPIs of $GA^{(i)}$: in $GASM^{(i+1)}_j$, the expert can restrict the inherited relations between $GA^{(i)}$ or between attributes and/or KPIs values of $GA^{(i)}$. However, it is not possible to remove inherited relations because they are inherited from the knowledge of $GASM^{(i)}_j$ that is common to all $GASM^{(i+1)}_j$.

Step 4) Enrich the knowledge of $GASM^{(i+1)}_j$: in this step, the expert can enrich the knowledge of $GASM^{(i+1)}_j$ by adding specific knowledge that only belongs to the specific family of artifacts. They can enrich the knowledge of $GASM^{(i+1)}_j$ by 1) adding new $GA^{(i)}$ specific to the $GASM^{(i+1)}_j$ with their quantities and 2) defining new relations between $GA^{(i)}$ or between attributes and/or KPIs values of $GA^{(i)}$ which are explained as follows:

- Add new $GA^{(i)}$ specific to $GASM^{(i+1)}_j$ with their quantities: one or more new $GA^{(i)}$ can be added to $GASM^{(i+1)}_j$ with their quantities only specific to the $GASM^{(i+1)}_j$. These newly identified $GA^{(i)}$ with their quantities enable the expert to model the $GASM^{(i+1)}_j$ based on specific knowledge of the family of artifacts that do not belong to its parent. These new $GA^{(i)}$ and their quantities must be considered in the KPIs aggregation methods.

- Define new relations between $GA^{(i)}$ or between attributes and/or KPIs values of $GA^{(i)}$: new relations only belonging to $GASM^{(i+1)}_j$ can be defined. These relations can link 1) the inherited $GA^{(i)}$ and new $GA^{(i)}$, 2) only new $GA^{(i)}$, 3) attributes and/or KPIs values of inherited $GA^{(i)}$ and new $GA^{(i)}$, and 4) attributes and/or KPIs values of new $GA^{(i)}$.

Since the GASMs are mapped to a CSP, the specialization follows exactly the same principle, notated $GASM^{(i+1)}_j(CSP)$:

- In $GASM^{(i+1)}_j(CSP)$, each $GA^{(i)}$ is substituted by its $GADM^{(i)}(CSP)$, and the quantities correspond to variables and domains, prefixed by I or S to know if attributes have been inherited or not. .

- KPIs aggregation methods are used as inherited (marked as 'I') or redefined as constraints linking all the relevant KPIs of the $GADM^{(i)}(CSP)$ composing the current $GASM^{(i+1)}_j(CSP)$.

- Relations are expressed in terms of compatibility tables and numerical functions on all relevant and available variables, coming either from the current $GASM^{(i+1)}_j(CSP)$ and all the variables of the $GADM^{(i)}(CSP)$, marked as I or S, as well as their tuples.

As shown in Figure 27, similar to the specialization of $A.GA^{(i)}$ (see Figure 17), the specialization of $A.GASM^{(i)}_j$ is created in which $A.GASM^{(i)}_j$ is specialized into $B.GASM^{(i+1)}_j$. Then, $B.GASM^{(i+1)}_j$ is mapped into CSP.

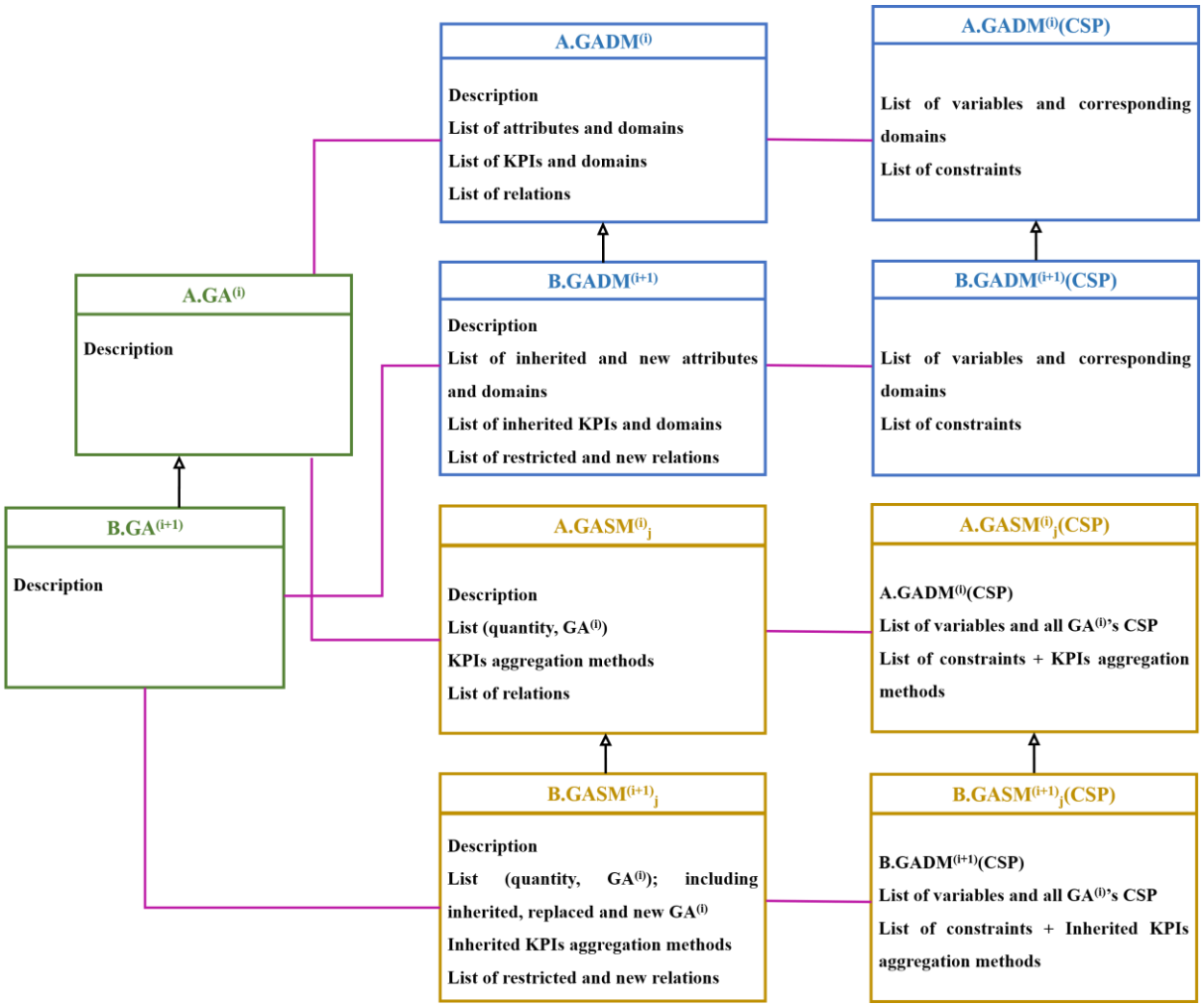

Figure 27. Specialization of $A.GASM^{(i)}_j$

The example of Figure 28 illustrates GASM specialization for families of bikes, where $Bike.GASM^{(2)}_j$ is specialized into $CityBike.GASM^{(3)}_j$. The GADMs are represented but not their *GADM(CSP)* for better clarity. $CityBike.GASM^{(3)}_j$ is translated into CSP ($CityBike.GASM^{(3)}_j(CSP)$). In $CityBike.GASM^{(3)}_j$, GAs with their quantities, KPIs aggregation methods and relations that are inherited from $Bike.GASM^{(2)}_j$, are tagged by 'I' while the GAs with their quantities and relations that are uniquely dedicated to $CityBike.GASM^{(3)}_j$ are tagged by 'S'.

In $CityBike.GASM^{(3)}_j$, $Wheel.GA^{(2)}$ is replaced by a more specialized $GA^{(i)}$ ($S: CityWheel.GA^{(3)}$) and its quantity is restricted to {2}. A new $GA^{(i)}$ is added ($S: Light.GA^{(2)}$) and a relation is restricted (the user 'Man' is only compatible with the material 'carbon'). A new relation is also added (between user and light color). Then, $CityBike.GASM^{(3)}_j$ is translated into the CSP ($CityBike.GASM^{(3)}_j(CSP)$), in which the CSP of the $CityBike.GADM^{(3)}$ is embedded. Moreover, in this CSP, the domain of the variable WheelQty is restricted by adding a compatibility table. The embedded CSP corresponding to $Wheel.GADM^{(2)}$ is replaced by a more specialized CSP

(*S: CityWheel.GADM$^{(3)}$(CSP)*). A new CSP is added (*S: Light.GADM$^{(2)}$(CSP)*). A new variable with its corresponding domain is added (S: LightQty). An existing constraint is restricted by adding a tuple which is tagged by 'S' (representing that if the user of city bike is a man, it is only compatible with the material carbon for the seat). Two constraints related to KPIs aggregation methods are updated. A new compatibility table is added (linking the color of light to the users of a city bike). Constraint filtering is applied leading to the restriction of the domains of the variables, removing inconsistent values.

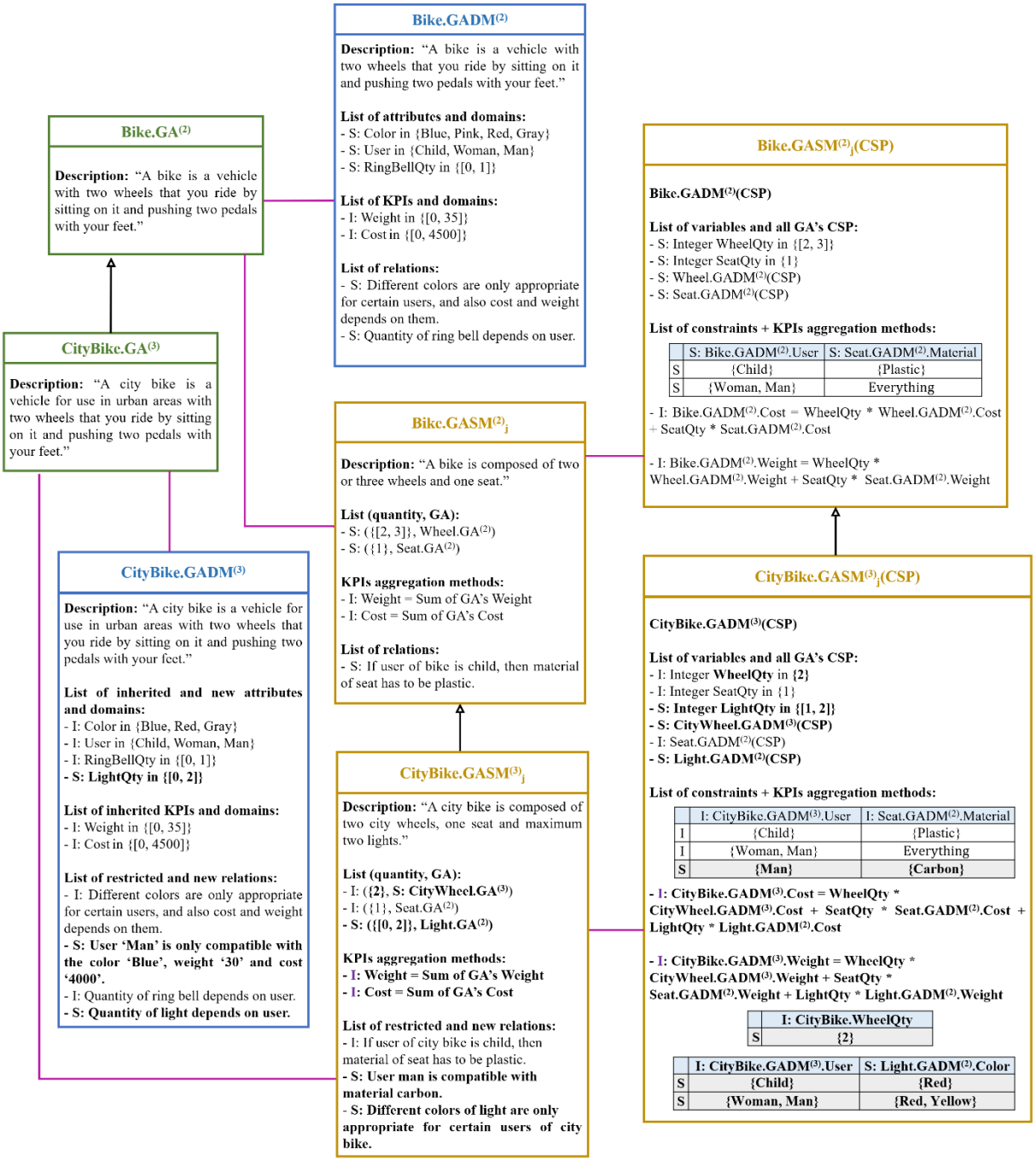

Figure 28. An example of *Bike.GASM$^{(2)}_j$* specialization

By applying the generalization or specialization process, a taxonomy of GASMs is created based on the GAs taxonomy to classify GASMs according to their common knowledge including the structure, KPIs aggregation methods, and relations between $GA^{(i)}$ or between attributes and/or KPIs of $GA^{(i)}$. An example of this taxonomy is illustrated in Figure 29 along with the GAs taxonomy (represented in Figure 19). The most general GASM ($System.GASM^{(1)}{}_1$), is described by common knowledge among all GASMs, and is the parent of several specialized GASMs, including $Bike.GASM^{(2)}{}_j$, $CityWheel.GASM^{(2)}{}_j$ and $MountainWheel.GASM^{(2)}{}_j$. $CityWheel.GASM^{(3)}{}_j$ and $MountainWheel.GASM^{(3)}{}_j$ are generalized into $Wheel.GASM^{(2)}{}_j$. Furthermore, $Bike.GASM^{(2)}{}_j$ is specialized into $CityBike.GASM^{(3)}{}_j$, and $MountainBike.GASM^{(3)}{}_j$. Since in GASM specialization, all specialized GASMs inherit the knowledge of their ancestors, $Bike.GASM^{(2)}{}_j$, and $Wheel.GASM^{(2)}{}_j$ inherit all the knowledge of $System.GASM^{(1)}{}_1$. $CityBike.GASM^{(3)}{}_j$ and $MountainBike.GASM^{(3)}{}_j$ inherit all the knowledge of $Bike.GASM^{(2)}{}_j$, including the knowledge of $System.GASM^{(1)}{}_1$. $CityWheel.GASM^{(3)}{}_j$ and $MountainWheel.GASM^{(3)}{}_j$ inherit all the knowledge of $Wheel.GASM^{(2)}{}_j$.

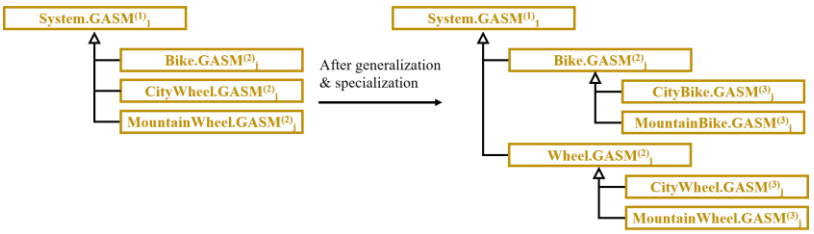

Figure 29. GASMs taxonomies

In the following, the synthesis of this section is provided.

3.2.4. Synthesis

In this section, we have focused on our first research question: "Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?" by defining the generalization and specialization process of generic models. Our approach involved defining and proposing methods for $GA^{(i-1)}$ generalization and $GA^{(i+1)}$ specialization. Based on that, we defined $GADM^{(i-1)}$ generalization and $GADM^{(i+1)}$ specialization as well as $GASM^{(i-1)}{}_j$ generalization and $GASM^{(i+1)}{}_j$ specialization.

Our proposed approach allows for the creation of GAs, GADMs, and GASMs at different levels of abstraction using commonality, generalization and specialization relations and by applying inheritance principles which can help to better manage knowledge separating their descriptive views and structural views and defining three taxonomies of $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}{}_j$. These three taxonomies are closely related and they are the ontology for system configuration.

In the following section, we explain how we can update GAs, GADMs, and GASMs.

3.3. Update of $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$

The experts need to frequently update generic models due to several reasons. Firstly, the structure of the system may change over time due to new GAs being added and old GAs being removed. Therefore, updating the generic models is essential to ensure that it accurately represents the current structure of the systems. Secondly, as new information becomes available about the modeled systems, generic models may need to be updated to incorporate this information. For example, if new information is discovered about the relations between components, the model may need to be updated to reflect this. Thirdly, as new techniques or methods are developed for creating systems, the model may need to be updated. Therefore, this helps to keep generic models up to date and improve them.

This section is dedicated to the update of $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$. Therefore, first, in section 3.3.1, the GA update is proposed, followed by the GADM update in section 3.3.2, and then the update of GASM is defined in section 3.3.3. Finally, the synthesis of the section is represented in section 3.3.4.

3.3.1. Update of Generic Artifacts

A GA can undergo one or multiple updates following the evolution of designs, technologies, components, etc. The definition of GA update is given below.

Definition 12: Generic Artifact update
GA update is a process of updating a current $GA^{(i)}$ by modifying or updating its knowledge.

To update a $GA^{(i)}$, its description must be updated.

As represented in the example of Figure 30, $Bike.GA^{(2)}$ is updated only by updating its description. As the description of $Bike.GA^{(2)}$ has been updated, the expert has to check the description of all the descendants of $Bike.GA^{(2)}$ and modify them if necessary. In the example of Figure 30, the description of the $Bike.GA^{(2)}$ is shortened as well as the description of the $CityBike.GA^{(3)}$ and the description of $MountainBike.GA^{(3)}$. These modifications have to be done manually by the expert by checking all the descriptions of the descendants of the updated $GA^{(i)}$ ($Bike.GA^{(2)}$ in this example).

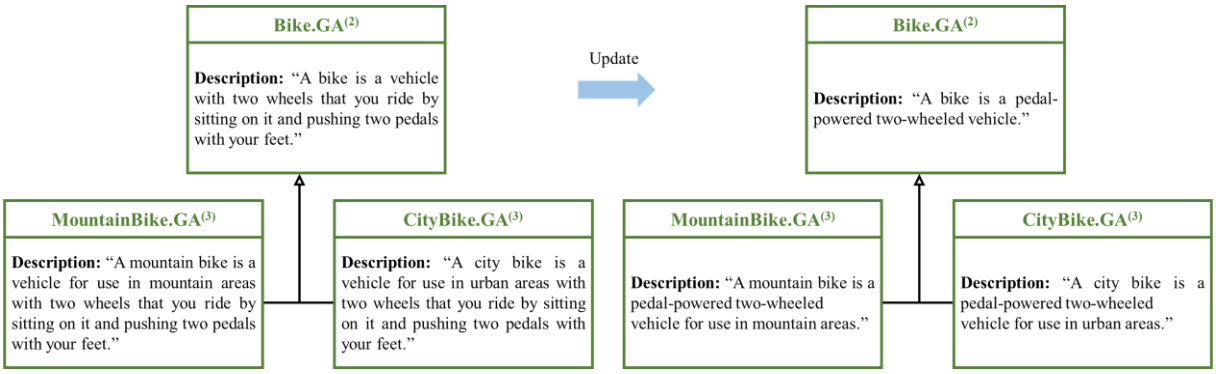

Figure 30. An example of $GA^{(i)}$ update for a family of bikes

The following section is dedicated to the $GADM^{(i)}$ update.

3.3.2. Update of Generic Artifact Descriptive Models

GADMs need to be updated because the characteristics of the system may be changed or modified over time, new attributes may be added, or old attributes may be removed. Thus, a GADM can undergo one or multiple updates.

Definition 13: Generic Artifact Descriptive Model Update
GADM update is a process of updating the current $GADM^{(i)}$ by updating (modifying, removing or adding) its knowledge including elements such as attributes with their domains, KPIs with their domains, and relations between attributes and/or KPIs values.

$GADM^{(i)}$ update is a four steps process:

Step 1) Update description of $GADM^{(i)}$,

Step 2) Update attributes with their domains,

Step 3) Update KPIs with their domains,

Step 4) Update relations between attributes and/or KPIs values.

All modifications made to the $GADM^{(i)}$ are applied into $GADM^{(i+1)}$ and its descendants as well, making it easy for experts to design and update GADMs.

We map the updated $GADM^{(i)}$ within its CSP ($GADM^{(i)}(CSP)$) as follows.

Step 1) Update attributes with their domains corresponds to updating the variables with their domains,

Step 2) Update KPIs with their domains corresponds to updating the variables with their domains,

Step 3) Update relations corresponds to updating the constraints.

Then, the consistency of the updated $GADM^{(i)}$ has to be checked by propagating the constraints.

The example of Figure 31 illustrates the updating of the $Bike.GADM^{(2)}$ of Figure 13, $Bike.GADM^{(2)}$ is updated by removing values of the attribute Color (values Pink and Gray are removed), removing an attribute (RingBellQty is removed), adding a new attribute with its domain (Size ∈ {XS, S, M, L, XL}), restricting the domain of the Cost KPI (Cost ∈ {[500, 4500]}) as well as removing the relation related to the removed attribute (RingBellQty). $Bike.GADM^{(2)}(CSP)$ is updated as well. The values Pink and Gray are removed from the domain of variable Color and also from the constraint, the variable RingBellQty is removed, the variable Size is added with its domain, the constraint which linked RingBellQty with the User type is removed, the domain of cost is restricted (its lower bound is changed to '500'), and the lower bound of cost in the compatibility table is also changed to '500'. Then, constraint filtering is applied and the domains of variables are restricted.

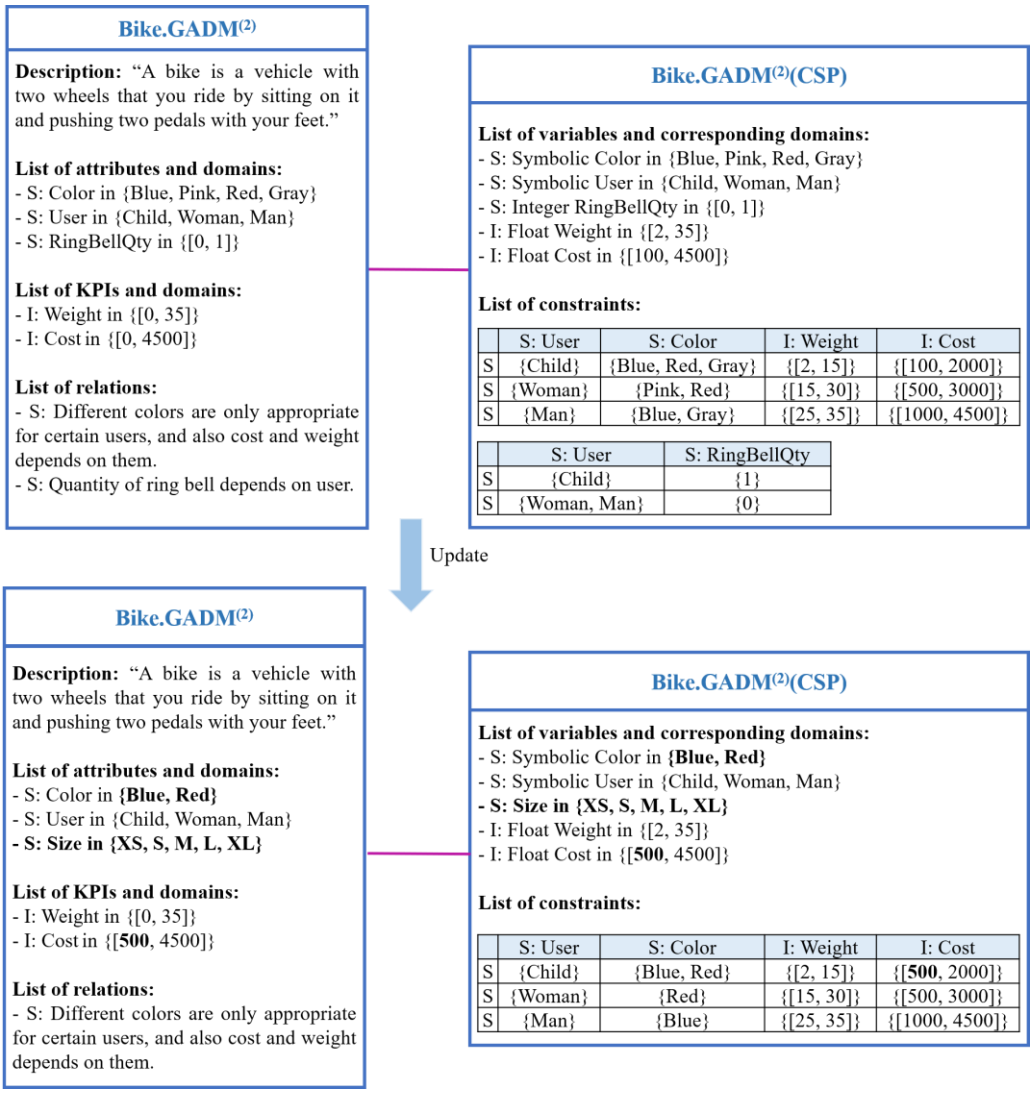

Figure 31. An example of *Bike.GADM*$^{(2)}$ update

All the modifications made in *Bike.GADM*$^{(2)}$ are propagated to all the descendants. This means that *CityBike.GADM*$^{(3)}$ and *MountainBike.GADM*$^{(3)}$ will be updated as well. However, every modification must be changed by the expert in charge of the knowledge update. As shown in Figure 32, in *CityBike.GADM*$^{(3)}$, the Gray value of the attribute Color is removed, the attribute RingBellQty is removed, Size that is a new attribute with its domain is added, the domain of the Cost KPI is restricted to {[500, 4500]}, the relation related to the deleted attribute RingBellQty is removed. The CSP of *CityBike.GADM*$^{(3)}$ is also updated. The value Gray is removed from the domain of the variable Color as well as from the constraint. The variable RingBellQty is removed. The variable Size is added with its domain. The constraint linking RingBellQty to User is removed. The domain of cost is restricted. Then, constraint filtering is applied and the domains of variables are restricted.

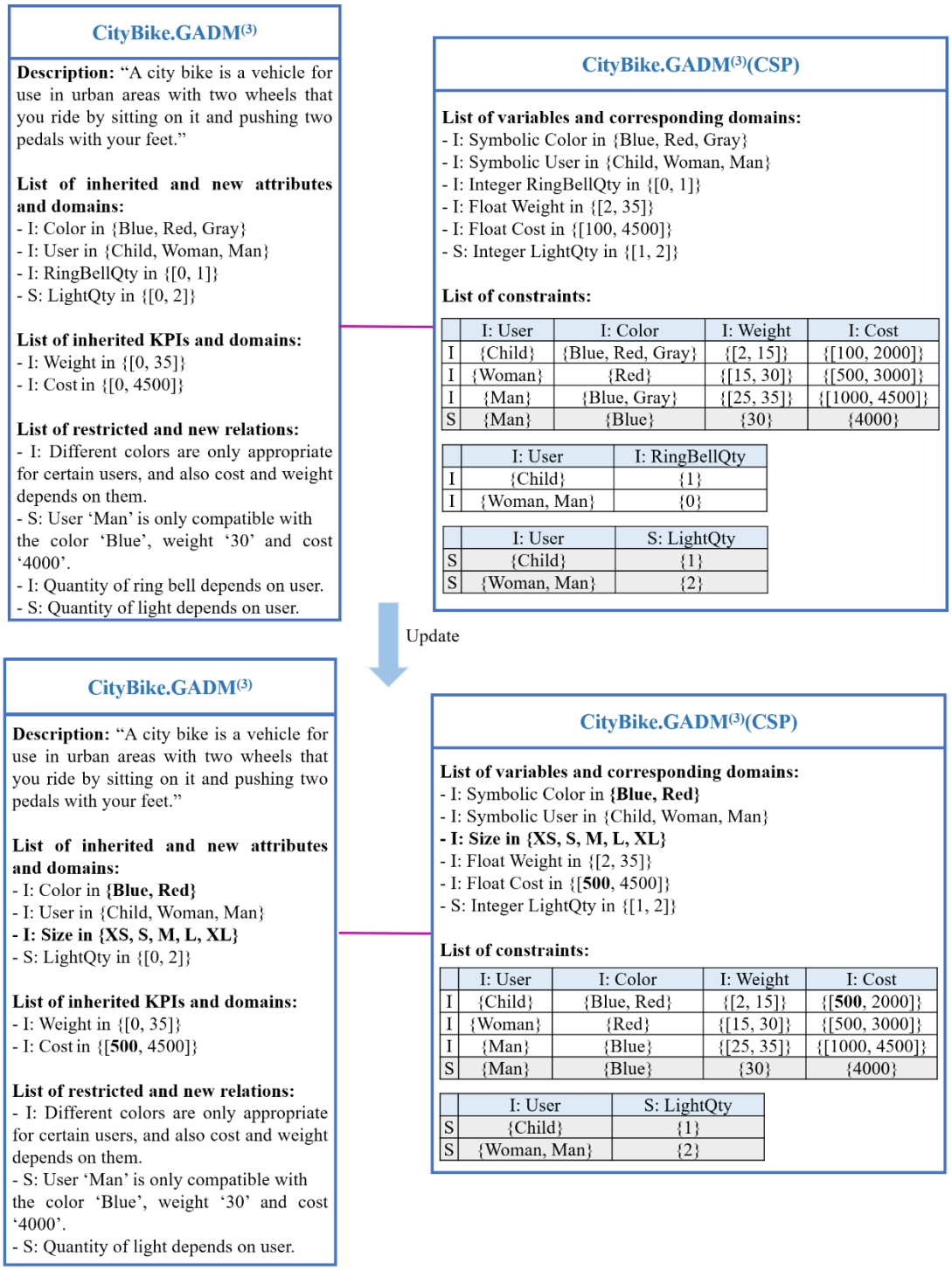

Figure 32. Example of propagation of *Bike.GADM(2)* updates into *CityBike.GADM(3)*

The following section is dedicated to the $GASM^{(i)}_{j}$ update.

3.3.3. Update of Generic Artifact Structural Models

A $GASM^{(i)}_j$ needs to be updated when the structural view of the system has changed. New $GA^{(i)}$ may be added, $GA^{(i)}$ may be removed or $GA^{(i)}$ may be changed into the structure.

Definition 14: Generic Artifact Structural Model Update
GASM update is a process of updating a current $GASM^{(i)}_j$ by updating (modifying, removing or adding) its knowledge including GASM composition (quantity, $GA^{(i)}$), KPIs aggregation methods, and relations between $GA^{(i)}$, or between attributes and/or KPIs of $GA^{(i)}$.

This $GASM^{(i)}_j$ update is a four steps process:

Step 1) Update the description of $GASM^{(i)}_j$,

Step 2) Update identified $GA^{(i)}$ belonging to $GASM^{(i)}_j$ with their quantities,

Step 3) Update aggregation methods for each KPI of $GASM^{(i)}_j$,

Step 4) Update relations between $GA^{(i)}$ or between attributes and/or KPIs values of $GA^{(i)}$.

Note that any modifications made to $GASM^{(i)}_j$ are inherited by its specialized $GASM^{(i+1)}_j$ and all its descendants, which helps experts to design and update GASMs.

We propose to map the updated $GASM^{(i)}_j$ within CSP ($GASM^{(i)}_j(CSP)$ is updated).

Step 1) Update identified $GA^{(i)}$ with their quantities corresponds to updating existing CSPs and the domains of variables,

Step 2) Update relations corresponds to updating the constraints. It also includes updating the constraints related to KPIs aggregation methods.

Then, the consistency of the updated $GASM^{(i)}_j$ needs to be checked by filtering the constraints.

As illustrated in Figure 33, $Bike.GASM^{(2)}_j$ (represented in Figure 15) is updated by replacing $Wheel.GA^{(2)}$ with $Wheel_2.GA^{(2)}$ and restricting its quantity to 3, changing the quantity of *Seat.GA* from 1 to 2, and adding a new $GA^{(i)}$ ($Frame.GA^{(2)}$). Then, $Bike.GASM^{(2)}_j(CSP)$ is updated as well. The domain of variable $Wheel_2Qty$ is changed to 3, the domain of variable SeatQty is changed to 2, a new variable is added (FrameQty), $Wheel.GADM^{(2)}(CSP)$ is replaced with $Wheel_2.GADM^{(2)}(CSP)$, and a new CSP is added i.e. $Frame.GADM^{(2)}(CSP)$. Two constraints corresponding to KPIs aggregation methods are updated by taking into account $Wheel_2.GA^{(2)}$, *and* $Frame.GA^{(2)}$.

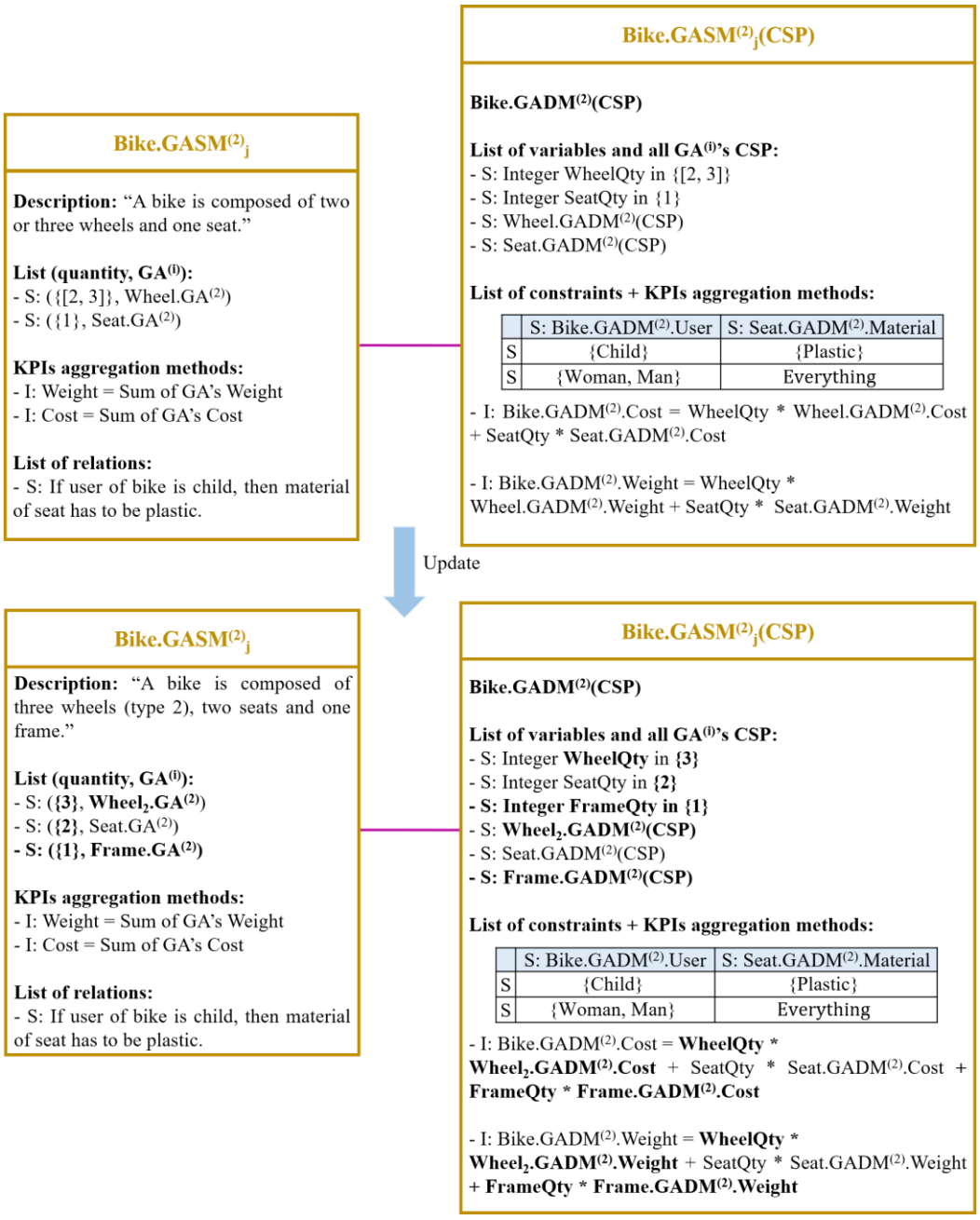

Figure 33. An example of *Bike.GASM*$^{(2)}_j$ update

Every modification in *Bike.GASM*$^{(2)}_j$ is inherited by all its descendants. However, in Figure 34, only the case of *CityBike.GASM*$^{(3)}_j$ is shown. Every modification must be changed by the expert. As shown in Figure 34, *Wheel.GA*$^{(2)}$ is replaced with *Wheel*$_2$*.GA*$^{(2)}$ (Wheel$_2$ is a GA corresponding to another family of wheels) and its quantity is restricted to 3, the quantity of *Seat.GA*$^{(2)}$ is changed to 2, and *Frame.GA*$^{(2)}$ is added. The CSP of *CityBike.GASM*$^{(3)}_j$ is also updated. The domains of variables WheelQty and SeatQty are respectively changed to 3 and 2, FrameQty which is a new variable is added, *Wheel.GADM*$^{(2)}$*(CSP)* is replaced with *Wheel*$_2$*.GADM*$^{(2)}$*(CSP)*, and *Frame.GADM*$^{(2)}$*(CSP)* is added. Two constraints corresponding to KPIs aggregation methods are updated by considering *Wheel*$_2$*.GA*$^{(2)}$*, and Frame.GA*$^{(2)}$.

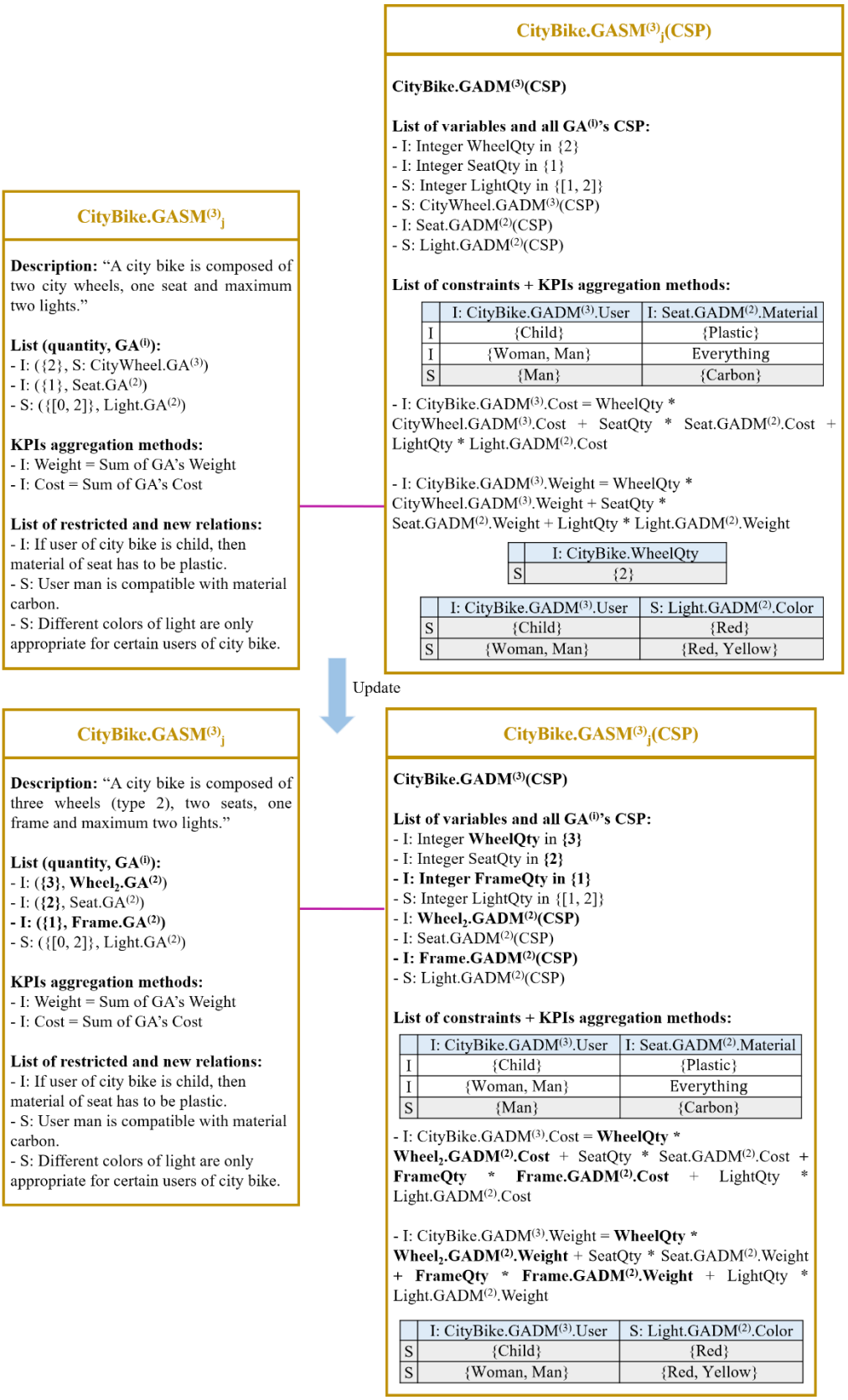

Figure 34. Example of propagation of *Bike.GASM*$^{(2)}_j$ updates to *CityBike.GASM*$^{(3)}_j$

The synthesis of this section is represented as follows.

3.3.4. Synthesis

In this section, we have defined processes that enables to update $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$. After their definition, we have explained how their corresponding CSP can be updated. This proposal facilitates the maintenance and updates of generic models since when a GA, GADM, and GASM at a higher level of abstraction is updated, the changes or modifications are propagated into all the lower-level GAs, GADMs and GASMs (all the descendants) by the expert. Therefore, it is not required to modify descendants separately. Consequently, it reduces the efforts required for managing and updating formalized knowledge. Moreover, when a GA, GADM or GASM is updated, the propagation of the updates requires to check the validity of all the descendants. Therefore, that participates to the content of validity of the formalized knowledge.

The following section is dedicated to the discussion of our proposed approach.

3.4. Single-model approach or multi-model approach?

To formalize and manage generic models in the domain of system configuration, assuming a set of system families, a single-model approach**,** following the proposal of Männistö (Männistö et al., 2001) and a multi-model approach can be employed.

In the single-model approach (Männistö et al., 2001), only one generic model is created that represents all the families of systems. This generic model incorporates all the knowledge and diversity relevant to all families of systems. As the model becomes more complex, it can become difficult to maintain, and update. It can also become difficult to comprehend and interpret it because it contains all the knowledge for the families of systems. The knowledge about the different families of systems is mixed up. This can lead to errors and inconsistencies, and it can also make it difficult to identify and resolve problems when they occur. The generic model can become too large and complex, making validation and knowledge maintenance difficult to perform.

On the other hand, the multi-model approach consists of multiple (several) generic models, each one representing a specific family of systems. This means that each generic model contains only the knowledge and diversity relevant to a specific family of systems. Generic models are simpler, and the knowledge is easier to understand. However, there may be redundant knowledge modeling when two system families share similar components or attributes (Erens & McKay, 1994).

The use of a single-model approach can often result in a complex model, which may be difficult to comprehend, maintain and update. On the other hand, a multi-model approach may lead to redundancy, where similar models share common knowledge. However, our proposed approach addresses these concerns by using the concept of commonality of generic models and through generalization and specialization relations as well as inheritance principles, which make it possible to define generic models at different levels of abstraction.

In Figure 35, a single approach is represented for a simplified bike example while in Figure 36, a multi-model approach is shown. In Figure 37, our approach is represented in which there

exists a bike model at a higher level of abstraction containing the commonality of lower level models. This model is created by generalizing the CityBike model and MountainBike model. Conversely, the Bike model is specialized into the CityBike model and MountainBike model.

When a new bike needs to be designed, the generic bike model can be reused and specialized. This reduces the amount of knowledge modeling activity that is required. The experts can focus on innovation while reusing the elements that remain common to all bikes.

In the different sections, we have presented a taxonomy of Generic Artifacts (GA), a taxonomy of Generic Artifact Descriptive Models (GADM), and a taxonomy of Generic Artifact Structural Models (GASM). All these taxonomies are strongly connected and they constitute an ontology for system configuration. In the different models, we have associated constraint satisfaction problems (CSP). The main advantage is that they constitute a powerful formalism to model knowledge for system configuration. They allow to formalize all the allowed characteristics for systems, subsystems, and components. Using filtering techniques, it is possible to check the consistency of the models and remove inconsistent characteristics. Therefore, it is a first step for validating the formalized knowledge before storing it.

In the proposed modeling approach, we have dissociated descriptive views from structural views. This is important for knowledge reuse. Either the expert of the knowledge is only interested in descriptive characteristics of the systems to configure (the colors, the quantity of windows, the power of the engine, etc.), or she/he is interested in their deep structure (the choice of components, their bill of material, the characteristics of a specific component, etc.)

In the proposed modeling approach, generic models are defined at various levels of abstraction which facilitate the maintenance and update of generic models. If the expert wants to change the characteristics of the systems (change the cost, update the colors, add a new attribute…) or change the structure of systems (replace a wheel by a new one, add a new component to the system, etc.), he/she can update the generic model at a higher level of abstraction, in that case, the modifications are inherited by all the descendants. On the other hand, if the expert wants to create new systems, he/she can reuse the generic model at a higher level of abstraction since it contains the commonality of several systems and then enrich it with specific knowledge which is only dedicated to the relevant system. Therefore, the expert can define new generic models at a lower level of abstraction which are much more specific.

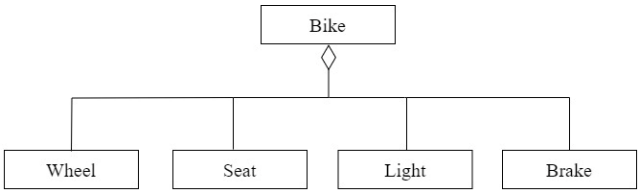

Figure 35. An example of single-model approach

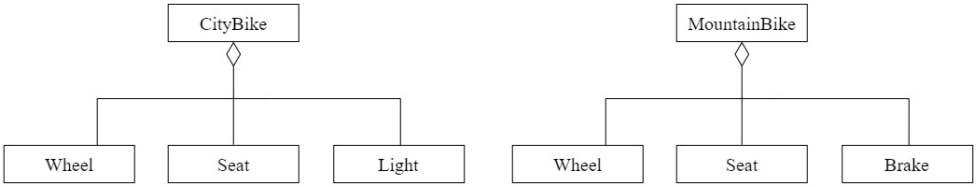

Figure 36. An example of multi-model approach

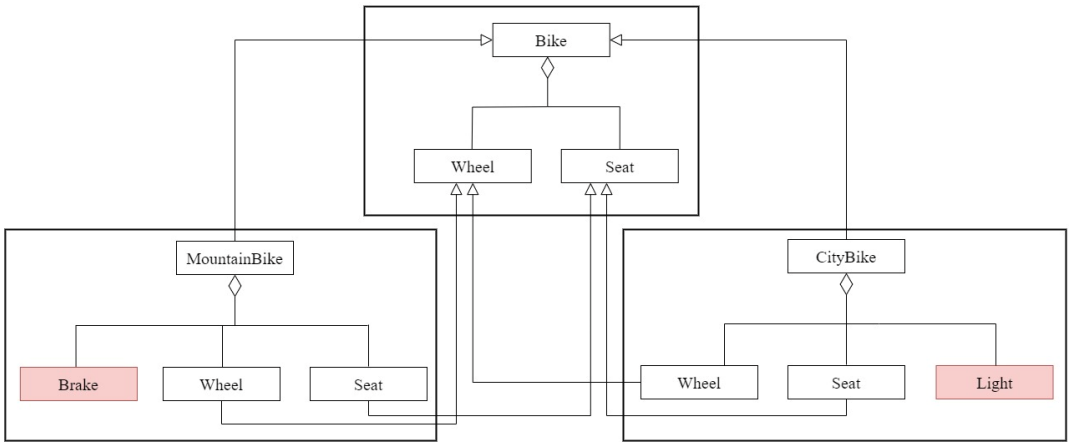

Figure 37. An example of our proposed approach

In the following, the synthesis of this chapter is represented.

3.5. Synthesis

The first objective of our research work is to formalize knowledge for system configuration. This chapter presents our first contribution, a knowledge formalization process for system configuration using the association of ontologies, CSP approaches, and inheritance principles to create generic models, which comes in response to our first research question: "Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?"

Our proposal in section 3.1, allows modeling knowledge about a family of artifacts and creating generic models while distinguishing descriptive views and structural views (KFC1). Therefore, after defining a GA, we describe its descriptive view by a GADM, and its structural view by GASM if necessary. We formalize each of these views as CSP to check their consistency during the formalization process (KFC4) and illustrate our proposal on a simple bike example.

Our proposal in section 3.2, allows for creating an ontology of generic models. We proposed processes for GA generalization and specialization of $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}{}_{j}$, then their translation into CSP. It allows creating generic models at different levels of abstraction (KFC3), structuring them from the most general one to the most specialized one, using generalization and specialization relations and by employing the concept of commonality and inheritance principles. This approach allows us to better manage knowledge, facilitate maintenance, and update (KFC2). Depending on whether the attributes, indicators, and relationships of the GA, GADM, and GASM are inherited or specific to the element, they are tagged with 'I' or 'S' respectively. This ensures the consistency of the model ontologies by tracing the origin of the knowledge.

Our proposals in section 3.3, enable us to update generic models to improve them and keep them up to date. Therefore, first, we defined the update of GA. Our proposals allow us to better manage knowledge since modifications applied on the $GA^{(i)}$, $GADM^{(i)}$, and $GASM^{(i)}_j$ at a high level of abstraction are propagated to all the GAs, GADM, or GASMs at lower levels of abstraction.

This contribution was featured in the three following conference papers:

- Maryam Mohammad Amini, Thierry Coudert, Élise Vareilles, Michel Aldanondo. Integration of Ontologies and Constraint Satisfaction Problems for Product Configuration. IEEM 2021 - International Conference on Industrial Engineering and Engineering Management, Dec 2021, Singapore, France. pp.578-582,
- Maryam Mohammad Amini, Thierry Coudert, Élise Vareilles, Michel Aldanondo. System Configuration Models: Towards a Specialization Approach. MIM 2022 - 10th IFAC Conference on Manufacturing Modelling, Management and Control, Jun 2022, Nantes, France. pp.1189 - 1194, ⟨10.1016/j.ifacol.2022.09.551⟩.
- Élise Vareilles, Thierry Coudert, Michel Aldanondo, Maryam Mohammad Amini. Capitalisation de connaissances en configuration de biens et de services : vers une meilleure gestion de la communalité des modèles. CIGI QUALITA MOSIM 2023, Jun 2023, Trois-Rivières, Canada. 8 p.

The next chapter begins the second step of this thesis, dedicated to reusing formalized knowledge. This chapter contains our second contribution, devoted to our second research question, "How can ETO requirements be processed during configuration activity? "

4. Knowledge reuse for system configuration

This chapter is dedicated to our second research question: "How can ETO requirements be processed during configuration activity?" In chapter 2, we have highlighted the absence of study about reusing formalized generic models either descriptive or structural, and at various levels of abstraction to fulfill non-negotiable non-standard requirements of users in ETO situations. In response to this gap, we propose a reusing process in this chapter, which constitutes our second contribution.

Before moving on to ETO situations, we start with CTO situations where system configuration is performed. Therefore, we formalize knowledge reuse in CTO situations in Section 4.1, where the objective is to meet the CTO requirements as defined in Chapter 2 taking into account our proposals. In section 4.2 we describe the knowledge reuse step in ETO situations, where the objective is to meet the ETO requirements of the users, as defined in Chapter 2. In section 4.3, we explain the link between the knowledge reuse step in CTO and ETO situations. Finally, in section 4.4 we summarize the chapter.

As illustrated in Figure 38, we adopt the knowledge management process explained in chapter 2, for system configuration knowledge reuse. All the generic models that exist in the GMB can be shared. Then, stored generic models are reused to meet user requirements in both CTO and ETO situations. Finally, at the end of the configuration activity, the results of the configuration, i.e. the solutions (outputs) are stored in an Experience Base (EB). Experts can use the EB to update the generic models (as explain in Chapter 3.4) to add new solutions to the catalog. However, this topic is out of the scope of this thesis.

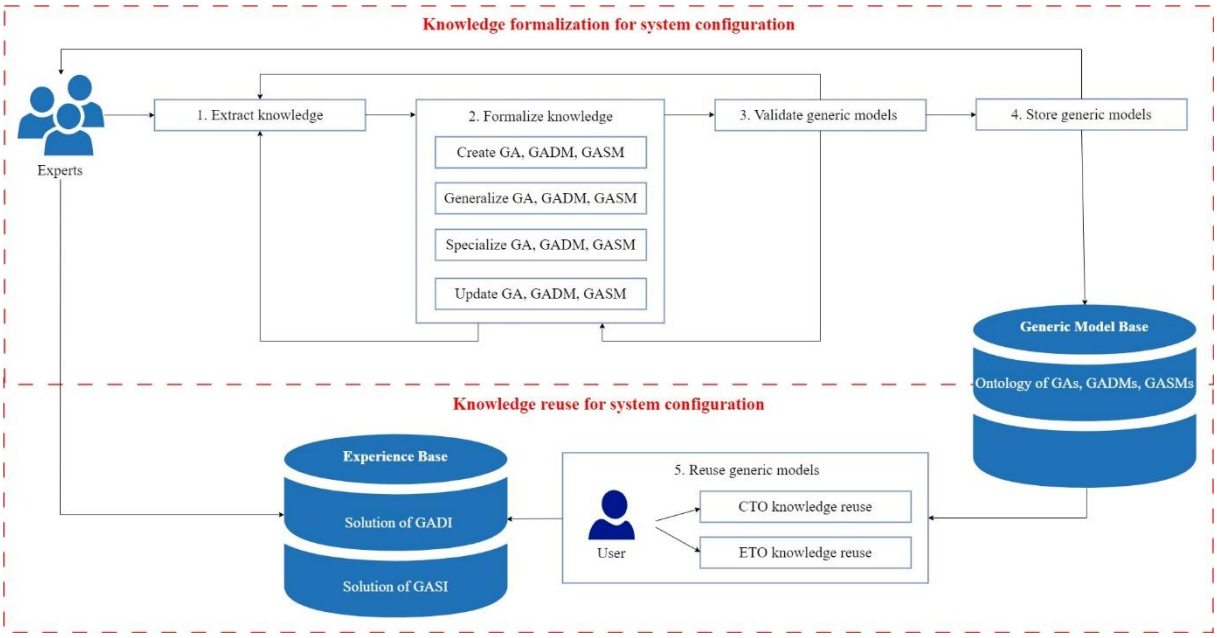

Figure 38. Knowledge reuse for system configuration

4.1. CTO knowledge reuse for system configuration

This section is dedicated to knowledge reuse in CTO situations through which system configuration can be achieved. First, in section 4.1.1, configuration activity in CTO situations is presented with regards to our proposals. Then, in section 4.1.2, the consequence and impact of the previous modeling propositions are clearly discussed. Finally, in section 4.1.3, the synthesis of the section is provided.

4.1.1. CTO configuration activity

First, we describe how our proposals on knowledge formalization can be applied in CTO situations where all requirements expressed by the user can be met by exploiting formalized generic models stored in the GMB. In this thesis, we refer to configuration activity in CTO situation as "CTO configuration activity". As defined in chapter 2, we consider CTO Requirements as standard requirements (or requirements expressed as such), whether negotiable or non-negotiable, meaning that the choices made by a user are consistent with the current attributes' domains. In CTO configuration activity, CTO requirements can be fulfilled as they are supported by the generic models. To achieve this, after selecting a relevant generic model, two topics play an important role: 1) generic model instance creation and 2) generic model instance processing. In the first one, the definitions of GA, GADM and GASM instances, respectively GAI, GADI and GASI are provided and in the second one, the configuration of these instances in order to reach a solution is described.

4.1.1.1. CTO configuration principles

In order to configure a system which meets the CTO requirements, the user needs first to identify what the generic models necessary for the configuration according to descriptive and structural views are.

CTO configuration activity refers to an interactive process by which systems are tailored to meet CTO requirements. It involves defining user requirements, applying constraint filtering mechanisms, and deleting inconsistent solutions according to the constraints and choices made by the user until reaching a solution.

By reading the GA name and description (as defined in Chapter 3, definition 3), a GA corresponding to the family of systems the user wants to configure is selected in the taxonomy of GA.

As mentioned in chapter 3, a GA is associated to its specific GADM which corresponds to its descriptive view. The GA can also be linked to various GASM which describe several versions of its structural view. Therefore, once the user has selected the GA, its associated GADM is automatically selected and the user have to decide whether selecting a version of GASM (if it exists) or not. It depends on the kind of configuration the user wants to carry out. If the user wants to configure a system with only the descriptive view, only the GADM is necessary. If the user is also interested by the structural view of the system to configure, one GASM has to be selected among the different available versions.

4.1.1.2. Generic model instances

In order to configure a system, it is necessary to create instances of generic models formalized during the knowledge formalization process (see chapter 3). These instances are: Generic Artifact Instance (GAI), Generic Artifact Descriptive Instance (GADI) and Generic Artifact Structural Instance (GASI). These instances are defined as follows:

Definition 15: Generic Artifact Instance or $GAI^{(i)}$ in CTO
A Generic Artifact Instance, notated $GAI^{(i)}$ is an instance of the selected $GA^{(i)}$, at a certain level of abstraction i, which is created during the CTO configuration activity in order to answer to CTO requirements expressed by a user. It is structured as the corresponding $GA^{(i)}$ (with a description). We denoted it as $CTO.GAI^{(i)}$.

For instance, an instance of $Bike.GA^{(i)}$ is denoted $CTO.Bike.GAI^{(i)}$.

Definition 16: Generic Artifact Descriptive Instance or $GADI^{(i)}$ in CTO
A Generic Artifact Descriptive Instance, notated $CTO.GADI^{(i)}$ is the instance of the $GADM^{(i)}$ associated to a selected $CTO.GAI^{(i)}$. It contains the same elements than the corresponding GADM: description, list of attributes and domains, list of KPIs and domains and list of relations.

This *CTO.GADI*$^{(i)}$ is used during the CTO configuration as the descriptive view. Within the *CTO.GADI*$^{(i)}$, the attributes domains and KPIs domains will be restricted following the user's requirements and the relations between attributes using filtering mechanisms. Thus, every *CTO.GADI*$^{(i)}$ is mapped into a corresponding CSP denoted *CTO.GADI*$^{(i)}$*(CSP)*.

Definition 17: Generic Artifact Structural Instance or *GASI*$^{(i)}_j$ in CTO

A Generic Artifact Structural Instance, notated *GASI*$^{(i)}_j$ is the instance of a *GASM*$^{(i)}_j$ when a *GASM*$^{(i)}_j$ is chosen by the user. It is used when a structural view of the *GAI*$^{(i)}$ is necessary for the CTO configuration. As a *GASM*$^{(i)}_j$ contains a list of couples (quantity, *GA*$^{(i)}$) corresponding to its structure, for every *GA*$^{(i)}$ in the *GASM*$^{(i)}_j$, an instance *CTO.GAI*$^{(i)}$ is created in the *GASI*$^{(i)}_j$ (see Figure 39). A *GASI*$^{(i)}_j$ is mapped into its corresponding CSP denoted *GASI*$^{(i)}_j$*(CSP)*. We denoted it as *CTO.GASI*$^{(i)}_j$.

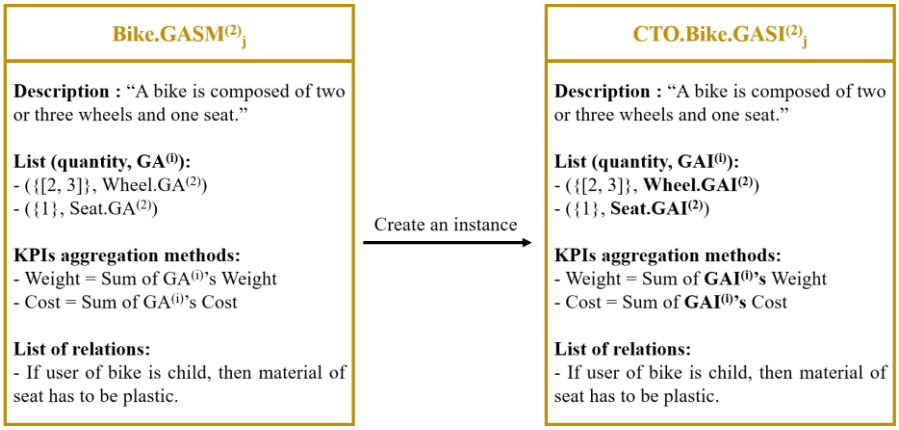

Figure 39. Example of creation of an instance of a *CTO.Bike.GASM*$^{(2)}_j$

The UML diagram of Figure 40 illustrates the different generic models, their instances and the relations between them. For a specific CTO configuration activity, a *CTO.GAI*$^{(i)}$ is an instance of a *GA*$^{(i)}$. A *GA*$^{(i)}$ can be associated to one or zero *CTO.GAI*$^{(i)}$. If a *GA*$^{(i)}$ has not been selected by the user, that *GA*$^{(i)}$ will have no associated *CTO.GAI*$^{(i)}$. A *CTO.GADI*$^{(i)}$ is an instance of a *GADM*$^{(i)}$, which is mapped into its *CTO.GADI*$^{(i)}$*(CSP)*. A *GADM*$^{(i)}$ can be associated to zero or one *CTO.GADI*$^{(i)}$*(CSP)* (if a *GA*$^{(i)}$ is selected for the CTO configuration, its *CTO.GADI*$^{(i)}$ is instantiated). A *CTO.GASI*$^{(i)}_j$ is an instance of a *GASM*$^{(i)}_j$ which is mapped into its *CTO.GASI*$^{(i)}_j$*(CSP)*. A *CTO.GASI*$^{(i)}_j$ is created only when the structural view is considered as necessary by the user during the CTO configuration. For a *CTO.GAI*$^{(i)}$, only one *CTO.GADI*$^{(i)}$ is built and one or zero *CTO.GASI*$^{(i)}_j$ is built (following the necessity of structural view or not).

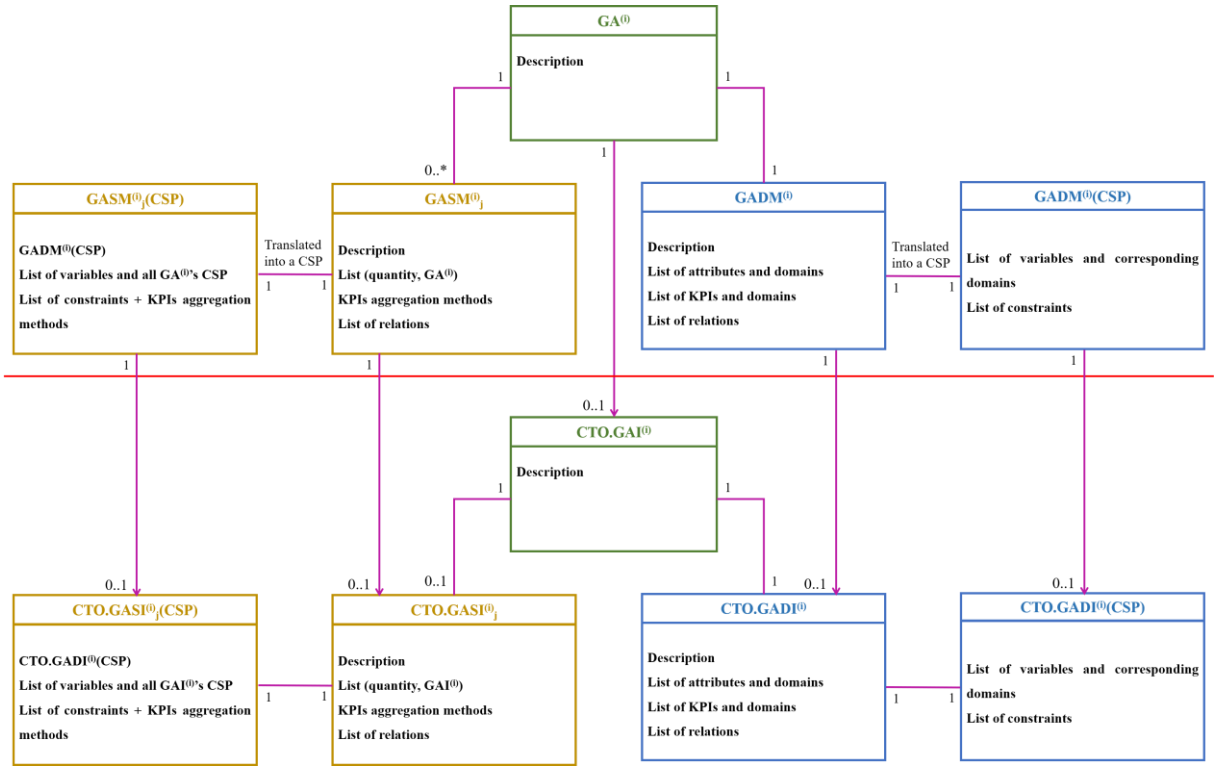

Figure 40. UML diagram for *CTO.GAI*$^{(i)}$, *CTO.GADI*$^{(i)}$ and *CTO.GASI*$^{(i)}_{j}$ and their translation into their corresponding CSP

The aim of CTO configuration is to reach a CTO solution which meet all the CTO requirements. A CTO requirement and a CTO solution are defined as follows.

Definition 18: CTO requirement
A CTO requirement corresponds to the selection of a value from the domain of an attribute. We denoted it as *CTO.rq*.

Definition 19: CTO solution
A CTO solution for a *CTO.GAI*$^{(i)}$ corresponds to a set of attributes where the domains have been restricted to singletons during the CTO configuration activity by respecting the defined constraints. For the KPIs, ranges of values for a solution are accepted as they can correspond to some lack of knowledge, uncertainties or several possibilities. During the CTO configuration, if the choice to use the structural view is made, the CTO solution integrates the CTO instance *GASI*$^{(i)}_{j}$. Then, the bill of material is also part of the solution. We denoted it as *CTO.Solution*.

4.1.1.3. Generic model instance processing

In order to configure systems in CTO situations by exploiting the instances of the selected generic models, a generic model processing activity has to be performed. The generic process for knowledge reuse in CTO is proposed in Figure 41 and Figure 42. It is a recursive process based on a sub-process denoted "Configure CTO GAI" composed of two parts: one dedicated to the descriptive view, denoted "Configure CTO GADI" (Figure 41) and one optional, dedicated to the structural view, denoted "Configure CTO GASI" (Figure 42).

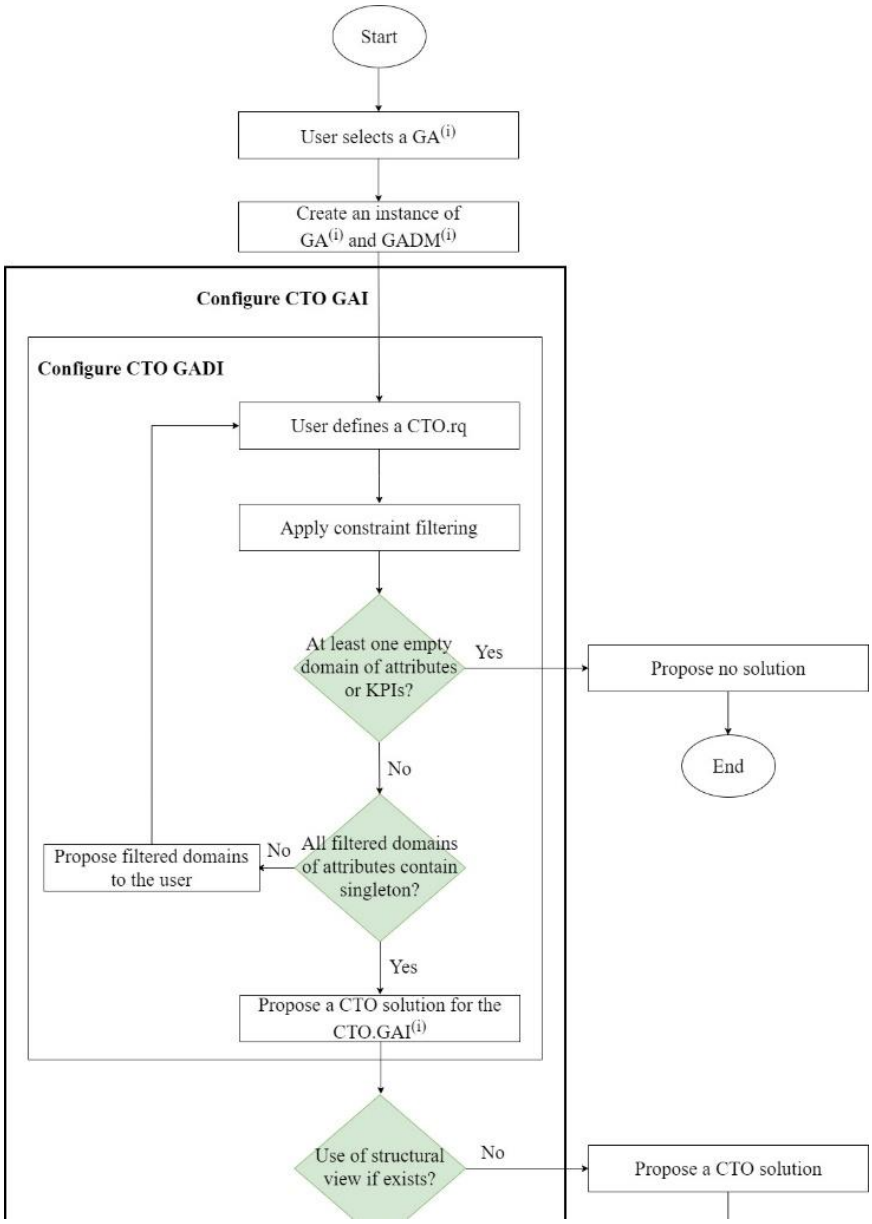

Figure 41. Flowchart for knowledge reuse in CTO configuration: GADI configuration

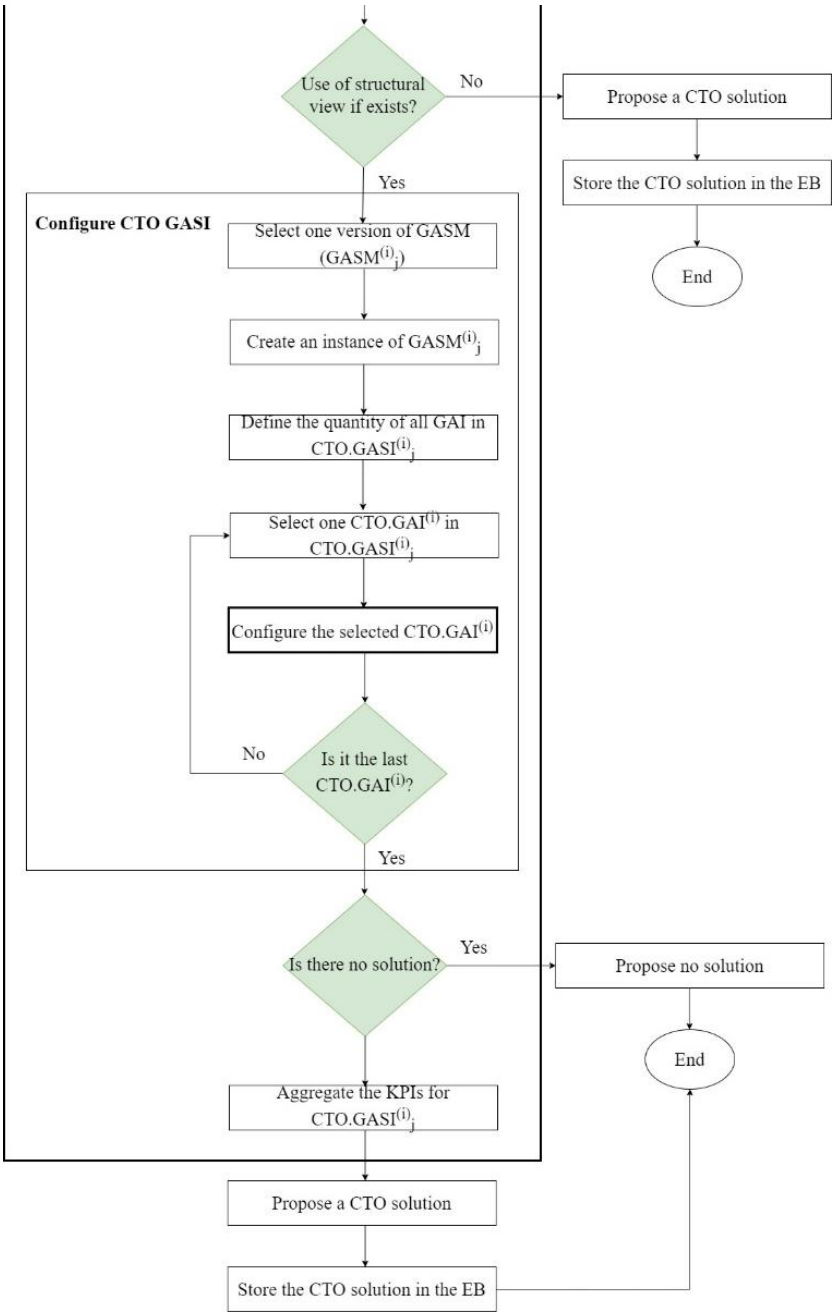

Figure 42. Flowchart for knowledge reuse in CTO configuration: GASI configuration

In this CTO knowledge reuse process, a $GA^{(i)}$ is first selected by the user. The instance $CTO.GAI^{(i)}$ is created with the instance $CTO.GADI^{(i)}$. Then, the $CTO.GAI^{(i)}$ and the $CTO.GADI^{(i)}$ are provided to the "Configure CTO GAI" sub-process. Using the $CTO.GADI^{(i)}$, the user launches the "Configure CTO GADI" sub-process and defines one requirement *CTO.rq* restricting the domain of a variable in the CSP. A constraint filtering mechanism is applied using the corresponding CSP (i.e. the $CTO.GADI^{(i)}(CSP)$). If one empty domain is obtained, the CTO knowledge reuse process ends with no solution. Otherwise, the definition of requirements and the filtering are continued until all filtered domains for attributes contain singletons (i.e. a single value). Then, a CTO solution for the $CTO.GAI^{(i)}$ is proposed to the user. It corresponds to a set of couples (attribute, value) or (variable, value). The KPIs can either have an interval or a single value in their domains.

Then, the user is asked to express if the structural view is necessary or not. If it is not necessary, the configuration is over and the solution is stored in the Experience Base (EB) and the user can exploit the obtained CTO solution. If the structural view is necessary, the user launches the "Configure CTO GASI" sub-process and has to select one version of the $GASM^{(i)}_j$ thanks to its description. Then, an instance $CTO.GASI^{(i)}_j$ is created. Therefore, for each $CTO.GAI^{(i)}$ which belongs to $CTO.GASI^{(i)}_j$, its quantity has to be defined and the sub-process "Configure GAI" is reused recursively. The CTO knowledge reuse process ends when all the $CTO.GAI^{(i)}$ have been configured and the $CTO.GASI^{(i)}_j$ KPI are aggregated. Then, the instantiated $CTO.GASI^{(i)}_j$ is added to the CTO solution.

At the end of the CTO knowledge reused process, the solution (which can be only descriptive or which can include a bill-of-material - i.e. structural) is stored into the EB. This will enable later reuse of the system configuration experience in similar situations. However, this is not described in this thesis.

In the following section, we explain what are the consequences of our proposals mentioned in chapter 3 in CTO knowledge reuse for system configuration.

4.1.2. Consequences of modeling propositions

This section is dedicated to the consequences and impact of our proposals on knowledge reuse in CTO situations. Therefore, first, descriptive view versus structural view and then multi-level abstraction modeling are explained.

- **Descriptive view versus structural view**

As mentioned in chapter 3, the descriptive view of a $GA^{(i)}$ is defined by a Generic Artifact Descriptive Model ($GADM^{(i)}$), while its structural view is defined by a Generic Artifact Structural Model ($GASM^{(i)}_j$).

When a user selects a $GA^{(i)}$, its associated $GADM^{(i)}$ is selected as well. It implies that the choice of $GADM^{(i)}$ is predetermined since each $GA^{(i)}$ is linked to only one $GADM^{(i)}$. On the other hand, the selected $GA^{(i)}$ can have multiple $GASM^{(i)}$ associated with it or none at all.

Once the user selects a $GA^{(i)}$ and builds an instance of $GADM^{(i)}$, the CTO configuration of $CTO.GADI^{(i)}$ begins. During the CTO configuration of the $CTO.GADI^{(i)}$, the user decides whether to configure GASM as well. If the user wants to configure GASM, first, she/he selects one version of GASM from the available GASMs. To select one GASM, the user can compare different GASMs based on their KPIs.

On the other hand, since a $GASM^{(i)}_j$ is defined recursively, it is composed of several $GA^{(i)}$. Therefore, for each $GA^{(i)}$, the user needs to decide whether to configure only $GADM^{(i)}$ or configure both $GADM^{(i)}$ and $GASM^{(i)}_j$. It's important to note that for each $GA^{(i)}$, the configuration of $GADM^{(i)}$ is mandatory however, a decision on which $GASM^{(i)}_j$ be configured and how deeply depends entirely on the model and user's decision.

- **Multi-level abstraction modeling**

In chapter 3, we have explained that at the end of the knowledge formalization, three taxonomies of GA, GADM, and GASM are created. Each of them can be defined at different levels of abstraction.

From the GAs taxonomy, first, the user can select either a general or a specialized GA. In the case that the user selects a GA at a higher level of abstraction ($GA^{(i-1)}$), both the associated GADM and GASMs are also generic and at higher level of abstraction (respectively $GADM^{(i-1)}$ and $GASM^{(i-1)}_j$). It is important to notice that the user selects a $GA^{(i)}$ when none of the specialized GAs ($GA^{(i)}$) can fulfill requirements this choice can be made by looking at $GA^{(i)}$'s name and description. Once the user has selected $GA^{(i-1)}$, an instance of its associated GADM (notated $CTO.GADI^{(i-1)}$) is configured and an instance of its associated GASM (notated $CTO.GASI^{(i-1)}_j$) can be configured as well. During configuration, the user can access the common body of knowledge to meet their requirements.

On the other hand, when a user selects a specialized GA at a lower level of abstraction ($GA^{(i+1)}$), both the associated GADM and GASMs are also specialized in nature and at a lower level of abstraction (respectively $GADM^{(i+1)}$ and $GASM^{(i+1)}_j$). Once the user has selected $GA^{(i+1)}$, an instance of its associated GADM (notated $CTO.GADI^{(i+1)}$) is configured and an instance of its associated GASM (notated $CTO.GASI^{(i+1)}_j$) can also be configured. During configuration, the user can access a more restricted but enriched body of knowledge to fulfill their particular requirements.

In the following, the synthesis of the chapter is presented.

4.1.3. Synthesis

To address our second research question, it is required to first study CTO situations. Therefore, this section was dedicated to the CTO knowledge reuse. First, we briefly explained about generic model selection. Then, we discussed CTO configuration activity. We began by defining the CTO principles and the generic model instances used for CTO configuration. Generic Artifact Instance ($CTO.GAI^{(i)}$), Generic Artifact Descriptive Instance ($CTO.GADI^{(i)}$) and Generic Artifact Structural Instance definition ($CTO.GASI^{(i)}_j$) were defined. Then, a generic CTO knowledge reuse process has been detailed. It enables to provide CTO solutions to meet CTO requirements.

Second, we explained the impact of our proposals related to knowledge formalization in CTO situations including selecting and configuring descriptive and/or structural views and selecting and configuring according to different levels of abstraction.

After describing the knowledge reuse in CTO situations, we need to move towards knowledge reuse in ETO situations which is the subject of the following section.

4.2. ETO knowledge reuse for system configuration

This section is devoted to knowledge reuse in ETO situations. Therefore, in section 4.2.1, the transition from CTO to ETO configuration activity is studied. Then, in section 4.2.2, the adaptation of CTO instances to ETO is discussed. Finally, in section 4.2.3, the section is synthesized.

4.2.1. From CTO towards ETO configuration activity

In CTO configuration activity, it is not possible to fulfill all types of requirements, particularly ETO requirements which are out of the generic models (as mentioned in chapter 2, sections 2.1.2.1 and 2.1.2.2). To address them, engineering activities are required. In our proposals, this is taken into account by a process which enables modifying existing pieces of knowledge or creating new ones. Therefore, ETO requirements must be fulfilled during configuration activity in ETO situation. In this thesis, we will refer to configuration activity in ETO situation as "ETO configuration".

The definition of an ETO requirements is given below.

Definition 20: ETO requirement
An ETO requirement corresponds to a need that cannot be expressed with the current instance. It may involve selecting a value outside the possible domain of an attribute, adding elements to the nomenclature, or choosing combinations not authorized by the underlying model. We denoted it as *ETO.rq*.

In order to perform an ETO configuration activity, we formalize a generic process for knowledge reuse in ETO situations by means of the flowchart of Figure 43 and Figure 44. Some parts of this process are similar to the generic processes of Figure 41 and Figure 42 for CTO configuration. It enables to reuse formalized pieces of knowledge allowing to modify them in order to meet ETO requirements. This process starts by asking to the user to check if a relevant $GA^{(i)}$ corresponding to the requirements is available. If it is the case, this $GA^{(i)}$ is selected in the taxonomy of GAs with it associated $GADM^{(i)}$. Otherwise, a new $ETO.GAI^{(i)}$ is created with its $ETO.GADI^{(i)}$. This corresponds to the realization of an engineering activity where a new system or family of systems has to be designed. Following the necessity or not to have a structural view for this new E$TO.GAI^{(i)}$, an $ETO.GASI^{(i)}$ is created or not. It is important to notice that only instances are created and not generic models such as GA, GADM or GASM. Indeed, these new pieces of knowledge are created to answer to ETO requirements but they are not yet validated as standard knowledge. Therefore, every new element created or modified during this ETO configuration process is embedded in an instance and it cannot be considered as standard knowledge. If every GAI, GADI and GASI are created -we can talk about a heavy ETO solution- no modification is required so this ETO solution can be proposed then capitalized in an EB. Note that the definitions of these instances are given in the next section.

Then, the sub-process denoted “Configure ETO GAI” is performed. It is a recursive process which enables to configure a GAI to meet the ETO requirements. This sub-process starts by the definition by the user of one ETO requirement denoted *ETO.rq*. With regard to this requirement,

the modification of the GADI is performed. Four cases are taken into account and this step will be described in the section 4.2.2.2. We denoted it as $ETO.GAI^{(i)}$ to distinguish it from the $CTO.GAI^{(i)}$.

Therefore, the $ETO.GADI^{(i)}$ is manually configured. That means that a restriction of the domains following the *ETO.rq* is done. This cannot be done automatically using a filtering mechanism because all the new added elements will be removed as they are not considered as consistent with the knowledge. The definition of CTO/ETO requirements by the user and the $ETO.GADI^{(i)}$ modification is continued until all the domains contains single values. Then, an ETO solution is obtained for the $GAI^{(i)}$. It corresponds to a set of couples (attribute, value) or (variable, value). The KPIs can have a range in their domains. The ETO solution is stored into the EB

If the structural view is required for ETO configuration, the user has to select the right version of the GASM and create an instance denoted $ETO.GASI^{(i)}_j$ (or to use the GASI which has been created at the beginning of the process).

If it is necessary to meet ETO requirements, the modification of the $ETO.GASI^{(i)}_j$ is performed following five complementary cases (see section 4.2.2.2). This step enables to obtain a modified $ETO.GASI^{(i)}_j$. Then, for each $GAI^{(i)}$ which belongs to the $ETO.GASI^{(i)}_j$, the sub-process "Configure ETO GAI" is performed recursively. At the end of the modification of the $ETO.GASI^{(i)}_j$, it is added to an ETO solution.

At the end of the ETO knowledge reused process, the ETO solution is stored into the EB. This will enable later reuse of the system configuration experience in similar situations, as well as the update of the GAs, GADMs, and GASMs stored in the GMB. However, this is not described in this thesis.

The definition of an ETO solution are given below.

Definition 21: ETO solution
An ETO solution for a $GA^{(i)}$ corresponds to a set of attributes whose domains are restricted to singletons during the configuration activity, free from the constraints defined by the generic model. If a dive into the structural view has been performed, the ETO solution has a nomenclature (which may or may not be consistent with the knowledge model). We denoted it as *ETO.solution*.

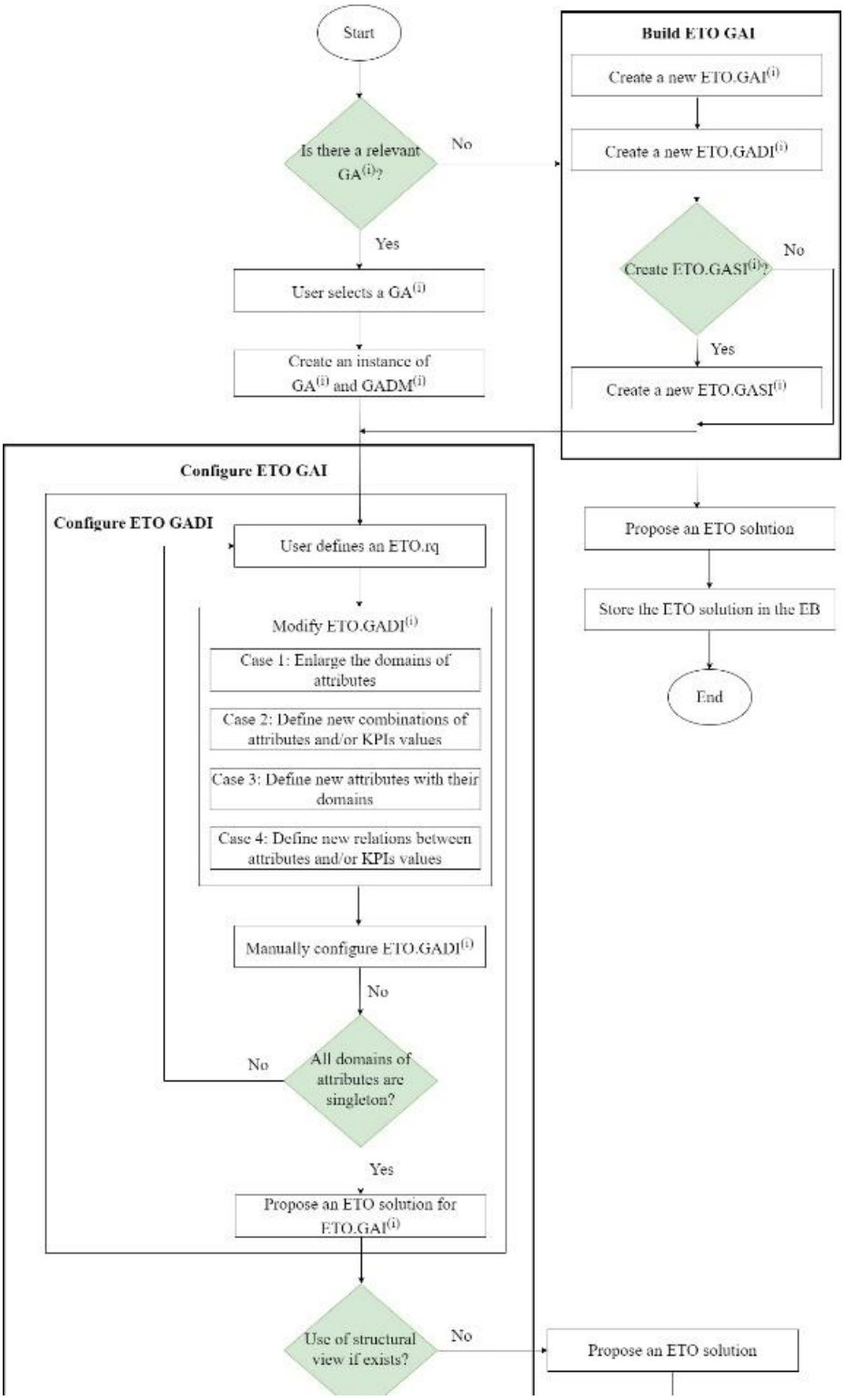

Figure 43. Flowchart for knowledge reuse in ETO situations: GADI configuration

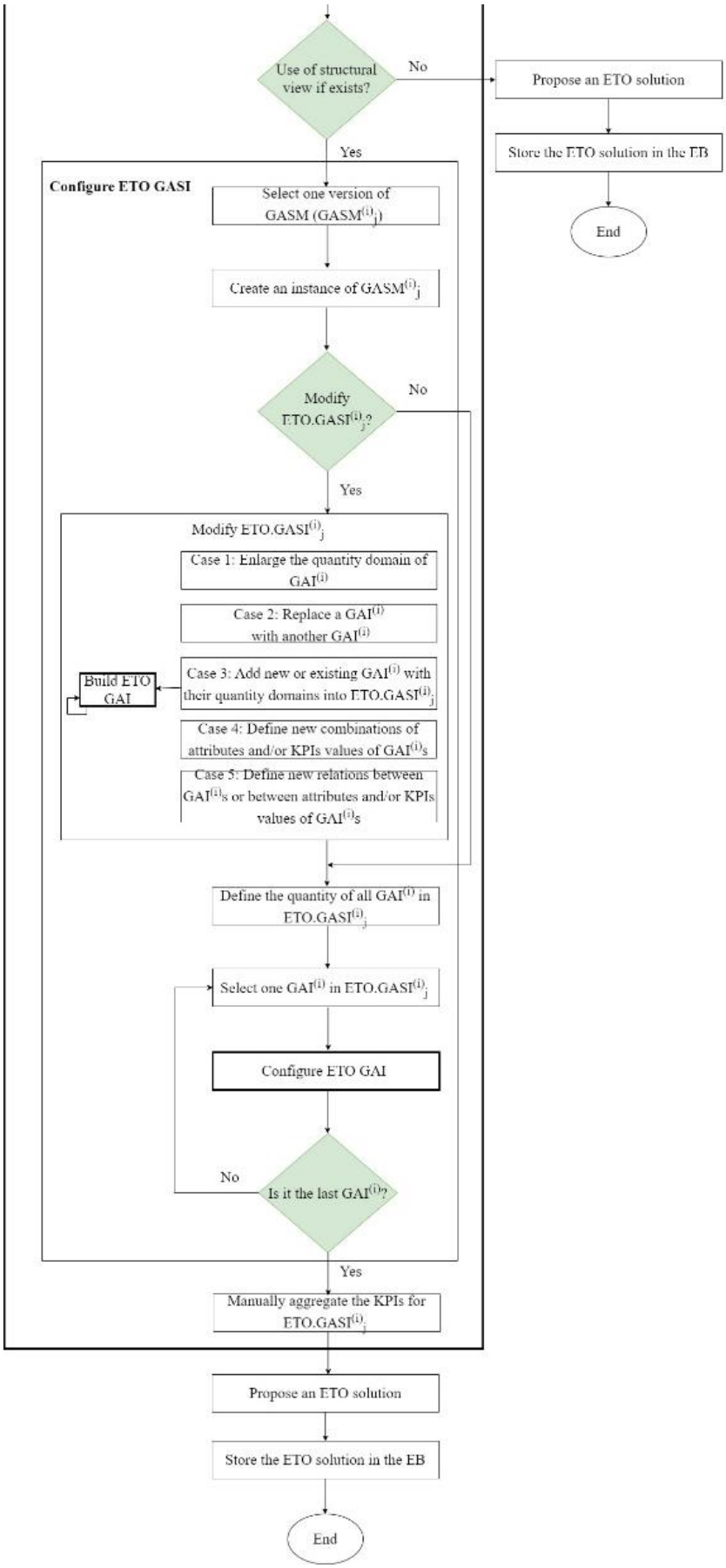

Figure 44. Flowchart for knowledge reuse in ETO situations: GASI configuration

In the following section, we will discuss how to adapt from CTO instances to ETO configuration.

4.2.2. Adaptation of CTO instances to ETO configuration

As mentioned before, in ETO configuration activity, to meet ETO requirements it may be required to modify the attributes, structure, or relations of the existing instances. Or even it may be needed to define entirely new attributes or relations for instance. This means that instances in CTO need to be adapted in order to meet the user's ETO requirements. Therefore, in this section, first, the definition of instances in ETO is represented. Then, in section 4.2.2.1, building new instances is discussed. After that in section 4.2.2.2 modifications of existing instances are proposed.

Definition 22: Generic Artifact Instance or $GAI^{(i)}$ in ETO
A Generic Artifact Instance, notated $GAI^{(i)}$ is a completely new instance or a modified one which is defined during ETO configuration activity in order to answer to ETO requirements of users. We denoted it as $ETO.GAI^{(i)}$ to distinguish it from the $CTO.GAI^{(i)}$.

Since the selected $GA^{(i)}$ is linked to a $GADM^{(i)}$, it is necessary to build an instance of the $GADM^{(i)}$.

Definition 23: Generic Artifact Descriptive Instance or $GADI^{(i)}$ in ETO
A Generic Artifact Descriptive Instance, notated $GADI^{(i)}$ is either a completely new instance or a modified one. It is defined during ETO configuration activity to answer to ETO requirements of users. It enables to use a descriptive view of the $ETO.GAI^{(i)}$ during ETO configuration. We denoted it as $ETO.GADI^{(i)}$.

When the choice to use the structural view is made, among several GASMs that the selected $GA^{(i)}$ can be associated with, the user has to choose one. Therefore, it is necessary to build an instance of the selected GASM.

Definition 24: Generic Artifact Structural Instance or $GASI^{(i)}_j$ in ETO
An instance of a GASM is denoted GASI, which can be either a completely new instance or a modified one. It is an element that enable to define a structural view for an $ETO.GAI^{(i)}$ during the ETO configuration activity to meet ETO requirements. We denoted it as $ETO.GASI^{(i)}_j$.

4.2.2.1. Build ETO GADI or ETO GASI

This section is dedicated to the definition of new instances based on ETO requirements of the user. Initially, the definition of GAI build is proposed, followed by the definition of GADI build. Finally, the definition of GASI build is provided.

During the ETO configuration activity, the user may require a $GA^{(i)}$ that does not exist in the taxonomy of GAs. We, therefore, build a new instance of $GA^{(i)}$ as follows.

Definition 25: GAI building in ETO
GAI building is a process of creating a new $GAI^{(i)}$ in ETO configuration activity to satisfy ETO requirements of users that cannot be met by reusing any of the $GA^{(i)}$ in GMB. We denoted it as $ETO.GAI^{(i)}$.

As for knowledge formalization to create a $ETO.GAI^{(i)}$, it is necessary to assign a name and provide a description. For instance, in Figure 45, a new $ETO.GAI^{(i)}$ representing an instance of the family of mirrors is defined. This new $ETO.GAI^{(i)}$ is built since the user demands $Mirror.GA^{(i)}$ however, this $GA^{(i)}$ does not exist in the taxonomy of GAs. In $ETO.Mirror.GAI^{(i)}$, the name Mirror conveys a vivid mental image of a mirror. Its description is provided as well.

Although a $ETO.GAI^{(i)}$ does not explicitly represent knowledge about the attributes or structure of an instance of artifacts family, it offers a clear comprehension of the instance. Note that once the new $ETO.GAI^{(i)}$ is built, it will not be stored within the GAs taxonomy (in the GMB), but it will be stored in the experience base.

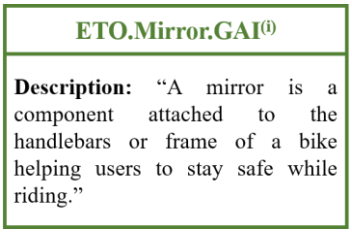

Figure 45. An example of $ETO.Mirror.GAI^{(i)}$

When a $ETO.GAI^{(i)}$ is created, it is necessary to create its associated $ETO.GADI^{(i)}$ in order to obtain a descriptive view of the $ETO.GAI^{(i)}$. The descriptive view or $GADI^{(i)}$ will enable to perform the ETO configuration activity by using the descriptive view.

Definition 26: GADI building in ETO
We define GADI building as a process of creating a new $GADI^{(i)}$ in ETO configuration activity to satisfy ETO requirements of users that cannot be met by reusing the formalized knowledge. It corresponds to the descriptive view of its associated $ETO.GAI^{(i)}$. We denoted it as $ETO.GADI^{(i)}$.

Similar to a $GADM^{(i)}$, $ETO.GADI^{(i)}$ is characterized by:

- **Name:** It shares the same name as the $ETO.GAI^{(i)}$, providing a clear mental representation of the $ETO.GADI^{(i)}$.
- **Description:** It offers a statement that portrays the current appearance of the $ETO.GADI^{(i)}$. This description can be the same as the $ETO.GAI^{(i)}$'s description or it can be more detailed.
- **Attributes with their domains:** It includes a list of key descriptive attributes and their valid domains, which depict the characteristics of the current $ETO.GAI^{(i)}$. The domain can include the uncertainty of each attribute.
- **KPIs with their domains:** It consists of a list of key performance indicators and their valid domains, which are necessary for evaluating the performance of the current $ETO.GAI^{(i)}$. The corresponding domains of KPIs also take into account the uncertainty associated with them. It means that the domain of KPIs can be a range or a singleton.
- **Relations between attributes and/or KPIs values (not mandatory):** It involves a list of relations that describe the solution space of the current $ETO.GAI^{(i)}$. These relations determine which combinations of attribute and/or KPI values are allowed or forbidden, thereby defining the possible characteristics of the $ETO.GAI^{(i)}$ along with its associated performances. This can be helpful to update the generic models stored in the GMB.

To formalize an $ETO.GADI^{(i)}$, we propose mapping it within a CSP:

- Attributes and domains correspond to variables and domains of the CSP,
- KPIs and domains are also treated as variables with their corresponding domains,
- Relations between attributes and/or KPIs of $ETO.GADI^{(i)}$ are expressed by employing compatibility tables and numerical functions.

By formalizing the $ETO.GADI^{(i)}$ as a CSP, we can apply filtering techniques to check and maintain the consistency of the $ETO.GADI^{(i)}$.

Figure 46 represents a new $ETO.GAI^{(i)}$, and its associated $ETO.GADI^{(i)}$ which are mapped into a CSP notated $ETO.GADI^{(i)}(CSP)$.

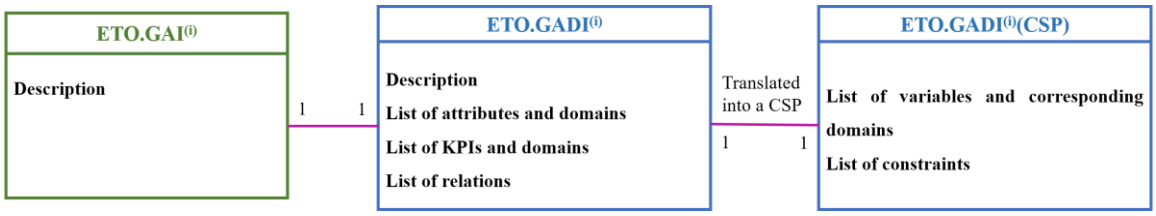

Figure 46. Build a new $ETO.GADI^{(i)}$ and its translation into a CSP

For example, Figure 47 illustrates an *ETO.GADI*$^{(i)}$ of an instance of mirror family which is built since the user wants *Mirror.GADM*$^{(i)}$ but this *GADM*$^{(i)}$ is not present in the taxonomy of GADMs. Therefore, this new *ETO.GADI*$^{(i)}$ is built and then translated into a CSP. In *ETO.Mirror.GADI*$^{(i)}$, there is one attribute (Diameter) and two KPIs (Weight and Cost), which correspond to three variables in *Mirror.GADI*$^{(i)}$*(CSP)*. It should be noted that *ETO.Mirror.GAI*$^{(i)}$ and *ETO.Mirror.GADI*$^{(i)}$ are built to meet ETO requirements. This is why the term "ETO" is used in *ETO.Mirror.GAI*$^{(i)}$, *ETO.Mirror.GADI*$^{(i)}$ and its CSP.

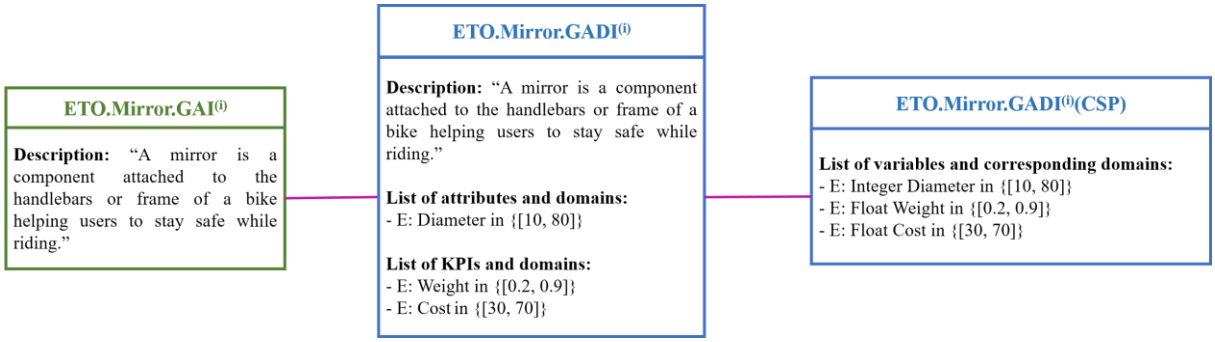

Figure 47. An example of building a *ETO.Mirror.GADI*$^{(i)}$ and its translation into a CSP

During the ETO configuration activity, the user may require to use a structural view for an *ETO.GAI*$^{(i)}$.

Definition 27: GASI building in ETO
We define GASI building as a process of creating a new GASI in ETO configuration activity to satisfy ETO requirements of users that cannot be met by reusing the formalized knowledge in GMB. We denoted it as *ETO.GASI*$^{(i)}$.

Therefore, an instance *ETO.GASI*$^{(i)}$ has to be created. It is important to notice that it is not an instance of an existing *GASM*$^{(i)}_j$ but it is built specifically during the ETO configuration activity in order to meet the ETO requirements of the user.

As for a *GASM*$^{(i)}_j$, an *ETO.GASI*$^{(i)}$ is characterized by:

- **Name:** It refers to the same name as the *ETO.GAI*$^{(i)}$ and aims to create a vivid mental image of the *ETO.GASI*$^{(i)}$.
- **Description:** It provides a statement that describes the current appearance or composition of the *ETO.GASI*$^{(i)}$.
- **GASI composition (quantity, GAI$^{(i)}$):** It is a list of pairs consisting of the quantity and *GAI*$^{(i)}$ (existing *GAI*$^{(i)}$ and new ones) that make up the first level of GASI composition.
- **KPIs aggregation methods:** It comprises a list of methods used to evaluate the KPIs of the associated *ETO.GADI*$^{(i)}$ based on its *GAI*$^{(i)}$s and their quantities.

- **Relations between GAI$^{(i)}$s or between attributes and/or KPIs of GAI$^{(i)}$s (not mandatory):** It represents a list of relations within the *ETO.GASI$^{(i)}$* that define the entire solution space of the current *ETO.GASI$^{(i)}$*. These relations enable updating the generic models stored in the GMB.

In order to formalize an *ETO.GASI$^{(i)}$*, our proposal involves mapping *ETO.GASI$^{(i)}$* within a CSP in the following manner:

- Each *GAI$^{(i)}$* in the GASI composition is substituted by its corresponding CSP, while the quantities are represented by variables and domains,
- KPIs aggregation methods are formalized as constraints that establish links between all KPIs associated with the composition of the current *ETO.GASI$^{(i)}$*,
- Relations are formalized by employing compatibility tables and numerical functions. As explained in section 3.1.3 the relations within *ETO.GASI$^{(i)}$* can have different types.

Consequently, the corresponding CSP consists of two parts: one for the *ETO.GADI$^{(i)}$* and another for the *ETO.GASI$^{(i)}$*.

In order to build a *ETO.GASI$^{(i)}$*, firstly, a name (the same as the *ETO.GAI$^{(i)}$*'s name) and a description must be defined to the *ETO.GASI$^{(i)}$*. Then, the *GAI$^{(i)}$*s that belong to the *ETO.GASI$^{(i)}$*, along with their quantities have to be defined. Furthermore, aggregation methods for each KPI of *ETO.GASI$^{(i)}$* must be defined. Additionally, relations between attributes and/or KPI values of the *GAI$^{(i)}$*s, must be established.

CSP allows to apply filtering techniques on every *GAI$^{(i)}$* locally. In this way, it ensures the consistency of the newly built *ETO.GASI$^{(i)}$*. However, this *ETO.GASI$^{(i)}$* will not be stored in the GMB, but it will be stored into the experience base for reusing.

Similar to GASM, the process of defining *ETO.GASI$^{(i)}$* is also iterative, recursive and bottom-up. Meaning that a *GAI$^{(i)}$* within a *ETO.GASI$^{(i)}$* can have its own *GASI$^{(i)}_{j}$*, and a new *ETO.GASI$^{(i)}$* cannot be defined without first building a new *ETO.GAI$^{(i)}$* and a new *ETO.GADI$^{(i)}$*.

Figure 48 illustrates that one new *ETO.GASI$^{(i)}$* is built for a new *ETO.GAI$^{(i)}$*, which is then mapped into a CSP.

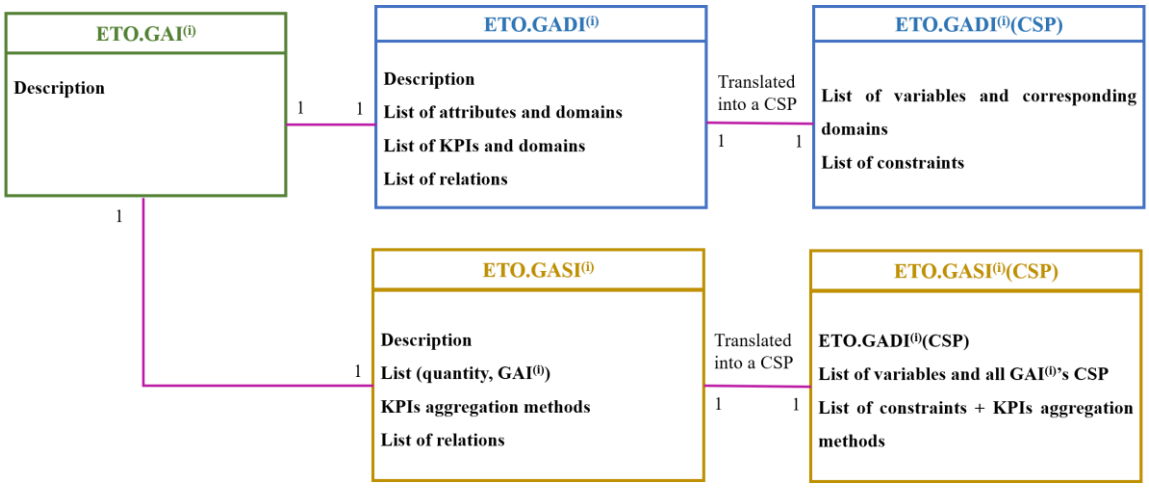

Figure 48. *ETO.GASI*$^{(i)}$ and its translation into a CSP

For instance, Figure 49 shows *ETO.Mirror.GASI*$^{(i)}$, as an instance representing a structural view for a family of mirrors. Based on the user requirements, *ETO.Mirror.GASI*$^{(i)}$ is composed of one *ETO.Glass.GAI*$^{(i)}$ and one *ETO.MirrorFrame.GAI*$^{(i)}$. The CSP of *ETO.Mirror.GASI*$^{(i)}$ encompasses the CSP of the *ETO.Mirror.GADI*$^{(i)}$. In *Mirror.GASI(CSP)*, *GAI*$^{(i)}$ within *ETO.Mirror.GASI*$^{(i)}$ are replaced with their CSP and quantities of both *GAI*$^{(i)}$ (*ETO.MirrorFrame.GAI*$^{(i)}$ and *ETO.Glass.GAI*$^{(i)}$) are represented by variables. KPIs aggregation methods are also formalized as constraints (represented as numerical functions). A relation in *ETO.Mirror.GASI*$^{(i)}$ exists: different materials of the mirror frame are only appropriate for the certain shape of the glass. This relation is formalized using a table of compatibility in the CSP of the *ETO.Mirror.GASI*$^{(i)}$ which links the compatible values of the variables 'Shape' (from *ETO.Glass.GADI*$^{(i)}$) and 'Material' (from *ETO.MirrorFrame.GADI*$^{(i)}$). Finally, constraint filtering is applied to ensure the consistency of the new *ETO.Mirror.GASI*$^{(i)}$.

Note that since *ETO.Mirror.GASI*$^{(i)}$ is built to meet ETO requirements, therefore in both *ETO.Mirror.GASI*$^{(i)}$ and its CSP the letter 'E' is used for the list (quantity, *GAI*$^{(i)}$), KPIs aggregation methods, relations, variables, CSPs, and constraints. For a compatibility table, the tag 'E' is written in the columns and tuples. However, for numerical functions, the tag 'E' is written at the beginning of the constraint.

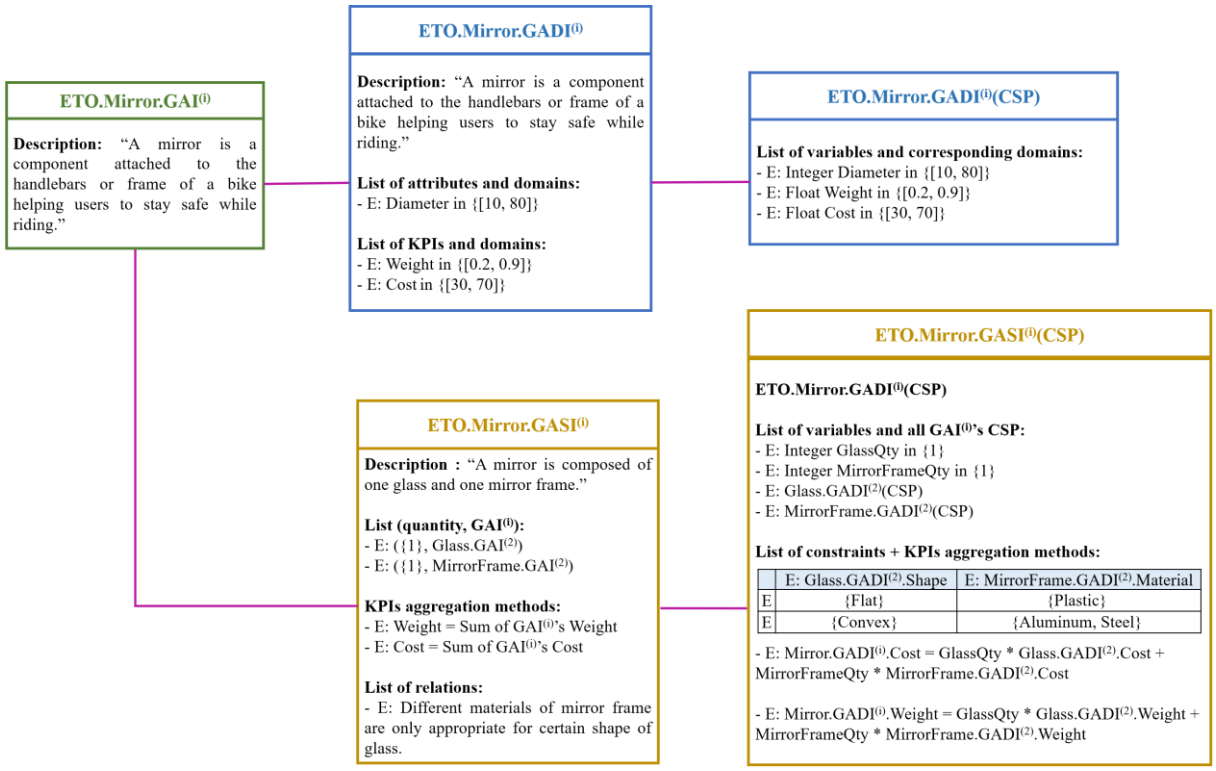

Figure 49. Example of a new *ETO.Mirror.GASI*$^{(i)}$ and its translation into a CSP

The following section is dedicated to the modification of instances.

4.2.2.2. Modify GADI or GASI

During ETO configuration, users may require attributes, structure, or relations that does not exist in the formalized knowledge models for configuration. Therefore, in order to fulfill these ETO requirements, existing instances must be modified leading to modifications in the solution space. Therefore, in this section which is devoted to the modification of instances, first, the definitions of GADI and GASI modification are proposed.

Definition 28: GADI modification
GADI modification is the process of modifying or adapting the current GADI during ETO configuration by enriching the available knowledge to meet ETO requirements. This may lead to an ETO solution for the *GAI*$^{(i)}$ that will be stored in the EB. This modification turns the *CTO.GADI*$^{(i)}$ into an *ETO.GADI*$^{(i)}$.

This GADI modification is a two-step process:

Step 1) Modify the name and description of the current *ETO.GADI*$^{(i)}$ (if necessary),

Step 2) Enrich the knowledge of the current *ETO.GADI*$^{(i)}$: based on the work of (Sylla, Guillon, Vareilles, et al., 2018) four cases are proposed. Therefore, the user can enrich the knowledge of the current *ETO.GADI*$^{(i)}$ by adding specific knowledge in order to meet ETO requirements. To enrich the knowledge, the following four cases for *ETO.GADI*$^{(i)}$ modifications are explained as follows:

- Case 1: Enlarge the domains of existing attributes: since the user requires one value of an attribute which is outside the current domains of attribute, this new value must be added to the domains of attribute. It enables to define an $ETO.GADI^{(i)}$ with new characteristics according to what the user wants.

- Case 2: Define new combinations of attributes and/or KPIs values: the user wants a new combination of attributes and/or KPIs values that is beyond the existing ones. Therefore, a relation must be extended in order to link at least two incompatible values of attributes and/or KPIs within an existing constraint.

- Case 3: Define new attributes with their domains: since the user requires a new attribute that does not previously exist, a new attribute with its domain must be defined. This allows the user to define an $ETO.GADI^{(i)}$ with specific characteristics (i.e. attributes with their domains).

- Case 4: Define new relations between attributes and/or KPIs values: the user may need for a relation that does not currently exist, aiming to establish a link between attributes and/or KPIs values. Alternatively, the user may wish to explicitly represent the allowed or forbidden combinations of attributes and/or KPIs values. Therefore, the user must add a new relation that can link new attributes with existing attributes, or new attributes. This is the only case where the solution space is not enlarged.

We propose to modify the CSP ($ETO.GADI^{(i)}(CSP)$) as follows:

- Case1: Enlarging the domains of attributes or KPIs corresponds to adding new values to the corresponding domains of variables. In this case, constraint filtering must not be applied since there may exist a constraint that after filtering, will delete the added values.

- Case 2: Defining new combinations of attributes and/or KPIs values corresponds to adding new tuples to the existing compatibility tables. The user extends the constraint by adding a new tuple that links at least two previous incompatible values of variables. It should be noted that this case mainly deals with compatibility tables. Therefore, if there is a numerical function, the user needs to modify it manually.

- Case 3: Defining new attributes with their domains corresponds to adding new variables and their corresponding domains.

- Case 4: Defining new relations corresponds to adding new constraints in $ETO.GADI^{(i)}(CSP)$ either compatibility tables or numerical functions).

For instance, as represented in Figure 50, in order to meet ETO requirements, $ETO.Wheel.GADI^{(2)}$ is modified. The requirement of the user can be related to a better resistance to shocks during off-road utilization of wheel. As no solution exists to meet this requirement, the $ETO.Wheel.GADI^{(2)}$ must be modified. This modification will correspond to the result of an engineering activity performed by designers. Then, it is translated into a CSP ($ETO.Wheel.GADI^{(2)}(CSP)$). The user wants the diameter of the wheel to be 29, therefore in

ETO.Wheel.GADI$^{(2)}$, the domain of the attribute Diameter is enlarged and the value 'E:29' is added into the domain. In the CSP of *ETO.Wheel.GADI*$^{(2)}$, the new value 'E:29' is added to the domain of the variable Diameter. Note that before 29, E is written to represent that this value is added to fulfill the ETO requirement. That corresponds to the case 1.

The user wants a wheel with 28 spokes. To define this requirement in *ETO.Wheel.GADI*$^{(2)}$, a new attribute with its domain is added (E: SpokeQty in {28}). In the CSP of *ETO.Wheel.GADI*$^{(2)}$, a new variable with its corresponding domain is added (E: SpokeQty). That corresponds to the case 3.

The user wants to express the fact that a wheel with the diameter 29, a material steel, a weight of 3, and a cost of 600 € are now compatible. Therefore, to meet this requirement, an existing relation is extended in *ETO.Wheel.GADI*$^{(2)}$, while in the CSP of *ETO.Wheel.GADI*$^{(2)}$, an existing constraint is extended by adding a new tuple (tagged using E), indicating that Diameter '29' is compatible with the material 'Steel', the weight '3', and the cost '600'. That corresponds to the case 2.

The user needs to define a relation between the quantity of spokes and wheel diameter. Therefore, in *ETO.Wheel.GADI*$^{(2)}$, a new relation indicating that the quantity of spoke depends on the diameter is defined. Then, in *ETO.Wheel.GADI*$^{(2)}$*(CSP)*, a new compatibility table is added linking the compatible values of variables I: Diameter and E: SpokeQty. Note that constraint filtering is not applied thus the domains of variables are not restricted. That corresponds to the case 4.

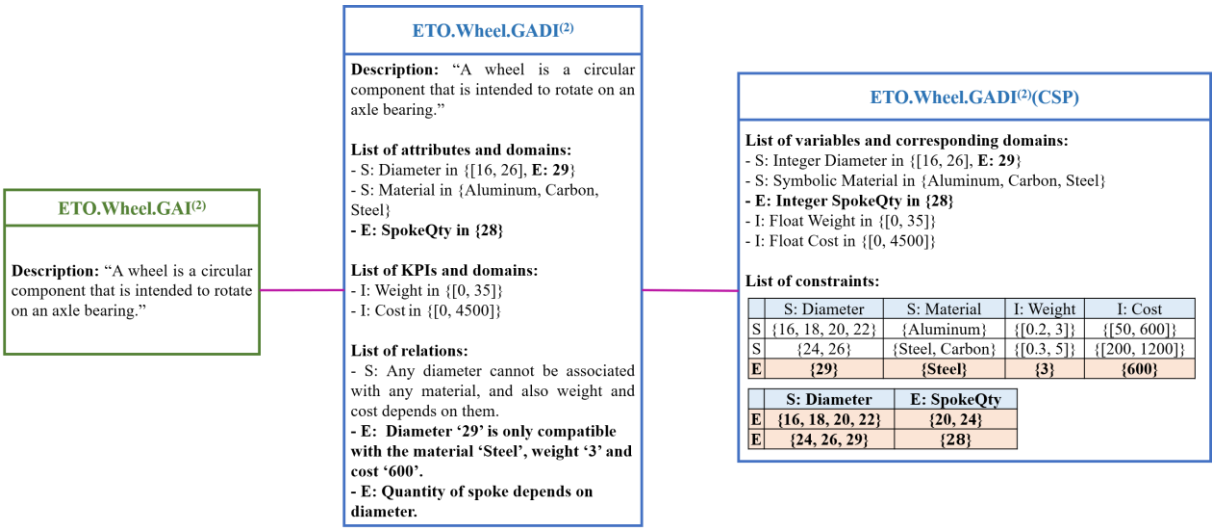

Figure 50. Example of modifying *ETO.Wheel.GADI*$^{(2)}$ and its translation into a CSP

In this example, only the descriptive view is necessary. The user doesn't need to take into account how the *Wheel.GAI* is composed. Therefore, the ETO configuration consists in modifying the instance *Wheel.GADI* of *Wheel.GASM* and to manually configure it. For instance, if the user wants that the diameter of the wheel is exactly '29', the configuration is done manually removing all the incompatible values from the variables domains. In the example of Figure 50, this will lead to a solution where the diameter is '29', the material is 'steel', the weight is '3', the quantity of spokes is '28' and the cost is '600'.

Similarly, in order to meet the user's requirements, a GASI may be modified. This is related to the decision taken by the user to use the structural view.

Definition 29: GASI modification
GASI modification is a process of modifying or adapting a GASI during ETO configuration by enriching their knowledge to meet ETO requirements. This may lead to an ETO solution that will be stored in the EB. This modification turns the $CTO.GASI^{(i)}_j$ into an $ETO.GASI^{(i)}_j$.

GASI modification is a two-step process:

Step 1) Modify the name and description of the current $ETO.GASI^{(i)}_j$ (if necessary),

Step 2) Enrich the knowledge of the current $ETO.GASI^{(i)}_j$: based on (Sylla, Guillon, Vareilles, et al., 2018) five cases are proposed. The user can enrich the knowledge of the current $ETO.GASI^{(i)}_j$ by adding specific knowledge to fulfill ETO requirements:

The BOM is modified in the following three cases.

- Case 1: Enlarge the quantity domains of $GAI^{(i)}$ within $ETO.GASI^{(i)}_j$: the user wants a quantity outside the existing domains of quantities. Therefore, to meet this requirement the new value of quantity must be added to the domain which did not previously exist. For example, enlarging a quantity of a $GAI^{(i)}$ from {[0, 1]} to {[0, 1], 3} will allow to use 0, 1 or 3 $GAI^{(i)}$ within the $ETO.GASI^{(i)}_j$. Enlarging the quantities domains of $GAI^{(i)}$ increases the solution space. After enlarging the quantities of $GAI^{(i)}$, KPIs aggregation methods must be updated.
- Case 2: Replace a $GAI^{(i)}$ with another $GAI^{(i)}$ within $ETO.GASI^{(i)}_j$: the user may require another $GAI^{(i)}$ existing in the taxonomy instead of a $GAI^{(i)}$ within $ETO.GASI^{(i)}_j$. In addition, the user may require to replace a $GAI^{(i)}$ with another $GAI^{(i)}$ which has been previously modified (resulting from previous ETO cases). Then, after replacing the GAI, KPIs aggregation methods need to be updated. Replacing a $GAI^{(i)}$ with a modified $ETO.GAI^{(i)}$ can increase the solution space.
- Case 3: Add new or existing $GAI^{(i)}$ with their quantity domains to the $ETO.GASI^{(i)}_j$: the user may request a new $GAI^{(i)}$ with its quantity domain to be added to the structural view. Therefore, to fulfill this requirement, one or more $GAI^{(i)}$, whether they are new or existing, along with their quantities, must be defined. Then, they are integrated into the current $ETO.GASI^{(i)}_j$. This allows to define an $ETO.GASI^{(i)}_j$ with a specific structure based on the user requirements. Then, KPIs aggregation methods need to be updated. These new $GAI^{(i)}$ which have been added to the $ETO.GASI^{(i)}_j$ make the solution space increase.

The knowledge is modified in the following two cases.

- Case 4: Define new combinations of attributes and/or KPIs values of $GAI^{(i)}$: the user demands a new combination of attributes and/or KPIs values of $GAI^{(i)}$s that were not allowed. Thus, a relation must be extended to link at least two incompatible values of attributes and/or KPIs of various $GAI^{(i)}$. Adding this new combination of attributes and/or KPIs values of $GAI^{(i)}$ increases the solution space. This case is similar to case 2 of $ETO.GADI^{(i)}$ modification which was related to the relations within each $ETO.GADI^{(i)}$. However, here the modification of the $ETO.GASI^{(i)}_j$ is related to the relations between attributes and/or KPIs values of different $GAI^{(i)}$ within the $ETO.GASI^{(i)}_j$.

- Case 5: Define new relations between $GAI^{(i)}$ or between attributes and/or KPIs values of $GAI^{(i)}$: the user may require a new relation that does not exist between $GAI^{(i)}$ or between attributes and/or KPIs values of the $GAI^{(i)}$ which compose the $ETO.GASI^{(i)}_j$. The user may require to explicitly represent the allowed or forbidden combinations of $GAI^{(i)}$ or combination of attributes and/or KPIs values of $GAI^{(i)}$. Thus, new relations must be defined which can link i) new $GAI^{(i)}$ with existing $GAI^{(i)}$, ii) new $GAI^{(i)}$, iii) attributes and/or KPIs values of new $GAI^{(i)}$ and existing ones, or iv) attributes and/or KPIs values of new $GAI^{(i)}$. This case is very similar to case 4 of $ETO.GADI^{(i)}$ modification. However, here the relations between different $GAI^{(i)}$s (or between attributes and/or KPIs values of different $GAI^{(i)}$) are considered.

We propose to modify the CSP as follows in $ETO.GASI^{(i)}_j(CSP)$.

- Case 1: Enlarging the quantities of $GAI^{(i)}$ corresponds to adding new values to the corresponding domains of variables,

- Case 2: Replacing a $GAI^{(i)}$ with another $GAI^{(i)}$ corresponds to replacing an existing CSP with another CSP,

- Case 3: Adding new or existing $GAI^{(i)}$ with their quantities corresponds to adding new CSPs and new variables.

- Case 4: Defining new combinations of attributes and/or KPIs values of $GAI^{(i)}$s corresponds to adding new tuples to the compatibility tables. The user extends the constraint by adding new constraint tuples linking at least two incompatible values of variables of $GAI^{(i)}$s.

- Case 5: Defining new relations corresponds to adding new constraints. These new relations are formalized as constraints that can be compatibility tables or numerical functions.

For example, as shown in Figure 51, to meet user's ETO requirements, the $ETO.Bike.GASI^{(2)}_1$ is modified and then it is translated into a CSP in which the CSP of the $Bike.GADI^{(2)}$ is embedded. The user wants to replace the existing $Wheel.GAI^{(2)}$ with the modified $Wheel.GAI^{(2)}$. To fulfill this requirement, in $ETO.Bike.GASI^{(2)}_1$, $Wheel.GAI^{(2)}$ is replaced with the modified one ($E: Wheel.GAI^{(2)}$). The letter 'E' at the beginning of the $Wheel.GAI^{(2)}$ represents that this

GAI has been modified in ETO configuration activity. In *ETO.Bike.GASI$^{(2)}_1$(CSP)*, *Wheel.GASI$^{(2)}$(CSP)* which is an embedded CSP is replaced with *E: Wheel.GASI$^{(2)}$(CSP)* which is a modified CSP since its corresponding *Wheel.GADI$^{(2)}$* is modified.

In this example, the user wants a bike with two seats. Thus, to meet this requirement in *Bike.GASI$^{(2)}_1$* the quantity of a *Seat.GAI$^{(2)}$* is enlarged and the value 'E: 2' is added to the quantity. Moreover, in *ETO.Bike.GASI$^{(2)}_1$(CSP)*, the mentioned value is added to the domain of the variable SeatQty. It is related to the case 1.

The user wants also a bike with a mirror. Therefore, in *ETO.Bike.GASI$^{(2)}_1$* a new GAI with its quantity is added (i.e. E: ({1}, *Mirror.GAI$^{(i)}$*)). In addition, in *ETO.Bike.GASI$^{(2)}_1$(CSP)*, a new CSP is added (*E: Mirror.GASI$^{(i)}$(CSP)*) as well as a new variable corresponding to its quantity is added (*E: MirrorQty*). It is related to the case 3.

The user wants to define a relation between the categories of bike users and the material of the seat. Therefore, in *ETO.Bike.GASI$^{(2)}_1$*, a relation is extended (E: the user child is only compatible with the material 'carbon'). Moreover, in *ETO.Bike.GASI$_1$(CSP)* an existing constraint is enlarged by adding a tuple that links incompatible values of *Bike.GADI$^{(2)}$.User* and *Seat.GADI$^{(2)}$.Material*. It is related to the case 4.

The user wants also to define a relation between the quantity of mirrors and the quantity of seats. Therefore, in *ETO.Bike.GASI$^{(2)}_1$*, a new relation is added (E: Quantity of mirror depends on the quantity of seat). In addition, this relation in *ETO.Bike.GASI$^{(2)}_1$(CSP)* is translated into a new compatibility table linking the compatible values of *Seat.GADI$^{(2)}$.SeatQty* and *E: Mirror.GADI$^{(i)}$.MirrorQty*. If there is only one seat, there can be one or two mirrors. But if there are two seats, it is mandatory to have two mirrors. This situation is also related to the case 5.

In *ETO.Bike.GASI$^{(2)}_1$(CSP)*, two constraints related to KPIs aggregation methods are updated. In which, whatever is modified during configuration is tagged with 'E' otherwise it is not tagged. It is important to notice that constraint filtering is not applied to check the consistency of the built *ETO.Bike.GASI$^{(2)}_1$(CSP)*. The ETO configuration activity has to be performed manually. If the user wants two seats, the solution will be quantity of two mirrors. If the bike is for a child, the material of the seat will be 'carbon'. It is done manually.

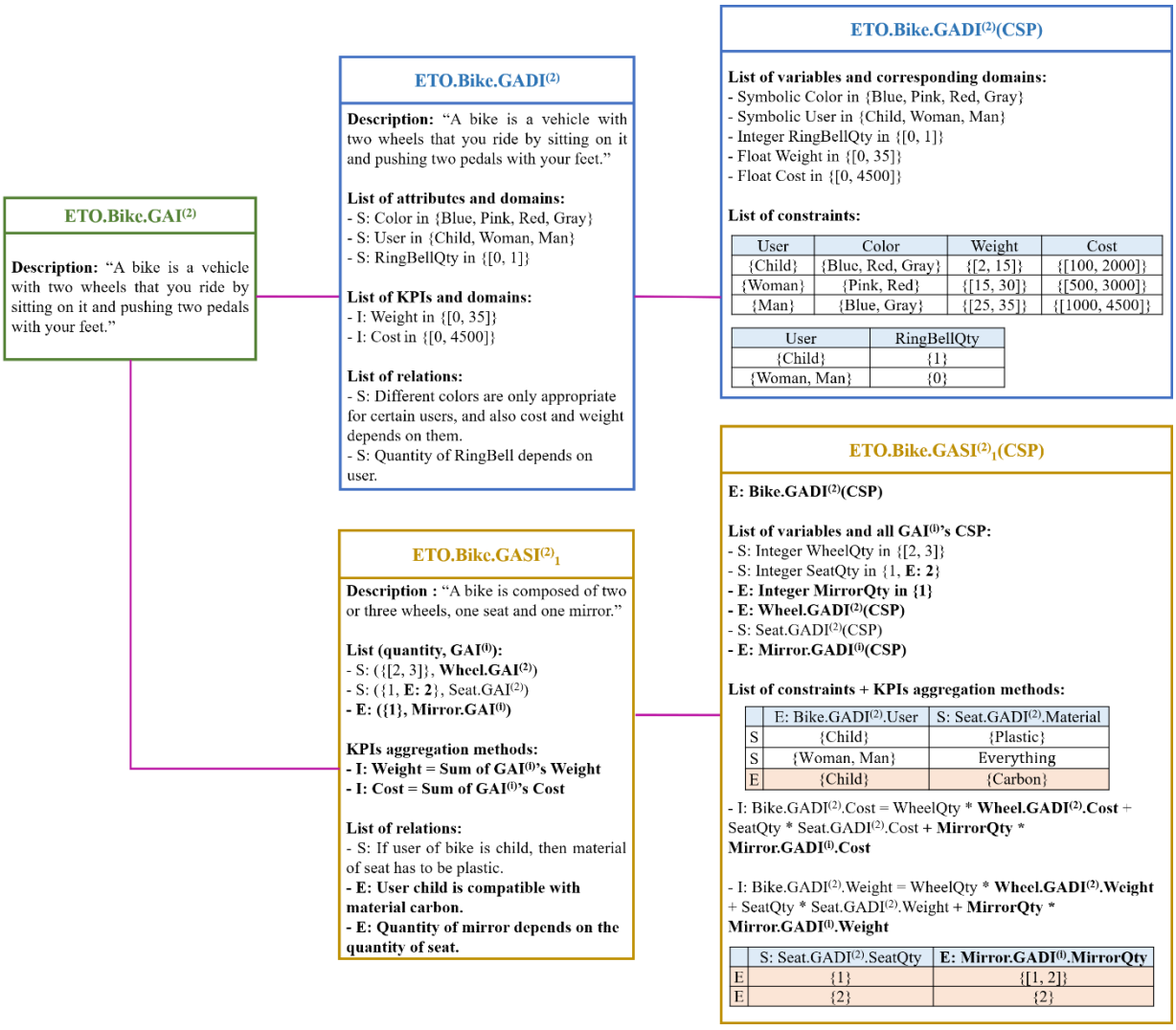

Figure 51. Example of modifying *ETO.Bike.GASI*$^{(2)}_{1}$ and its translation into a CSP

In the following section, the synthesis of the ETO configuration activity is presented.

4.2.3. Synthesis

In this section, we have focused on our second research question: " How can ETO requirements be processed during configuration activity?" by adapting configuration activity. Our approach involved proposing an adaptation of CTO to ETO by 1) creating a new GADI or GASI and 2) modifying GADI or GASI.

For the modification of *ETO.GADI*$^{(i)}$, based on the work of (Sylla, Guillon, Vareilles, et al., 2018) four cases were proposed. While, for the modification of *ETO.GASI*$^{(i)}_{j}$ five cases were proposed and the translation into a CSP was proposed as well (*ETO.GASI*$^{(i)}_{j}$*(CSP)*). All these cases were discussed which enables us to meet non-negotiable non-standard requirements, also referred to as ETO requirements. They are first formalized during configuration and then capitalized by storing the obtained ETO solutions of GADI or GASI in an EB at the end of configuration.

Furthermore, our approach enables us to interactively configure an instance each time user defines one requirement. In situations where new GADI or GASI are built, the consistency of these instances is checked to ensure that they are consistent pieces of knowledge ready to be integrated into the current instance. However, during GADI or GASI modification, the consistency of instances cannot be checked using filtering techniques because all the added values will be removed.

ETO configuration activity is guided by the formalized knowledge. All the standard generic models (GA, GADM, GASM) which have been defined beforehand and stored into the knowledge base can be reused, adapted, modified or completed by the user according to the non-negotiable and non-standard requirements (ETO requirements). Following the requirements and the level of expertise of the user, either descriptive view or structural view can be exploited. During the ETO configuration activity, the generic models are instances (GAI, GADI, GASI). They are instantiated from generic models or they are created specifically to answer to ETO requirements. All the generic model instances modifications or creations correspond to engineering activities performed by designers who have changed the standard generic model instances of systems. The ETO configuration is then performed manually by the user, checking manually that every constraint is satisfied and every variable value belongs to its validity domain. This is due to the fact that the user cannot modify all the constraints belonging to the generic model instances (integrating all the added values). This can only be done during knowledge formalization and not during knowledge reuse. However, for every new instance, the user is allowed to define constraints which will enable to check its consistency. However, as they are activities which are performed out of the knowledge formalization process, most of the time, no constraint is created. At the end of an ETO configuration activity, the obtained models (GAI, GADI, GASI) are stored into an Experience Base. This enable to reuse them during the knowledge formalization process. Every model which has been created specifically to answer to ETO requirements can be standardized and translated from instances to generic models. Moreover, in order to answer to new requirements, instances can be reused, adapted and modified. However, the exploitation of system configuration experiences is not treated in this thesis.

4.3. CTO-ETO knowledge reuse

We have proposed two generic processes for knowledge reuse. One process is focused on CTO configuration (described in section 4.1), while the other one is focused on ETO configuration (presented in section 4.2). Initially, we assumed that the user can only be in either the CTO or ETO context and cannot switch between the two. However, in reality, users may have both CTO and ETO requirements. This means that a user might begin by defining requirements that can be fulfilled by reusing the generic model and then proceed to those that cannot. Consequently, a user may start with the CTO configuration and later shift to the ETO configuration. The generic process for knowledge reuse in both CTO and ETO configuration is illustrated in Figure 52 and Figure 53. The CTO-ETO process is designed to support the following scenarios. Each scenario describes a different path a user might take to meet their specific requirements, depending on whether those requirements can be met using standard

models (CTO) or require additional engineering (ETO), and whether they involve only the descriptive view of the selected $GA^{(i)}$ or its structural view as well.

Scenario 1: we start with configuration of the descriptive view in CTO then shifting to the configuration of descriptive view in ETO. This scenario begins with the configuration of descriptive view in CTO. However, since the user's requirements cannot be met by reusing the existing generic models, the CTO is shifted into ETO. This shift involves modifying $GADI^{(i)}$ and then manually configuring in order to meet ETO requirements. The user is not interested to configure structural view. Therefore, at the end of configuration, an ETO solution is proposed then capitalized in an EB.

Scenario 2: we start with configuration of the descriptive view in CTO, then shifting to the configuration of descriptive view in ETO, and subsequently continuing with the configuration of structural view in ETO. Similar to the previous scenario, due to the ETO requirements of the user, the shifting from CTO to ETO is required (related to the configuration of descriptive view). Then, since the user wants to use the structural view, the configuration of GASI is performed in ETO. Note that the $GAI^{(i)}$s within $GASI^{(i)}_j$ are configured by implementing the sub process of "Configure CTO GAI". The output here is an ETO solution (including the result of a configured $GADI^{(i)}$ plus a configured $GASI^{(i)}_j$).

Scenario 3: we start with the configuration of the descriptive view in CTO, then configure the structural view in CTO and ultimately shifting to the configuration of structural view in ETO. In this scenario, the $GADI^{(i)}$ is configured in CTO. However, configuring the structural view in CTO is not possible. Therefore, the shifting from CTO to ETO is required, in which the $GASI^{(i)}_j$ must be modified. The output is an CTO solution (the output of GADI configuration) and an ETO solution (the output of GASI configuration).

With regard to these three scenarios, the solutions for such a configuration process can be of three different types:

1) 100% $CTO.GAI^{(i)}$, if all the requirements were expressed in terms of the selected generic models; the CTO process, presented in section ,4.1.1.3 has been carried out from start to finish.

2) 100% $ETO.GAI^{(i)}$, if all the requirements are outside the possibilities offered by the generic models - the ETO process, presented in section 4.2.1, has been carried out from start to finish.

3) $CTO\text{-}ETO.GAI^{(i)}$, where some of the requirements were supported by the generic models and some were not. In this case, the $GAI^{(i)}$ solution contains elements marked CTO and others marked ETO.

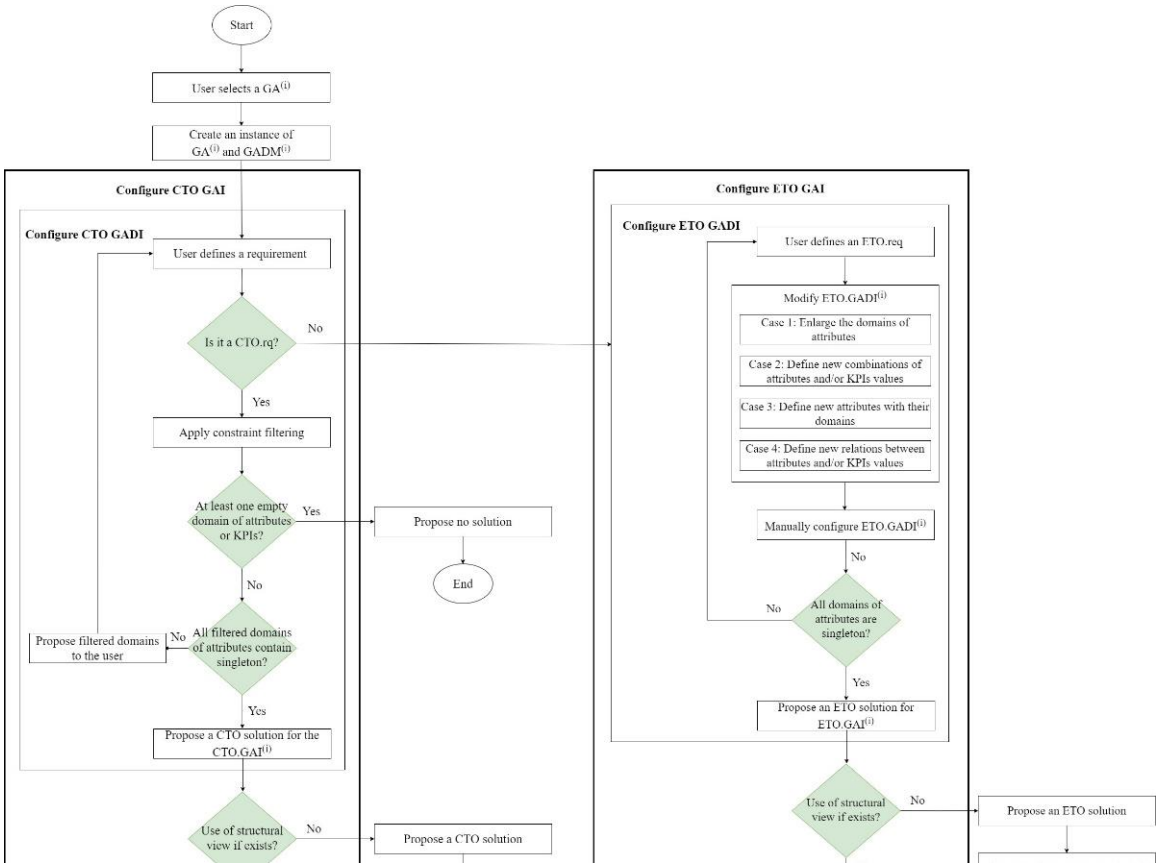

Figure 52. Flowchart for knowledge reuse in CTO-ETO configuration: GADI configuration

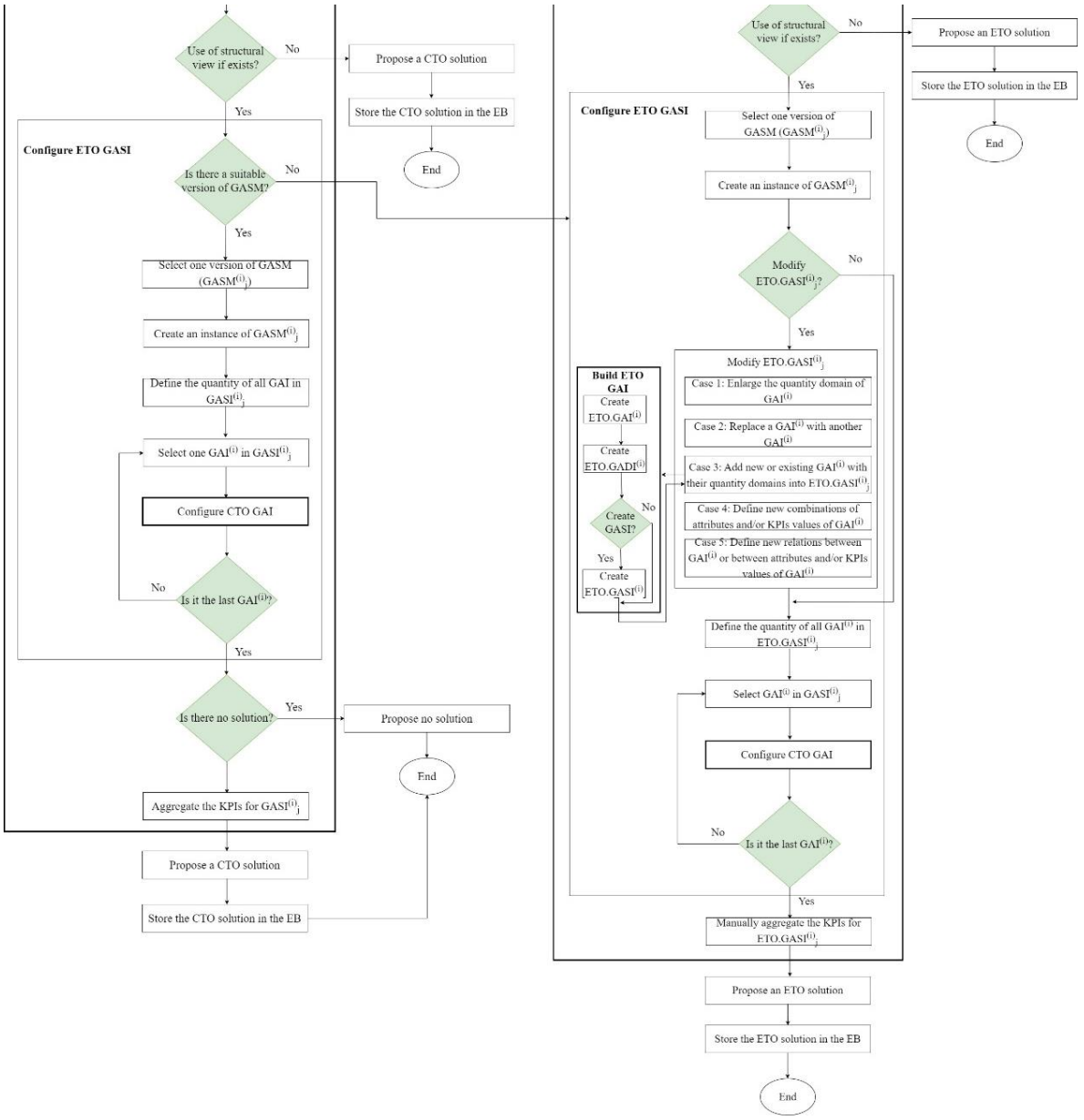

Figure 53. Flowchart for knowledge reuse in CTO-ETO configuration: GASI configuration

4.4. Synthesis

The second objective of our research work is to reuse or exploit the formalized generic models for system configuration. This chapter presents our second contribution, a knowledge reuse process for system configuration by adapting the CTO configuration activity to meet ETO requirements. It comes in response to our second research question: "How can ETO requirements be processed during configuration activity?"

Our proposals in section 4.1, allow us to select a generic model, following the needs of descriptive or structural view, then configure the instance of this generic model in CTO configuration activity to meet CTO requirements. Therefore, we first defined Generic Artifact Descriptive Instance ($GADI^{(i)}$) for the descriptive view and Generic Artifact Structural Instance ($GASI^{(i)}_j$) for the structural view and illustrated how these instances can be configured to meet

user requirements. In this section, a generic process for knowledge reuse in CTO is proposed and leads to $CTO.GAI^{(i)}$ solution.

Our proposals in section 4.2, are dedicated to knowledge reuse in ETO situations in order to fulfill non-negotiable non-standard requirements in the ETO configuration activity. Therefore, we proposed adapting CTO configuration activity towards ETO through either building new GADI or GASI or modifying existing ones. To formalize ETO requirements, we defined four cases for GADI modification and five cases for GASI modification, then formalized these cases using CSP. The ultimate goal was to capitalize ETO solutions obtained in the ETO configuration activity in an Experience Base (EB). Each of the attributes, values, KPIs, KPIs aggregation methods, constraints and tuples can be tagged by the letter 'E' to capitalize the ETO knowledge. This modifications of the tags to (S/I) to (E) is propagated to GAI, GADI and GASI. In this section, a generic process for knowledge reuse in ETO is proposed and leads to $ETO.GAI^{(i)}$ solution.

Our proposals in section 4.3, are dedicated to link between knowledge reuse in both CTO and ETO situations in order to fulfill both CTO and ETO requirements for system configuration. In this way, several scenarios are explained, and a generic process is illustrated using a flowchart. Considering both CTO and ETO requirements leads to a solution that is partly configured according to a generic model and partly designed for specific requirements. The solution is therefore made up of CTO and ETO artifacts in varying proportions, depending on the user's needs.

This research led to an oral presentation at the SAGIP 2023 conference:

- Maryam Mohammad Amini, Thierry Coudert, Elise Vareilles, Michel Aldanondo, Integration of constraint satisfaction problems and ontologies for the formalization and exploitation of knowledge in system configuration, SAGIP 2023, Marseille.

In the next chapter, we illustrate our proposals related to knowledge formalization and knowledge reuse on a simplified but realistic example of a bicycle and its implementation in OPERA, a software developed in the context of the ANR project OPERA (ANR-16-CE10-0010) for system configuration and risk management.

5. Use case and its implementation in OPERA: a bike example

In Chapter 3, we have presented our first contribution: a knowledge formalization process for system configuration using the association of ontologies, CSP approaches, commonality and inheritance principles to create generic models at different levels of abstraction. Chapter 4 is devoted to our second contribution: a knowledge reuse process for system configuration by adapting the CTO configuration activity to ETO configuration activity in order to meet ETO requirements.

In this chapter, we illustrate our proposals on a simplified but realistic example of a bicycle configuration implemented on the OPERA software. In section 5.1, we start with a short presentation of the OPERA software followed by the use case presentation. Then, in section 5.2, we illustrate our proposals for knowledge formalization in system configuration. Some examples of chapter 3 are implemented using the OPERA software. In section 5.3, we illustrate our proposals for knowledge reuse in system configuration (CTO situation and ETO situation). Finally, in section 5.4 we synthesize the chapter.

5.1. OPERA software and use case presentation

This section is devoted to the introduction of the OPERA software tool and the use case.

5.1.1. OPERA software

The OPERA project (acronym for "**O**utils logiciels et **P**roc**E**ssus pour la **R**éponse à **A**ppels d'offres") was initiated in November 2016 with funding from the ANR[5] and began in November 2016. Its primary objective is to provide bidding companies with knowledge-based support tools, enabling them to efficiently and confidently develop relevant bids when responding to invitations to tender. The consortium responsible for this endeavor consists of three research laboratories, namely CGI at IMT Mines Albi, ESTIA Recherche, and LGP at ENIT, along with four industrial partners - AES, Altran, Axsens, and Mécanumériс.

The OPERA software has been developed in order to formalize knowledge and build solutions in the situation of call for tenders. Firstly, by means of the OPERA software, experts can formalize knowledge on solutions for systems or services that can be delivered. It is possible to formalize knowledge on systems and services along with their realization process. Moreover, it is possible to formalize knowledge on risks which can arise during the process realization (the identified risks, their probability of occurrence, their impacts on the process and the activities to mitigate them). Then, it is possible to implement CSPs and to propagate constraints using filtering methods in order to maintain the consistency of the models. Secondly, according to customer's requirements, the user can configure solutions interactively (systems/services and processes can be configured) using constraint propagation (Guillon, Ayachi, et al., 2021), (2) evaluate these solutions according to the risks and to the confidence the user has on the requirements satisfaction (Sylla, Guillon, Vareilles, et al., 2018). Then, the user is helped to select a solution which fulfill all the requirements and in which she/he is confident.

In this thesis, we use the OPERA software as a system configuration software to implement and then verify our proposals for the knowledge formalization and knowledge reuse phases. The OPERA software has been chosen because it stands out as a tool that allows to: (1) formalize an ontology of generic models for system configuration following descriptive views and structural views (GA, GADM and GASM)), (2) handle different levels of abstraction for these generic models, (3) create instances in order to interactively configure systems.

5.1.2. Use case presentation

In this chapter, as illustrated in Figure 54, we check our proposals on a simplified but realistic example of knowledge formalization and knowledge reuse phases for bicycle configuration. We consider that a bike is only composed of two or three wheels, one seat, and one frame. To illustrate our proposal related to the knowledge formalization phase, first, we illustrate generic models creation (i.e. GA, GADM and GASM creation) on a bike example. Then, we illustrate GA generalization on a wheel example, in which City Wheel and Mountain Wheel generic models are generalized into Wheel generic models. Then, we show GA specialization on a bike

[5] Project n° ANR-16-CE10-0010

example in which Bike generic models are specialized into City Bike and Mountain Bike generic models.

To illustrate our proposal related to the knowledge reuse phase, first, we illustrate GADI filtering on a bike example. We present GASM selection on a wheel example, in which we select one version among two versions of a wheel (i.e. mountain wheel and city wheel). We illustrate GAI and GADI building on the mirror example. We show GADI modification on the instance of a wheel, in which we decided to add an attribute and a new constraint. We represent GASI modification on an instance of a bike where we decided to only add a new GAI (corresponding to a mirror) and a new constraint (the number of seats is constrained by the number of mirrors).

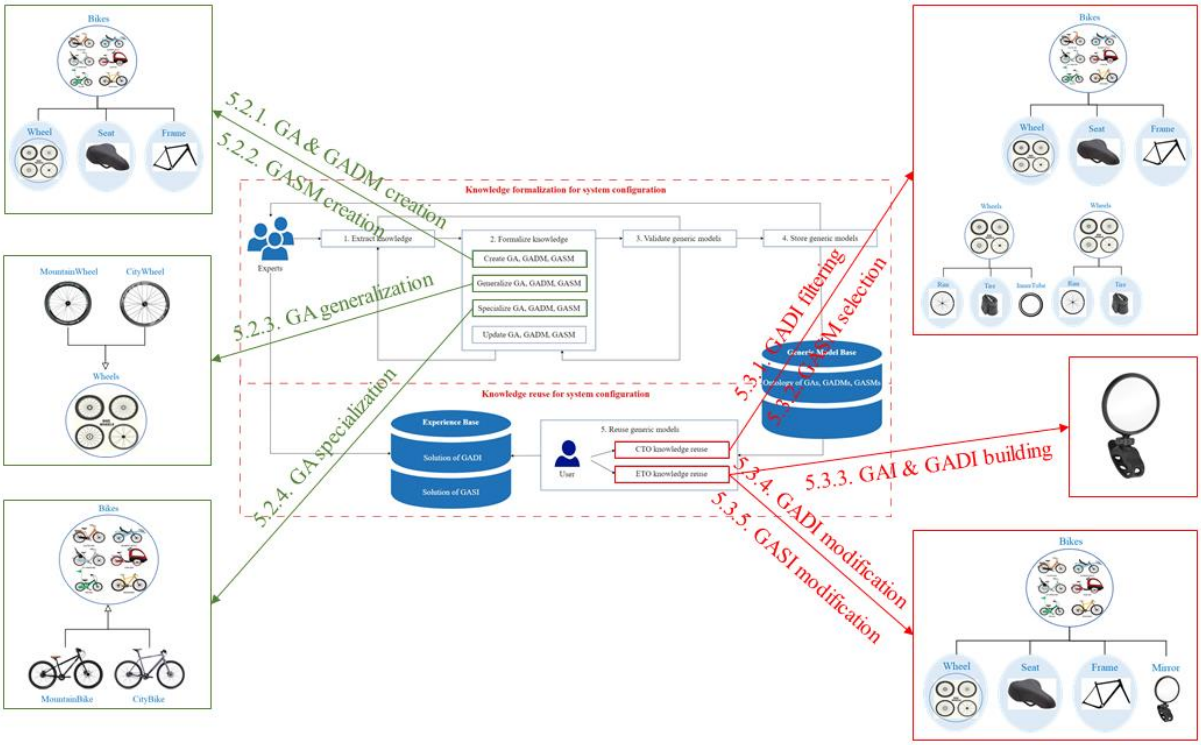

Figure 54. Scope of bike use case

5.2. Knowledge formalization for system configuration

This section is devoted to the application of our knowledge formalization proposals, described in Chapter 3, to the previously presented case study of bicycle configuration. First, in section 5.2.1, *Bike.GA*$^{(2)}$ and *Bike.GADM*$^{(2)}$ are defined. In section 5.2.2, *Bike.GASM*$^{(2)}{}_{1}$ is created. In section 5.2.3, *MountainWheel.GA*$^{(3)}$ and *CityWheel.GA*$^{(3)}$ are generalized into *Wheel.GA*$^{(2)}$, and in section 5.2.4, *Bike.GA*$^{(2)}$ is specialized into *CityBike.GA*$^{(3)}$ and *MountainBike.GA*$^{(3)}$.

5.2.1. GA and GADM Creation

First, we create the taxonomy of GAs. The most general GA is *System.GA*$^{(1)}$. It is specialized into several GAs as represented in Figure 55. For every GA, a name and a description are given (the Figure 55 only shows the taxonomy of GAs and the *Bike.GA*$^{(2)}$ description).

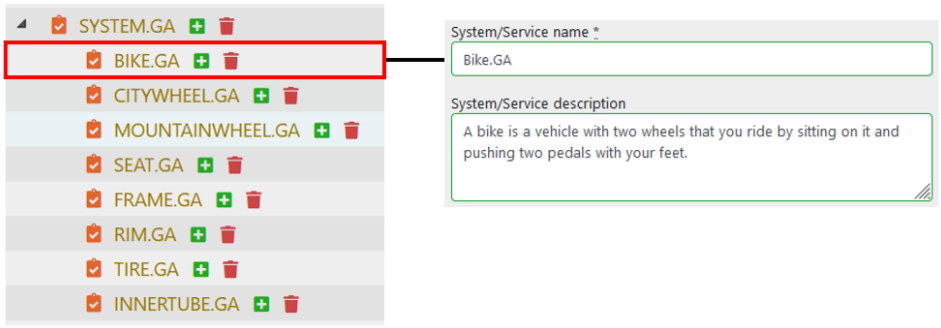

Figure 55. Taxonomy of GAs and the description of *Bike.GA*$^{(2)}$

Therefore, for every GA, the corresponding GADM (descriptive view) is created. That leads to the creation of the GADM taxonomy (not represented)

Associated to *System.GA*$^{(1)}$, we define *System.GADM*$^{(1)}$ as the most general GADM, containing two KPIs: Weight with a validity domain of {[0, 35]} and Cost with a validity domain of {[0, 4500]}. As we create all the other GADMs as a specialization of *System.GADM*$^{(1)}$, they inherit the Cost and Weight with the same domain.

We take the example of *Bike.GADM*$^{(2)}$, as illustrated in Figure 13. Then, we define *Bike.GADM*$^{(2)}$ as shown in Figure 56. To do so, we define three specific attributes. Two KPIs are inherited. We also define two constraints (compatibility tables). These tables represent sequences of compatible values for both attributes and KPIs. For instance, in the first compatibility tables, the first value of every tuple corresponds to the value of the variable 'User', the second one corresponds to the value of the variable 'Color', the third one corresponds to the value of the variable 'Weight' and the last one corresponds to the value of the variable 'Cost'. In the second table, the values correspond to the compatible values of the variables 'User' and 'RingBellQty' respectively.

Filtering these constraints leads to restricting the initial domains of the attributes and KPIs in order to obtain a consistent generic model. As represented in Figure 57, after filtering constraints, the initial domain of Weight is restricted to {[2, 35]} and for Cost to {[100, 4500]} which are represented in green boxes. The *Bike.GADM*$^{(2)}$ becomes a consistent piece of knowledge corresponding to the descriptive view of *Bike.GA*$^{(2)}$ which will be reused later in the knowledge reuse section..

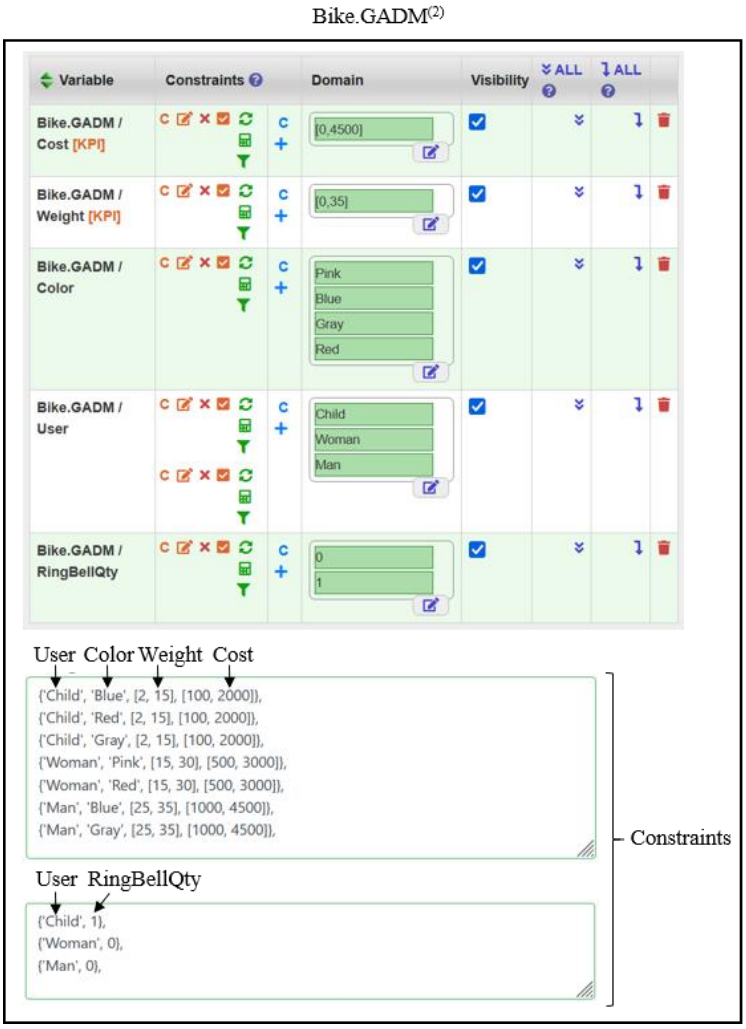

Figure 56. *Bike.GADM*$^{(2)}$ before filtering constraints

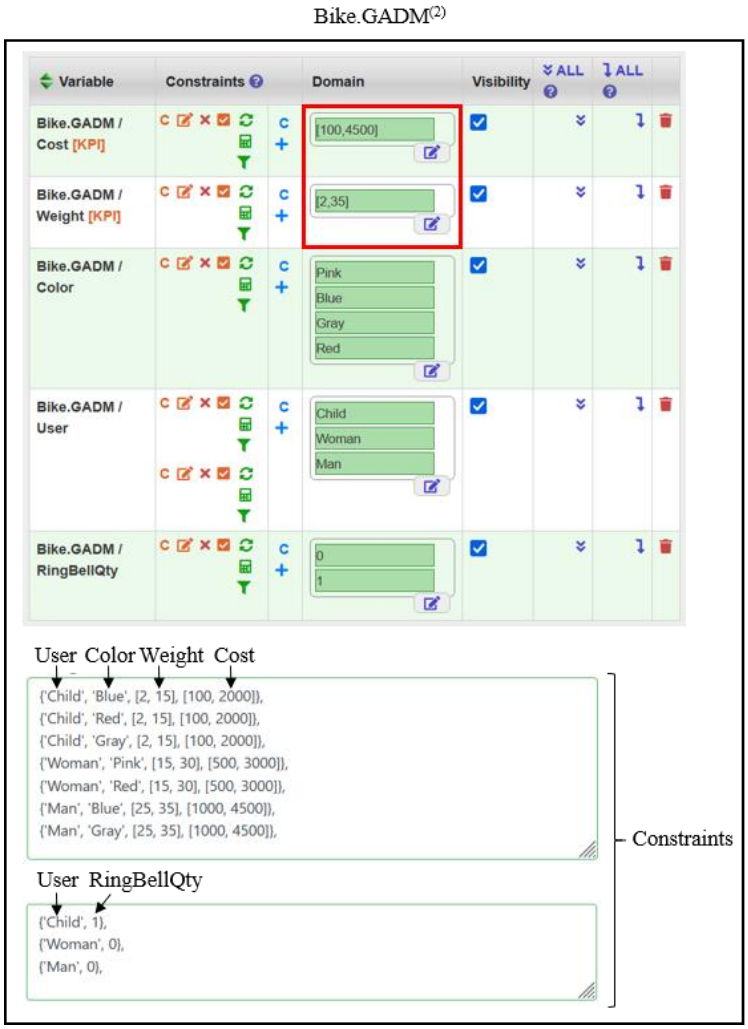

Figure 57. *Bike.GADM*$^{(2)}$ after filtering constraints

5.2.2. GASM Creation

We use the example of Figure 15 to create *Bike.GASM*$^{(2)}{}_1$, the first version of the structural view of *Bike.GA*$^{(2)}$. It is composed of two or three *Wheel.GA*$^{(2)}$ and one *Seat.GA*$^{(2)}$. We have also included a *Frame.GA*$^{(2)}$. We create *Bike.GASM*$^{(2)}{}_1$ (associated to *Bike.GADM*$^{(2)}$) as illustrated in Figure 58. We define its description. It should be noticed that in the OPERA software, we can define explicitly the quantity of GAs if they are a constant. If the quantity of a GA is a range or interval, a variable corresponding to the quantity of GA needs to be defined in the GASM. In our example, the quantity of *Seat.GA*$^{(2)}$ and *Frame.GA*$^{(2)}$ is '1', however, for *Wheel.GA*$^{(2)}$ the variable WheelQty is added since the quantity of wheel is an interval {[2, 3]}.

We define a constraint (compatibility table) in which the first and second values of every tuple respectively correspond to the value of the variable 'User' of *Bike.GADM*$^{(2)}$ and the variable 'Material' of *Seat.GADM*$^{(2)}$. We define two constraints (numerical functions) to aggregate the Cost and Weight for *Bike.GADM*$^{(2)}$. The first one represents the Weight for *Bike.GADM*$^{(2)}$ is equal to the sum of the weights of all GAs composing *Bike.GASM*$^{(2)}{}_1$ regarding their quantities. Similarly, the second one represents the Cost for *Bike.GADM*$^{(2)}$ is equal to the sum of the costs of all GAs composing *Bike.GASM*$^{(2)}{}_1$ considering their quantities. After constraint filtering, as

shown in Figure 58, the domains of Cost and Weight are further restricted to {[2.8, 17]} and {[215, 2210]} and the domain of variables 'User' (for *Bike.GADM*$^{(2)}$) and 'Material' (for *Seat.GADM*$^{(2)}$) are not changed.

In the Figure 58, one can observe that *Wheel.GA*$^{(2)}$ has its own GASM as it is composed of *Rim.GA*$^{(2)}$ and *Tire.GA*$^{(2)}$. *Bike.GASM*$^{(2)}{}_1$ is then a consistent piece of knowledge representing the first version of the structural view of *Bike.GA*$^{(2)}$.

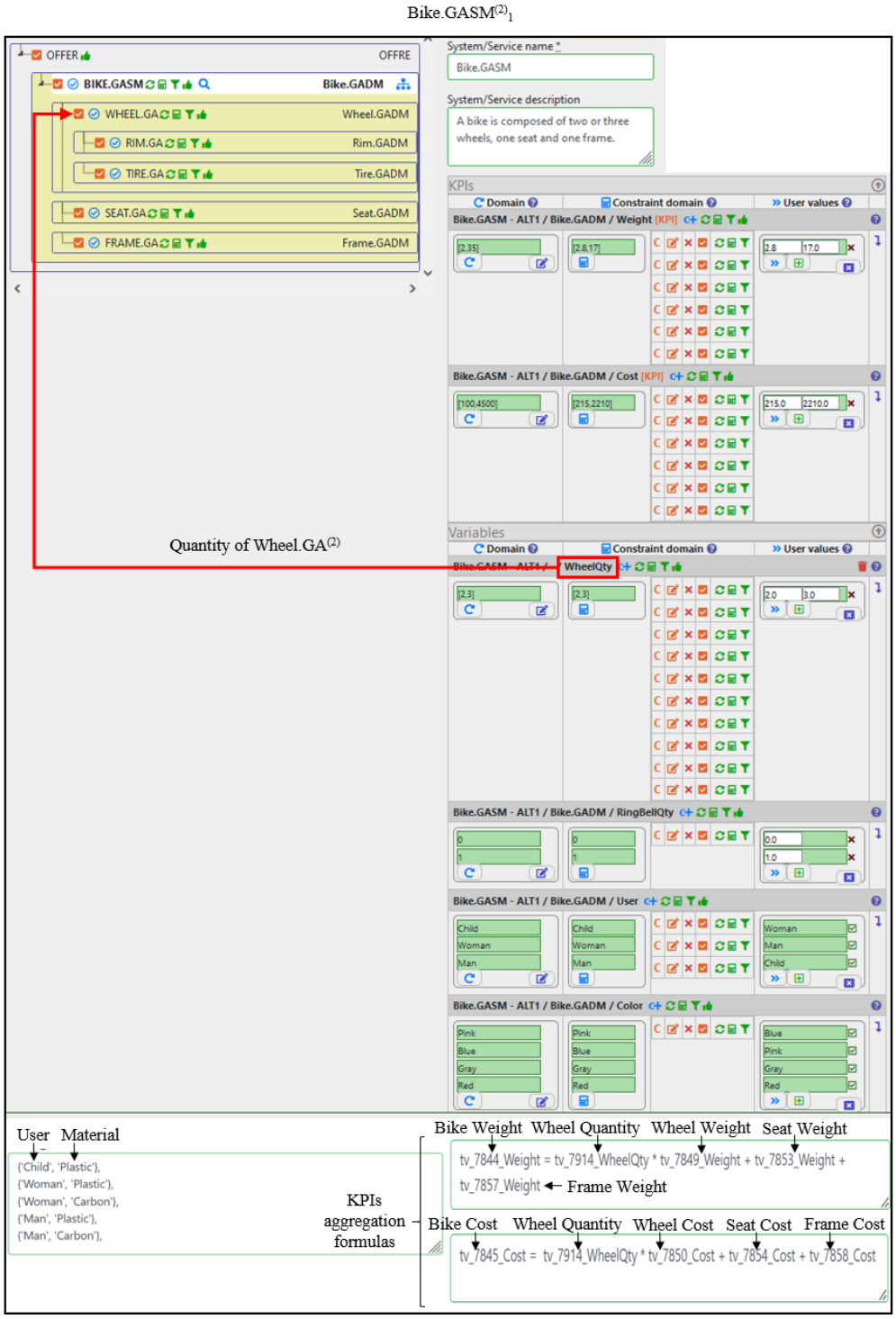

Figure 58. *Bike.GASM*$^{(2)}{}_1$

5.2.3. GA Generalization

As indicated in section 3.2.1, generalization enables to create a GA at a higher level of abstraction using the commonality of specific existing GAs. The generalization of GAs implies that both the associated GADM and GASM are also generalized.

First, we create *MountainWheel.GA*$^{(2)}$ (not represented here), *MountainWheel.GADM*$^{(2)}$, *CityWheel.GA*$^{(2)}$ (not represented here) and *CityWheel.GADM*$^{(2)}$, using the example of Figure 20. We define *MountainWheel.GADM*$^{(2)}$ as illustrated in Figure 59: we define two attributes, two inherited KPIs (there are inherited from *System.GADM*$^{(1)}$), and a compatibility table. In this compatibility table, the values of the tuples respectively correspond to the values of the attributes and KPIs 'Diameter', 'Material', 'Weight', and 'Cost'. Similarly, we define *CityWheel.GADM*$^{(2)}$ as shown in Figure 60. For this GADM, we define three attributes and two inherited KPIs. Two compatibility tables are also defined. In the first compatibility table, the values of tuples respectively correspond to values of the variables 'Diameter', 'Material', 'Weight', and 'Cost'. In the second one, the first and second values correspond to the values of 'Diameter' and 'InnerTubeQty' respectively. The domains of variables are represented after filtering the constraints for *MountainWheel.GADM*$^{(2)}$ and *CityWheel.GADM*$^{(2)}$.

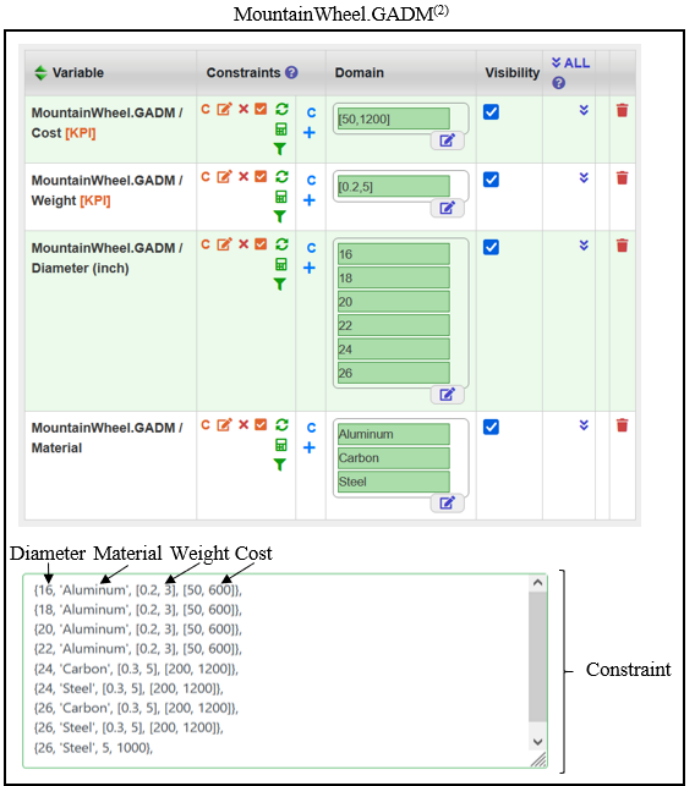

Figure 59. *MountainWheel.GADM*$^{(2)}$

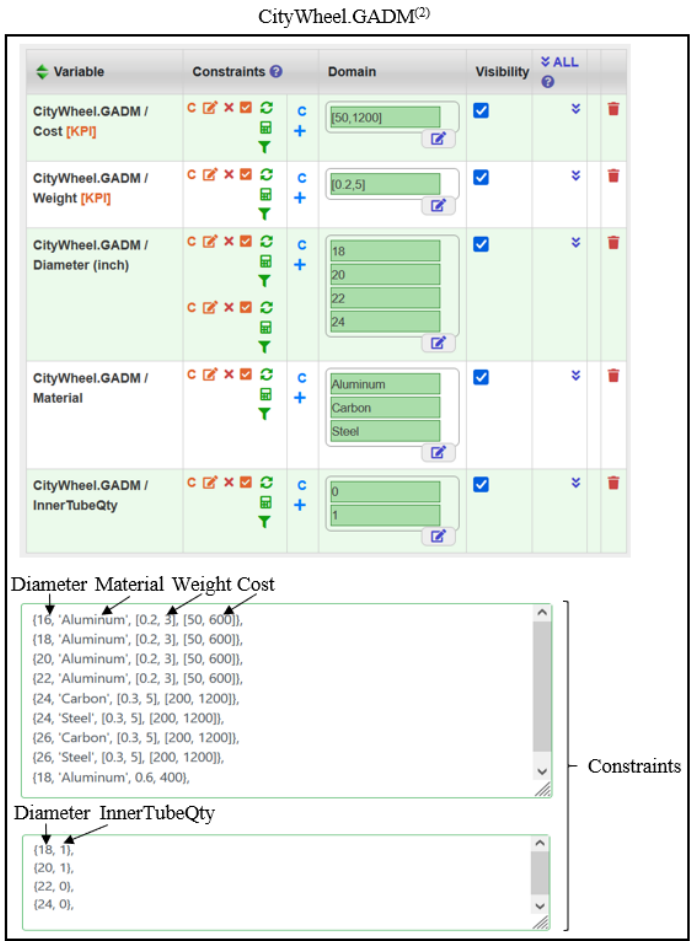

Figure 60. *CityWheel.GADM*$^{(2)}$

To create *Wheel.GADM*$^{(2)}$ by generalization of *MountainWheel.GADM*$^{(2)}$ and *CityWheel.GADM*$^{(2)}$, we refer to Figure 21. Based on the commonality of *MountainWheel.GADM*$^{(2)}$ and *CityWheel.GADM*$^{(2)}$, we define *Wheel.GADM*$^{(2)}$ as illustrated in Figure 61. *MountainWheel.GADM*$^{(2)}$ and *CityWheel.GADM*$^{(2)}$ are moved to the lower level (they become *MountainWheel.GADM*$^{(3)}$ and *CityWheel.GADM*$^{(3)}$ as illustrated in Figure 62). We define two common attributes (Diameter and Material), two common KPIs (Weight and Cost) as well as a common table of compatibility linking the compatible values of the following attributes and KPIs respectively: 'Diameter', 'Material', 'Weight' and 'Cost'. As shown in Figure 61, after filtering this constraint, the domain of 'Diameter' is restricted to {16, 18, 20, 22, 24, 26}, the domain of weight to [{0.2, 5}] and the domain of cost to [{50, 1200}].

To summarize, in the OPERA software, we can define GADMs at different levels of abstraction. However, the tags 'I' and 'S' which we used in our proposal to distinguish the inherited and specific characteristics (i.e. attributes, KPIs, domains, relations) cannot be defined.

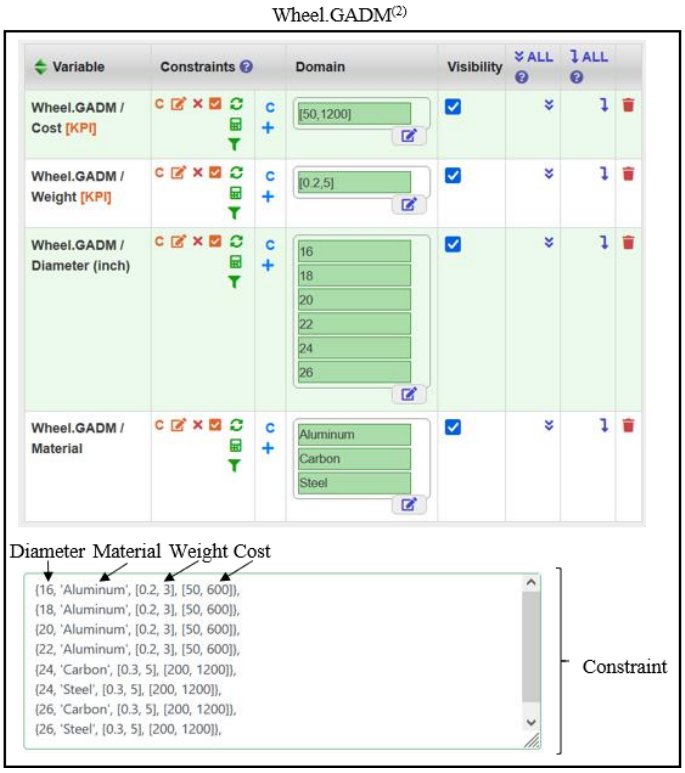

Figure 61. *Wheel.GADM(2)*

Figure 62 illustrates the taxonomy of GADMs before and after generalization. In this taxonomy, it is represented that *System.GADM(1)* is defined as the most general GADM.

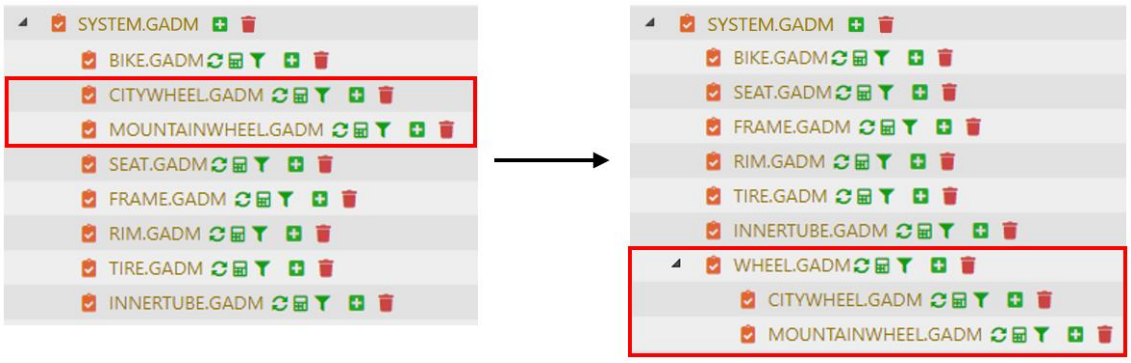

Figure 62. GADMs taxonomy before and after generalization

The generalization also implies the GASMs. Therefore, we define *Bike.GASM(2)* which is associated to *Bike.GA(2)* and *Bike.GADM(2)*. It is the generalization of *CityWheel.GASM(2)1*and *MountainWheel.GASM(2)1*which have to be firstly created.

We consider Figure 25 to define *CityWheel.GASM(2)1*and *MountainWheel.GASM(2)1*. We create *MountainWheel.GASM(2)1* as illustrated in Figure 63. First, we define its description. Then, we add *Rim.GA(2)* and *Tire.GA(2)* (they have to be created first). Since the quantity of both GAs are '1', we can define them directly in OPERA. Moreover, we define two constraints (numerical functions) in order to represent the equality between the diameter of the MountainWheel, the diameter of the Rim and the diameter of the Tire. We also define two constraints (numerical functions) to aggregate Weights and Costs.

Similarly, we create *CityWheel.GASM*$^{(2)}_1$ as represented in Figure 64. To do so, we add *Rim.GA*$^{(2)}$, *Tire.GA*$^{(2)}$, *InnerTube.GA*$^{(2)}$ with quantities '1'. We define a constraint (compatibility table) to represent the quantity of inner tube must be '1' in a city wheel. We define three numerical functions as well. The first one shows that the diameters of the city wheel and rim are equal, the second one represents the diameters of the city wheel and tire are equal and the last one shows that the diameters of the city wheel and inner tube are equal. Moreover, we define two numerical functions to aggregate weights and costs (respectively the sum of the weights and costs of *Rim.GADM*$^{(2)}$ and *Tire.GADM*$^{(2)}$).

For both *MountainWheel.GASM*$^{(2)}_1$and *CityWheel.GASM*$^{(2)}_1$, constraint filtering is applied in order to obtain consistent generic models. The domains of the variables of Figure 59 and Figure 60 are the ones obtained after filtering.

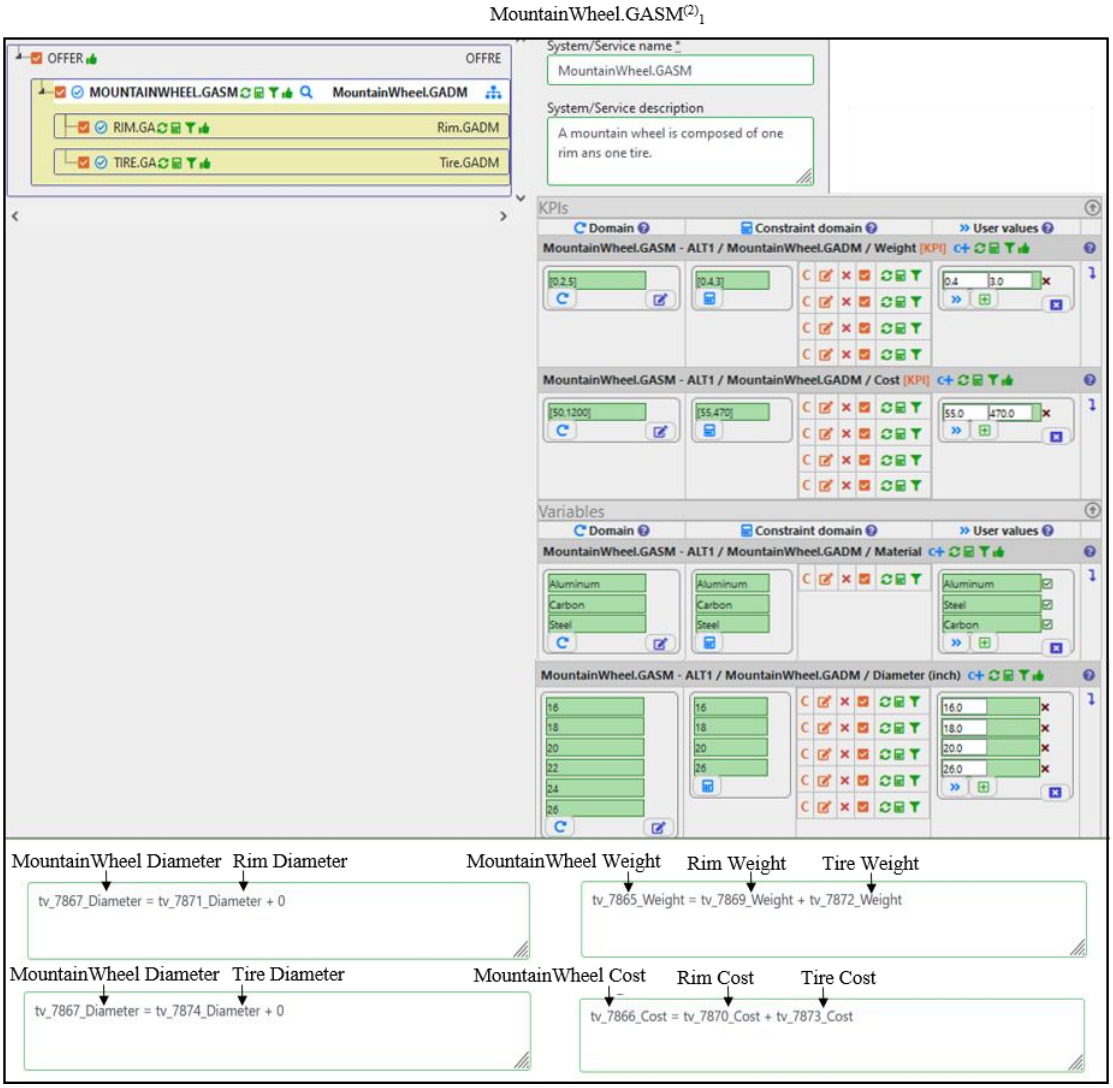

Figure 63. *MountainWheel.GASM(2)1*

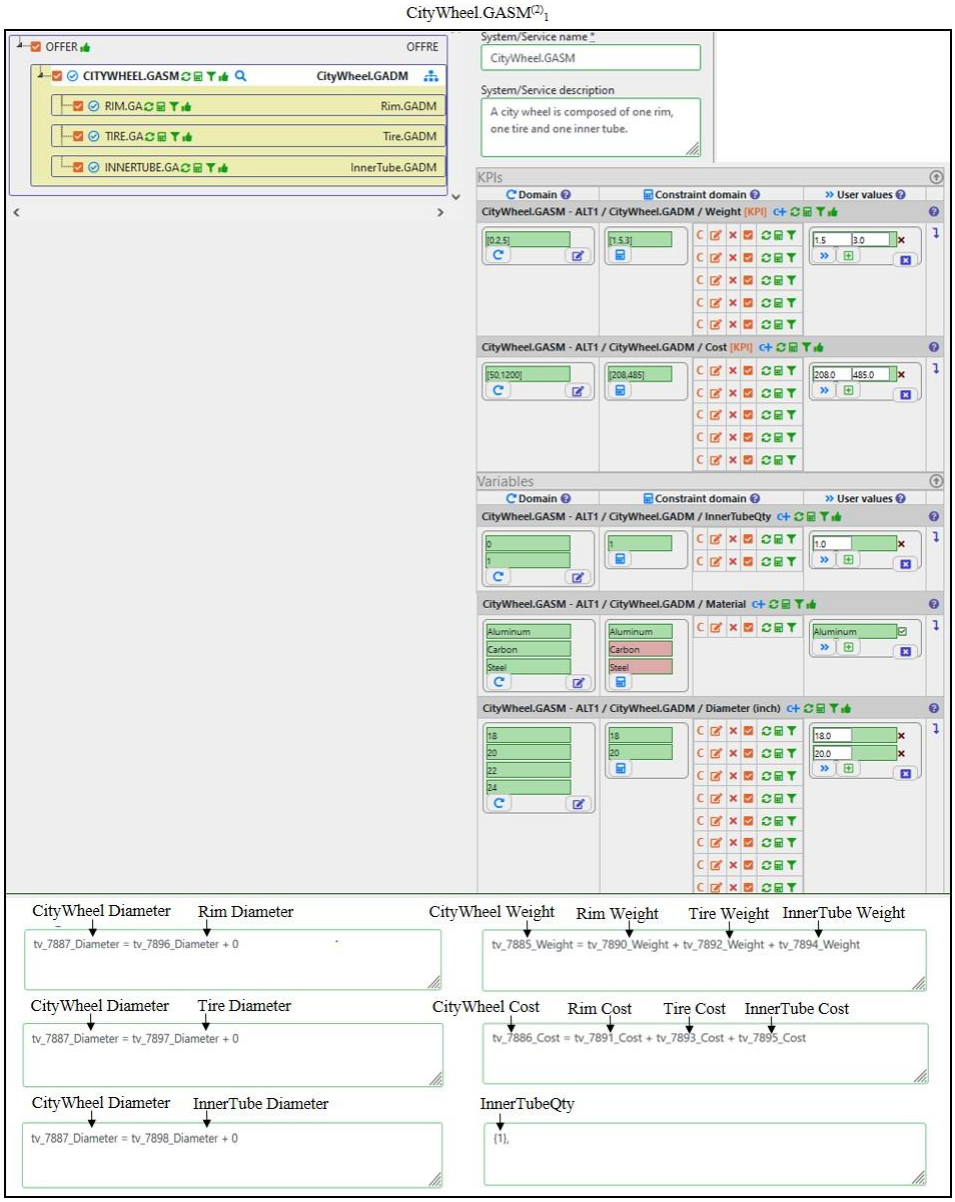

Figure 64. *CityWheel.GASM*$^{(2)}_1$

We take the example provided in Figure 26, to define *Wheel.GASM*$^{(2)}_1$using the generalization principle. Based on the commonality of *MountainWheel.GASM*$^{(2)}_1$ and *CityWheel.GASM*$^{(2)}_1$, we therefore define *Wheel.GASM*$^{(2)}_1$ as represented in Figure 65. We made it up of one *Rim.GA*$^{(2)}$, and one *Tire.GA*$^{(2)}$. We define two constraints (numerical functions) to indicate that: 1) the diameter of the wheel and rim are equal and 2) the diameter of the wheel and tire are equal. Additionally, we define two numerical functions for the aggregation of weights and costs. Finally, *MountainWheel.GASM*$^{(2)}_1$ and *CityWheel.GASM*$^{(2)}_1$are moved to the lower level (they become *MountainWheel.GASM*$^{(3)}_1$ and *CityWheel.GASM*$^{(3)}_1$.

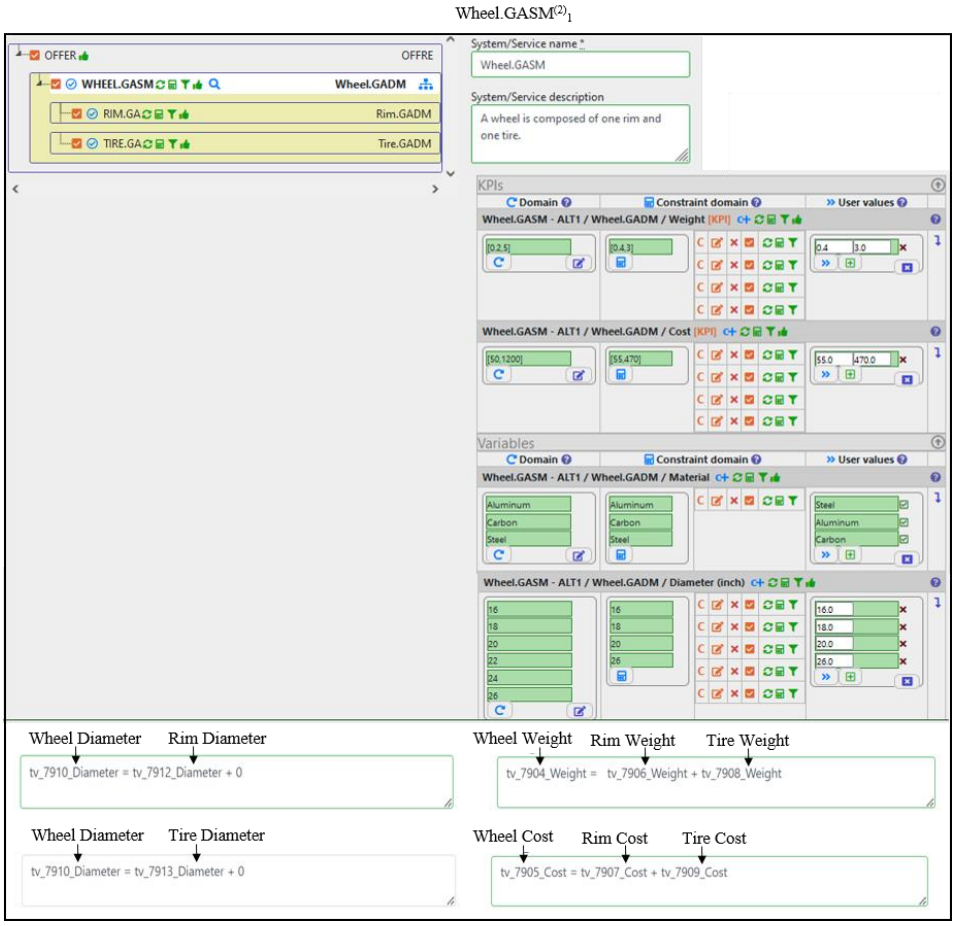

Figure 65. *Wheel.GASM*$^{(2)}{}_1$

5.2.4. GA Specialization

We take the example of Figure 23, which represents the specialization of *Bike.GADM*$^{(2)}$ into *CityBike.GADM*$^{(3)}$. We therefore define *CityBike.GADM*$^{(3)}$ as illustrated in Figure 66. To do so, the knowledge of *Bike.GADM*$^{(2)}$ is inherited by *CityBike.GADM*$^{(3)}$. In OPERA, the attributes, KPIs, and their domains are inherited however the constraints must be added manually. *CityBike.GADM*$^{(3)}$ is now characterized by three inherited attributes (Color, User and RingBellQty), two inherited KPIs (Weight and Cost), and two inherited constraints (the first and second compatibility tables represented in Figure 66). Then, we narrow the inherited knowledge. In this way, we remove the value 'Pink' from the domain of attribute 'Color'. We restrict the first constraint by adding a new tuple indicating that the user 'Man', the color 'Blue', the weight '30', and the cost '4000' are compatible. Then, we enrich the knowledge of *CityBike.GADM*$^{(3)}$. To do that, we add 'LightQty' with the domain {[1, 2]}. Moreover, we define a new constraint (compatibility table). In this table, the values respectively correspond to the compatible values of 'User' and 'LightQty'. After filtering these constraints, the domains of variables are restricted as shown in Figure 66.

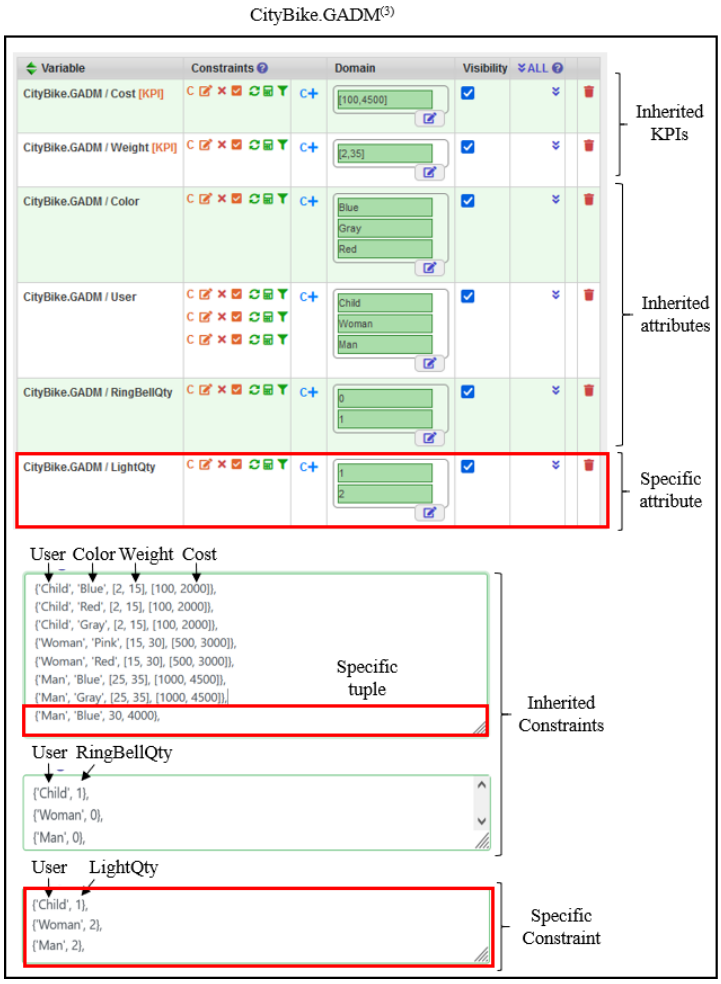

Figure 66. *CityBike.GADM(3)*

Figure 67 illustrates the taxonomy of GADMs before and after specialization. It illustrates that *Bike.GADM(2)* is specialized into *CityBike.GADM(3)* and *MountainBike.GADM(3)*. Note that in this section only the illustration of the creation of *CityBike.GADM(3)* is shown.

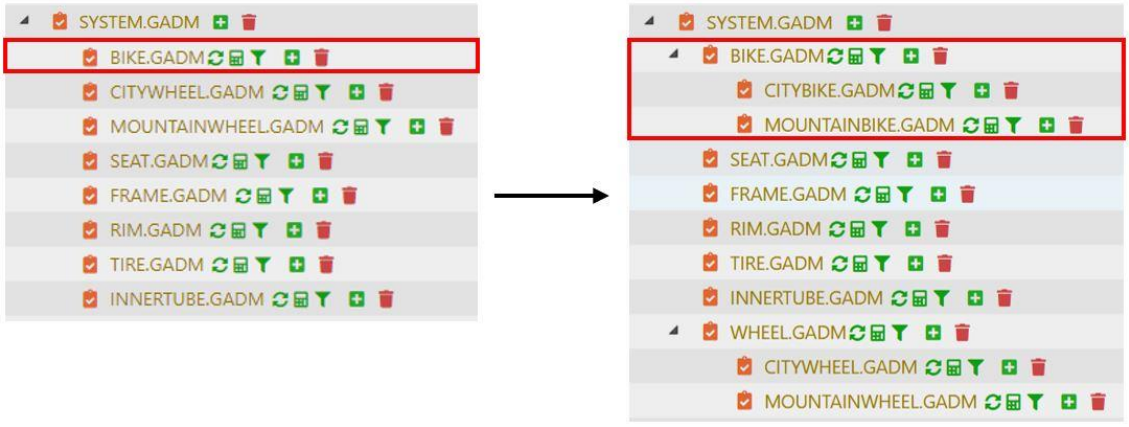

Figure 67. GADMs taxonomy before and after specialization

We use the example of Figure 28, in which *Bike.GASM(2)1* is specialized into *CityBike.GASM(3)1*. Then, we define *CityBike.GASM(3)1* as illustrated in Figure 68. To do so, first, we define its description then we associate *CityBike.GASM(3)1* to *CityBike.GADM(3)*. We also define

Wheel.GA$^{(2)}$, *Seat.GA*$^{(2)}$ and *Light.GA*$^{(2)}$ as its composing GAs. We associate each GA to its corresponding GADM. We define two variables corresponding to the quantity of *Wheel.GA*$^{(2)}$ and quantity of *Light.GA*$^{(2)}$. We define a constraint (compatibility table) to restrict the domain of variable 'WheelQty' to '2'. We add 'LightQty' with its domain. We add a new tuple to a compatibility table. The first value corresponds to the value of User' from *CityBike.GADM*$^{(2)}$ and the second one corresponds to the value of 'Material' (from *Seat.GADM*$^{(2)}$). Additionally, we define a new compatibility table, in which the first value corresponds to the value of 'User' (from *CityBike.GADM*$^{(2)}$) and the second one corresponds to the value of 'Color' (from *Light.GADM*$^{(2)}$). We also define two numerical functions to aggregate weights and costs. Finally, we apply constraint filtering, which narrows domains and removes inconsistent values.

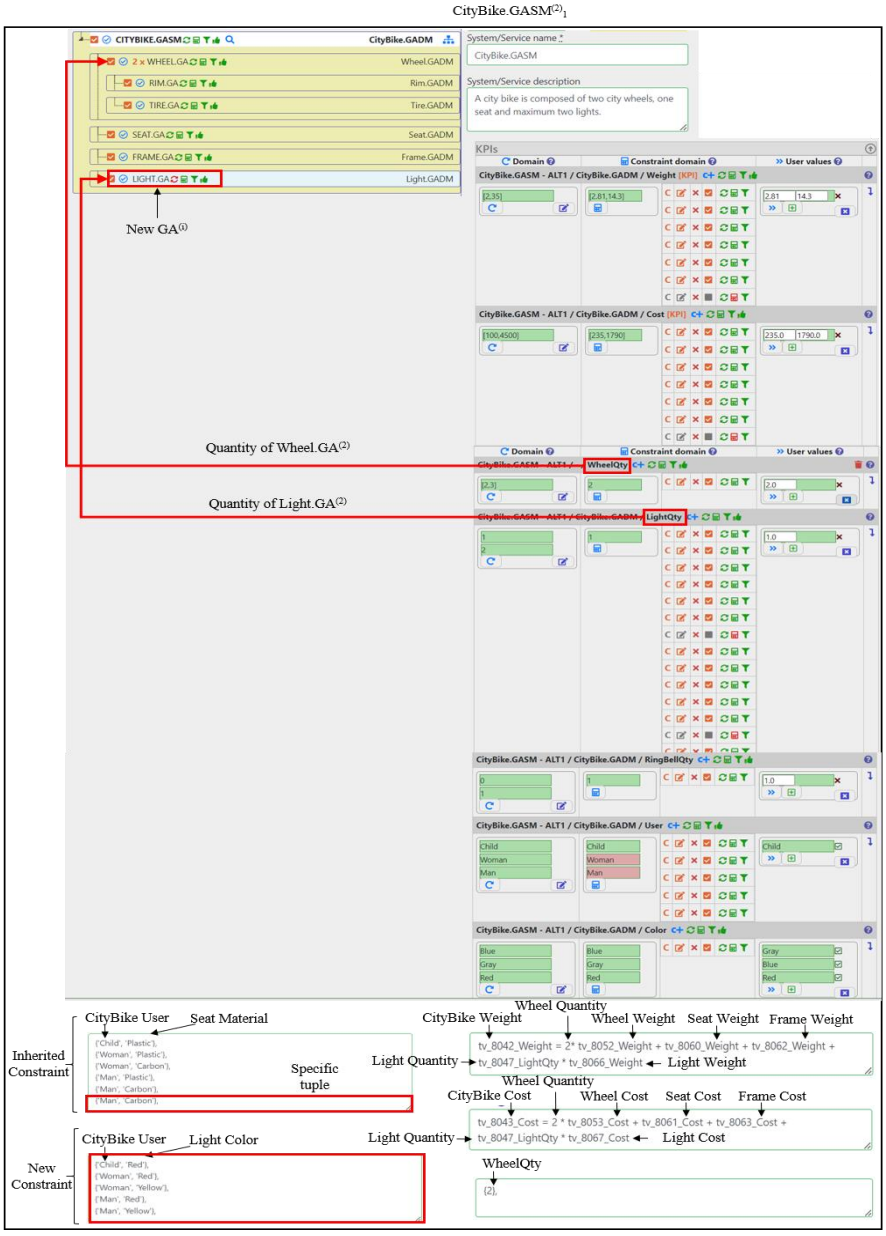

Figure 68. *CityBike.GASM*$^{(3)}_1$

In this second section, we have shown that it is possible to formalize knowledge for system configuration using the OPERA software. All the generic models we obtained are stored in

taxonomies and they constitute an ontology of generic models. All the elements are consistent pieces of knowledge. Then, it is necessary to verify that this knowledge can be reused and the following section is dedicated to the knowledge reuse for system configuration.

5.3. Knowledge reuse for system configuration

This section is dedicated to the illustration of our knowledge reuse process (explained in chapter 4) through examples of bicycle configuration. In section 5.3.1 and 5.3.2, knowledge reuse in CTO situation is shown. The creation and configuration of a solution using the descriptive view *CTO.Bike.GADI*$^{(2)}$ is presented in section 5.3.1. In section 5.3.2, the exploitation of the structural view *MountainWheel.GASM*$^{(3)}_{1}$ (first version of the structural view of Mountain.Wheel) in CTO situation is described. In section 5.3.3, 5.3.4 and 5.3.5, knowledge reuse in ETO situation is presented. First, in section 5.3.3, the creation of a new mirror (*ETO.Mirror.GAI*$^{(i)}$ and *ETO.Mirror.GADI*$^{(i)}$) is presented. In section 5.3.4, the modification of the descriptive view *ETO.Wheel.GADI*$^{(2)}$ is described and, finally the modification of the structural view *ETO.Bike.GASI*$^{(2)}_{1}$ is represented in section 5.3.5.

5.3.1. System configuration using the descriptive view

We refer to Figure 41, which represents the generic process for knowledge reuse in CTO situation, to configure the descriptive view of a bike instance by following a few steps:

- Step 1: Select *Bike.GA*$^{(2)}$: In this use case, the user wants to configure a bike. Therefore, *Bike.GA*, representing a family of bikes, is selected from the GAs taxonomy. We, therefore, can use its associated GADM from the GADMs taxonomy.
- Step 2: Create instances of *Bike.GA*$^{(2)}$ and *Bike.GADM*$^{(2)}$: We create *CTO.Bike.GAI*$^{(2)}$ and *CTO.Bike.GADI*$^{(2)}$ as illustrated in Figure 69.

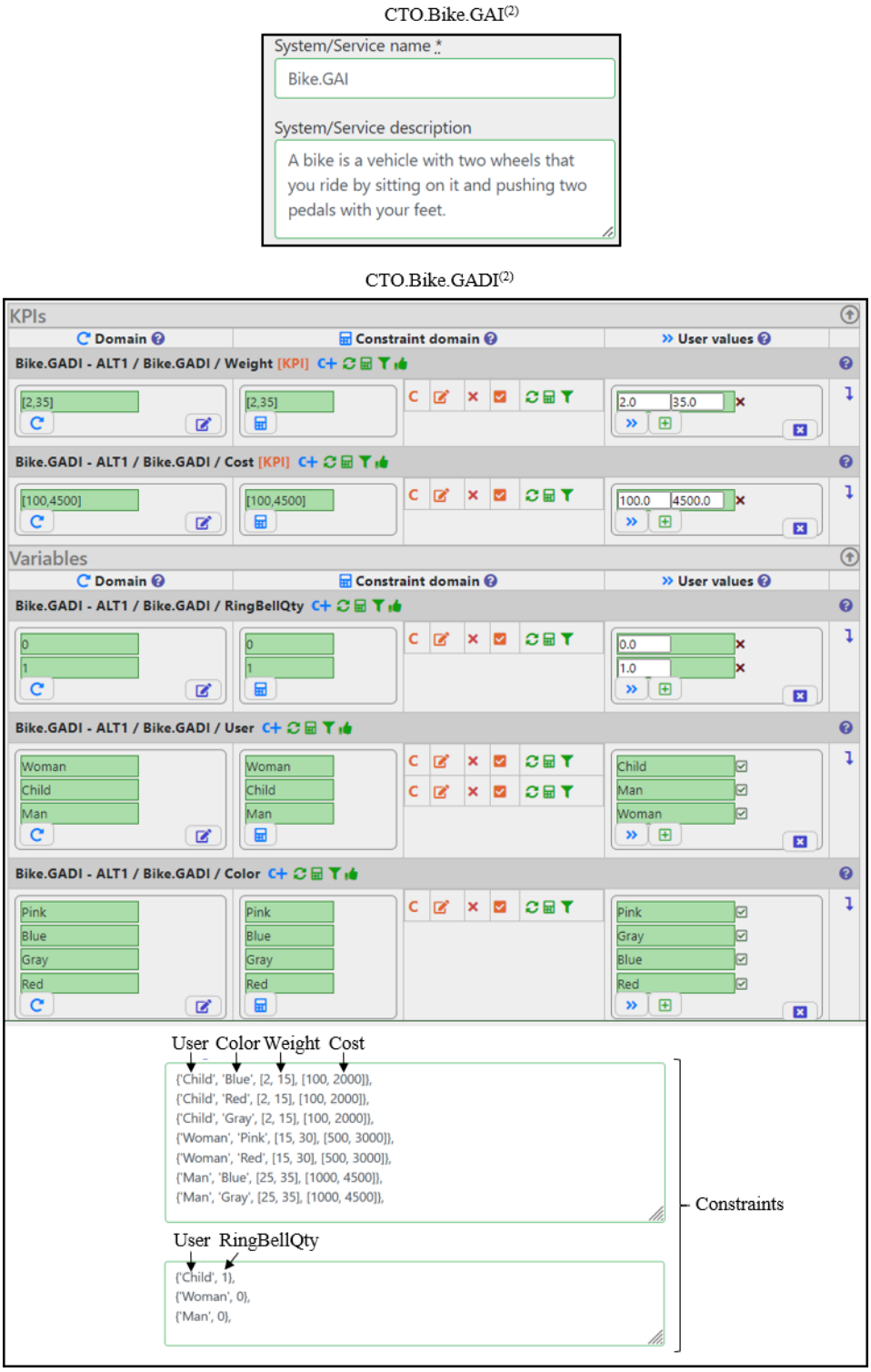

Figure 69. *CTO.Bike.GADI(2)*

- Step 3: User defines a requirement: The user defines that the bike is for a woman by restricting the domain of the “User” variable.

- Step 4: Apply constraint filtering: the OPERA software filters the values of attributes and KPIs based on the user's requirement. For example, the defined CTO requirement (User = {Woman}), impacts the domains of one attribute and two KPIs (Color, Weight and Cost). Therefore, after propagation, the filtered domain of these variables are as

follows: Color = {Pink, Red}, RingBellQty = {0}, Weight = {[15, 30]} and Cost {[500, 3000]} as represented in Figure 70.

Since no filtered domains are empty and the domain of all filtered domains of attributes are not singletons, allowed values (represented in green boxes) are proposed to the user.

- Step 5: User defines a second requirement: the color of the bike has to be pink.
- Step 6: Apply constraint filtering: in the OPERA software, constraint filtering is applied. In this case, this doesn't impact the domains of Weight and Cost. Therefore, Weight = {[15, 30]} and Cost = {[500, 3000]} remain as illustrated in Figure 70. There is no empty domain and all the values of attributes are singletons therefore a solution can be proposed.
- Step 7: Propose a CTO solution: based on the user requirements, a CTO solution is obtained in *CTO.Bike.GADI*$^{(2)}$ which is illustrated in Figure 70. In this solution the user is 'Woman', the color is 'Pink', the quantity of ringbell is '0', the weight is {[15, 30]} and the cost is {[500, 3000]}. In this solution, the domains of attributes are singleton while the domains of KPIs are ranges due to the uncertainty on the values of the Cost and Weight.
- Step 8: Decide on using the structural view if exists: the user decides whether a structural view (detailed configuration of components and sub-components) is necessary. In this example, the user is not interested in configuring the structural view. Therefore, a CTO solution is proposed for *CTO.Bike.GADI*$^{(2)}$: User = {Woman}, Color = {Pink}, RingBellQty = {0}, Weight = {[15, 30]} and Cost {[500, 3000]}.

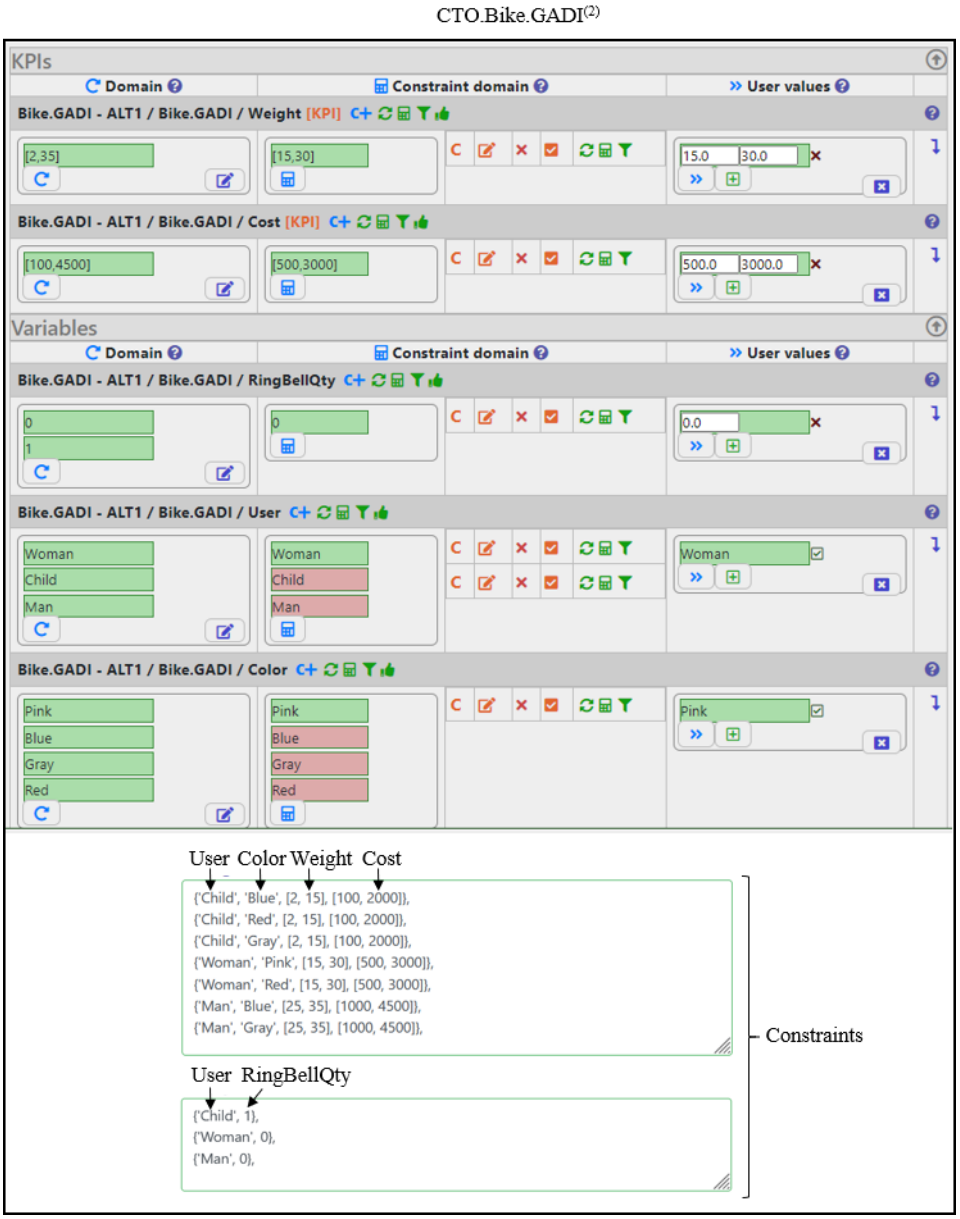

Figure 70. CTO solution of *CTO.Bike.GADI*$^{(2)}$

5.3.2. System configuration using the structural view

As explained in section 4.1.1, it is possible to configure the structural view of a GAI during a CTO configuration activity. We use the flowchart for knowledge reuse in CTO configuration which is shown in Figure 42. In this case, the configuration of a MountainWheel is performed using its structural view.

- Step 1: Configure the descriptive view of *CTO.MountainWheel.GAI*$^{(3)}$: first, the descriptive view of *CTO.MountainWheel.GAI*$^{(3)}$ (i.e. *CTO.MountainWheel.GADI*$^{(3)}$) is configured and then a CTO solution is proposed (Figure 71): the diameter is '20', the material is 'Aluminum', the weight is {[0.2, 3]} and the cost is {[50, 600]}.

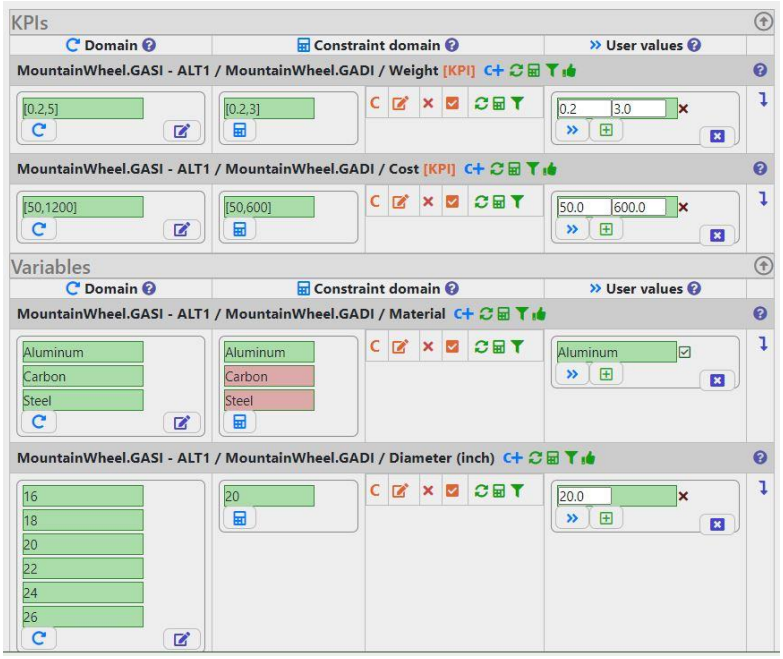

Figure 71. CTO solution of *CTO.MountainWheel.GADI*$^{(3)}$

- Step 2: Decide on using the structural view if it exists: in this example, the user is interested to configure the structural view of *CTO.MountainWheel.GAI*$^{(3)}$.
- Step 3: Select one version of *MountainWheel.GASM*$^{(3)}$ among *MountainWheel.GASM*$^{(3)}{}_1$ and *MountainWheel.GASM*$^{(3)}{}_2$, on the basis of their descriptions. In this example, the first version *MountainWheel.GASM*$^{(3)}{}_1$ is chosen.
- Step 4: Create an instance of *MountainWheel.GASM*$^{(3)}{}_1$: We create *CTO.MountainWheel.GASI*$^{(3)}{}_1$ as shown in Figure 72.

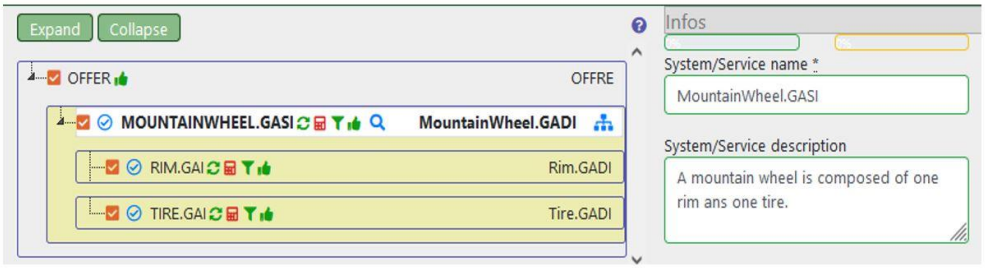

Figure 72. *CTO.MountainWheel.GASI*$^{(3)}{}_1$

- Step 5: Define the quantity of all GAIs: since *CTO.MountainWheel.GASI*$^{(3)}{}_1$ is composed of one *CTO.Rim.GAI*$^{(2)}$ and one *CTO.Tire.GAI*$^{(2)}$, their quantities are defined as follows: RimQty = {1}, TireQty = {1}
- Step 6: Select *CTO.Tire.GAI*$^{(2)}$ and configure it: to configure *CTO.Tire.GAI*$^{(2)}$, its descriptive view is configured (similar to section 5.3.1). At the end of the configuration, as shown in Figure 73, a CTO solution is found for *CTO.Tire.GADI*$^{(2)}$: Diameter = {20}, Cost = {[48, 70]}, Weight = {[0.7, 1.5]}.

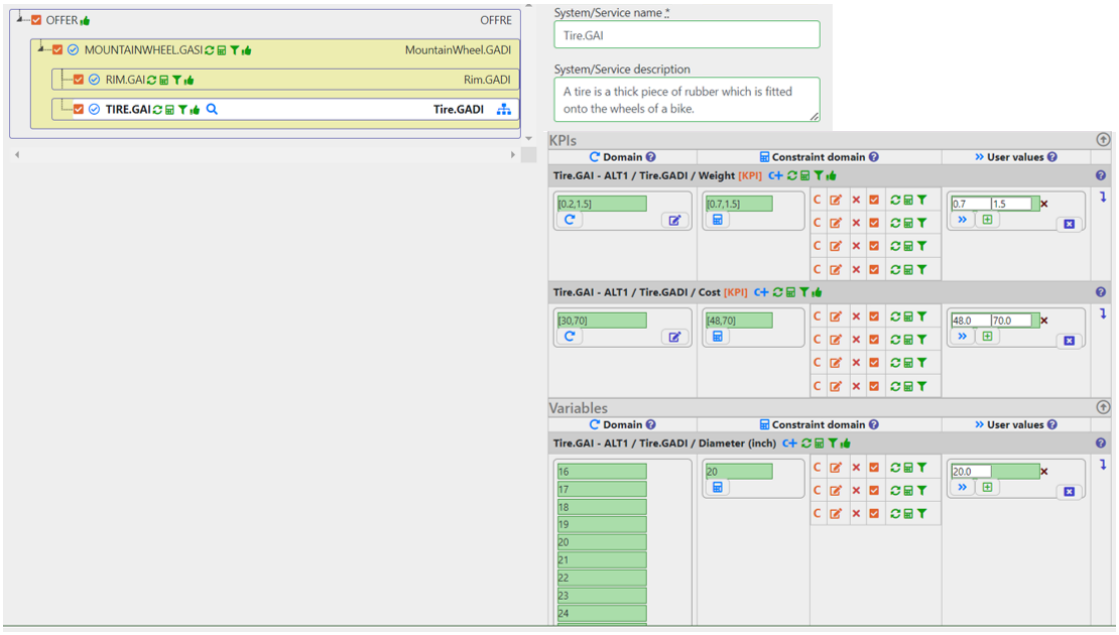

Figure 73. Solution for *CTO.Tire.GADI*$^{(2)}$

- Step 7: Select *Rim.GAI*$^{(2)}$ and configure it: at the end of configuration, a CTO solution is found: Diameter = {20}, Cost = {[48, 70]}, Weight = {[0.7, 1.5]}.

  There are no more GAIs and all the configured GAIs have a solution.

- Step 8: Aggregate the KPIs for *CTO.MountainWheel.GASI*$^{(3)}{}_1$: in OPERA, after filtering constraints, the calculated Cost and Weight are obtained as follows (Figure 74): Weight = {[1.3, 3]} and Cost = {[298, 470]}.

- Step 9: Propose a CTO solution: finally, the set of all solutions for the *CTO.Rim.GAI*$^{(2)}$ and *CTO.Tire.GAI*$^{(2)}$ construct the CTO solution for *CTO.MountainWheel.GASI*$^{(3)}{}_1$. The solution for the structural view *CTO.MountainWheel.GASI*$^{(3)}{}_1$ is: the *CTO.MountainWheel.GASI*$^{(3)}{}_1$ is composed of one *CTO.Rim.GAI*$^{(2)}$ and one *CTO.Tire.GAI*$^{(2)}$.

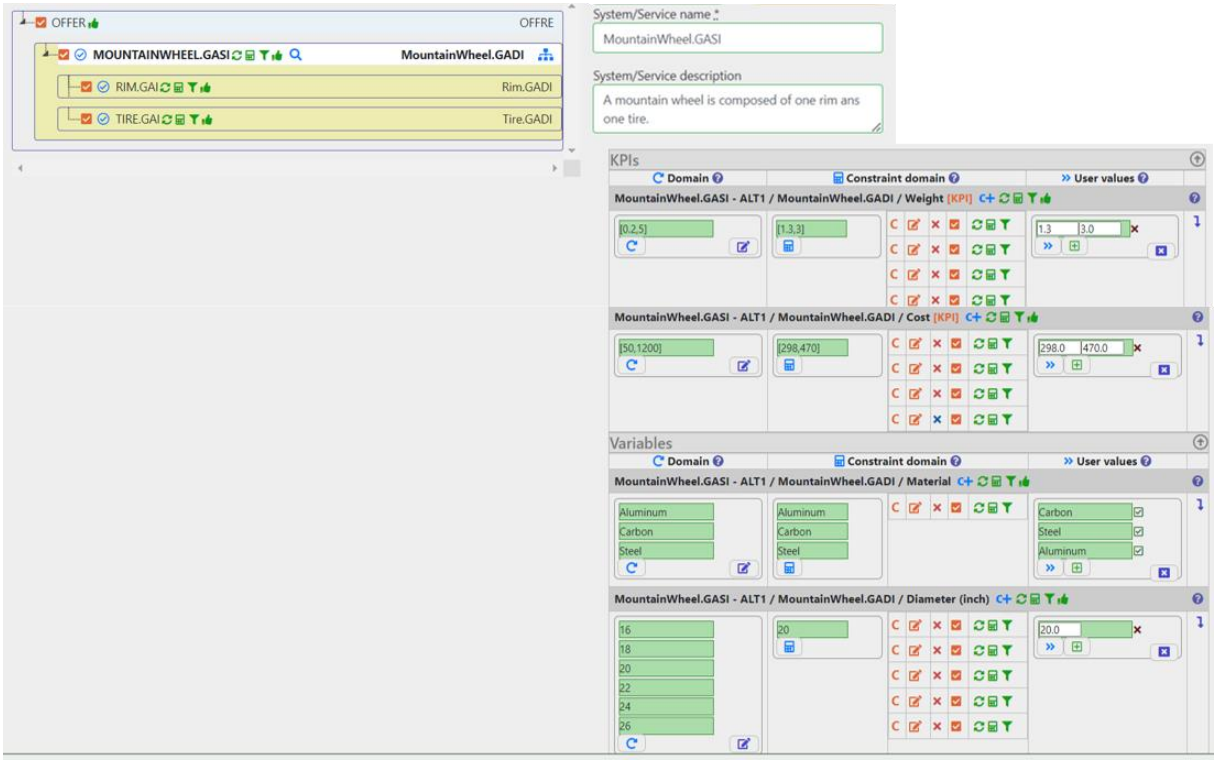

Figure 74. Solution of *CTO.MountainWheel.GASI*$^{(3)}{}_{1}$

5.3.3. GAI and GADI building

As explained in section 4.2.2, during the ETO configuration activity, the user may need a GA that does not belong to the GAs taxonomy. In our example, the user needs a mirror that isn't available in the taxonomy. Therefore, to satisfy this user requirement, we need to build a new instance of the mirror family in the OPERA software (i.e. *ETO.Mirror.GAI*$^{(i)}$), illustrated in Figure 75. Note that we define the level (i) as we don't know yet if this generic artifact will be added to the GAs taxonomy later nor its hierarchical level. We create its associated GADI represented in Figure 45. To meet ETO requirements, *ETO.Mirror.GADI* is built as shown in Figure 76. We define the attribute Diameter with domain {[10, 80]}, the KPI Weight with domain {[0.2, 0.9]} and the KPI Cost with domain {[30, 70]}. For this *ETO.Mirror.GAI*, the user decides that there is no need to create a structural view.

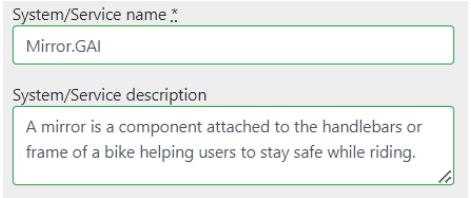

Figure 75. *ETO.Mirror.GAI*$^{(i)}$

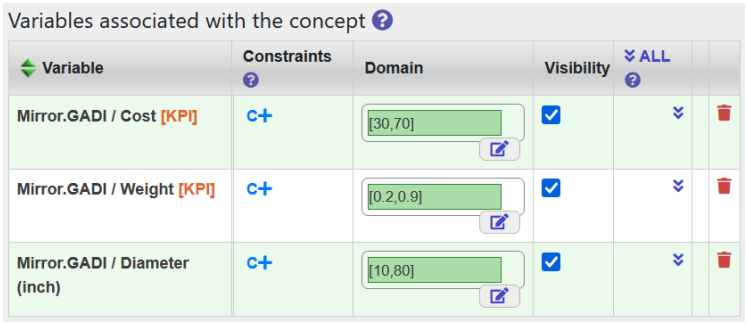

Figure 76. *ETO.Mirror.GADI*$^{(i)}$

5.3.4. GADI modification

As explained in section 4.2.1, ETO requirements can be satisfied during ETO configuration activity by modifying an existing GADI (i.e. an instance of an existing GADM which belongs to the GADMS taxonomy). The generic process for reusing knowledge in ETO (for GADI configuration) is illustrated in Figure 43 by means of a flowchart.

- Step 1: Select an existing GA in the taxonomy: the user selects *Wheel.GA*$^{(2)}$ and therefore, the configuration will be based on an instance of it.
- Step 2: Create an instance of *Wheel.GA*$^{(2)}$ and *Wheel.GADM*$^{(2)}$: as illustrated in Figure 77, *ETO.Wheel.GAI*$^{(2)}$ and *ETO.Wheel.GADI*$^{(2)}$ are created.

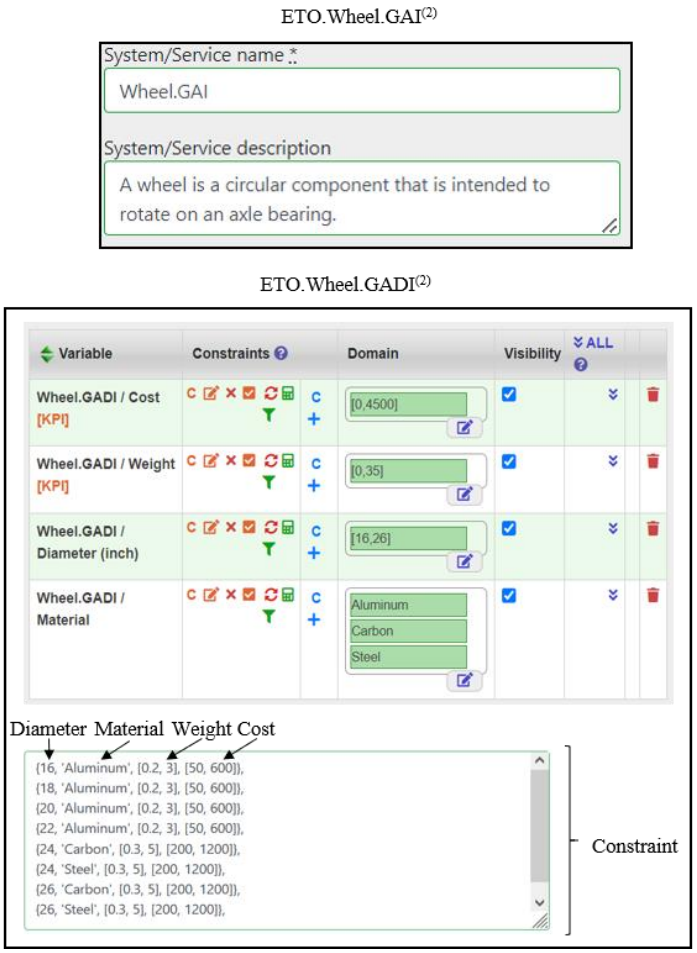

Figure 77. *ETO.Wheel.GAI*$^{(2)}$ and *ETO.Wheel.GADI*$^{(2)}$

- Step 3: User defines ETO requirements: the user has two requirements: 1) she/he wants a wheel with 28 spokes, 2) she/he wants to define a relation between the quantity of spokes and the diameter of wheel. The wheel diameters of '20', '24' and '26' are only compatible with quantities '16', '18', '20' and '22'. The diameter of '28' is only compatible with the quantities '24' and '26'.

  Step 4: Modify *ETO.Wheel.GADI*$^{(2)}$: to fulfill these ETO requirements, *ETO.Wheel.GADI*$^{(2)}$ must be modified. We, therefore, use the example of Figure 50. However, here, we only define two cases (3 and 4) which are respectively related to defining a new attribute with its domain and defining a new relation. As represented in Figure 78, we modify *ETO.Wheel.GADI*$^{(2)}$. To do that, we add the attribute SpokeQty with the domain {28}, and then we define a new compatibility table. The first value corresponds to the value of the attribute SpokeQty and the second one corresponds to the value of Diameter. As we do not apply constraint filtering, the domains of attributes and KPIs are not restricted during this step.

Figure 78. Modified *ETO.Wheel.GADI*$^{(2)}$

- Step 5: Manually configure *ETO.Wheel.GADI*$^{(2)}$: since the quantity of spoke is '28', due to the new constraint the diameter can be '24' or '26'. The user selects '26'. Then, due to the first constraint, the diameter '26' is only compatible with the material 'Steel' or 'Carbon'. The user selects 'Carbon'. Then, the domains of Weight and Cost have to be manually restricted to Weight = {[0.3, 5]}, Cost = {[200, 1200]} following the constraints of *ETO.Wheel.GADI*$^{(2)}$. The domain of all attributes are then reduced to singletons.

- Step 6: Propose an ETO solution: the proposed ETO solution is illustrated in Figure 79 where the quantity of spokes is '28', the diameter is '26', the material is 'Carbon', the weight is {[0.3, 5]}, and the cost is {[290, 1200]}. It should be noticed that the solution obtained here differs from the ones presented in chapter 4, because we are focusing on just two specific cases.

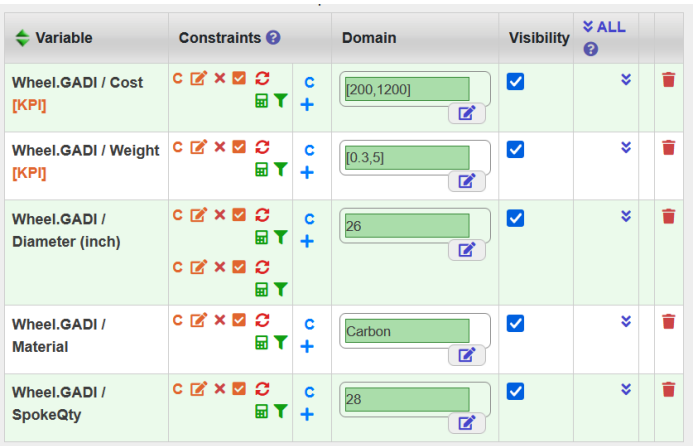

Figure 79. ETO solution of modified *ETO.Wheel.GADI*$^{(2)}$

5.3.5. GASI modification

As mentioned in section 4.2.1, ETO requirements can be satisfied during ETO configuration activity by modifying GASI. We use the generic process for reusing knowledge in ETO shown in Figure 44.

- Step 1: *Bike.GA*$^{(2)}$ is selected and then *ETO.Bike.GAI*$^{(2)}$ and *ETO.Bike.GADI*$^{(2)}$ are created. Then, the descriptive view of *ETO.Bike.GAI*$^{(2)}$ is manually configured, interactively with the user as represented on Figure 80. The user wants a bike for a 'Child'. The first constraint within *ETO.Bike.GADI*$^{(2)}$ represents that the user 'Child' is only compatible with the colors 'Blue', 'Red' and 'Gray' which the user selects 'Blue'. Then, the user manually restricts the domains of weight and cost to {[2, 15]} and {[100, 2000]} respectively. The second constraint within *ETO.Bike.GADI*$^{(2)}$ represents that for a user 'Child', the quantity of RingBell is '1'. The domains of attributes are singletons we, therefore, propose an ETO solution based only on the descriptive view.

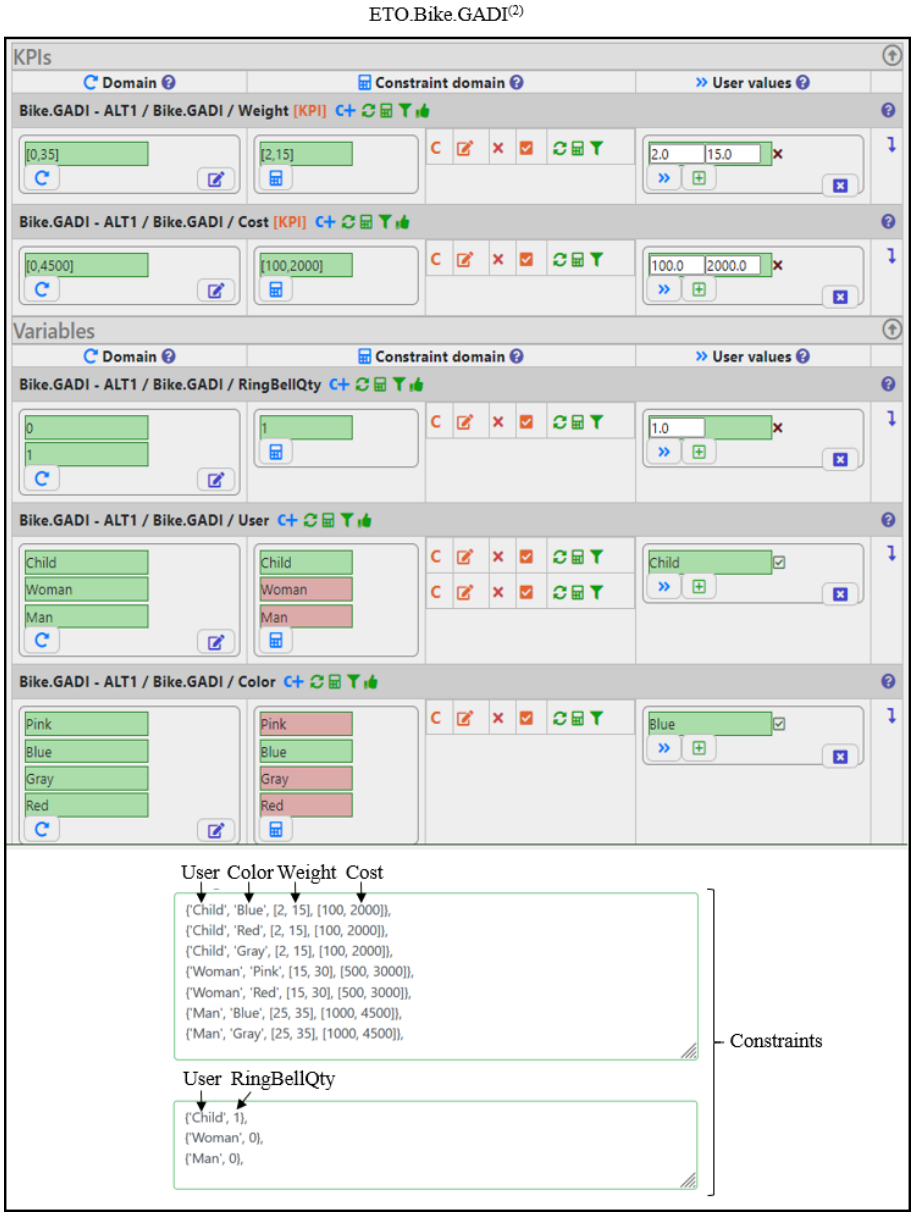

Figure 80. Solution of *ETO.Bike.GADI*$^{(2)}$

- Step 2: As a structural view is available for *Bike.GA*$^{(2)}$, the user wants to configure a structural view of *ETO.Bike.GAI*$^{(2)}$.
- Step 3: The user selects *Bike.GASM* $^{(2)}{}_1$, the first version of *the structural view of Bike.GA*$^{(2)}$ in the taxonomy of GASMs.
- Step 4: Create an instance of *Bike.GASM* $^{(2)}{}_1$: we define *ETO.Bike.GASI* $^{(2)}{}_1$ as shown in Figure 81.

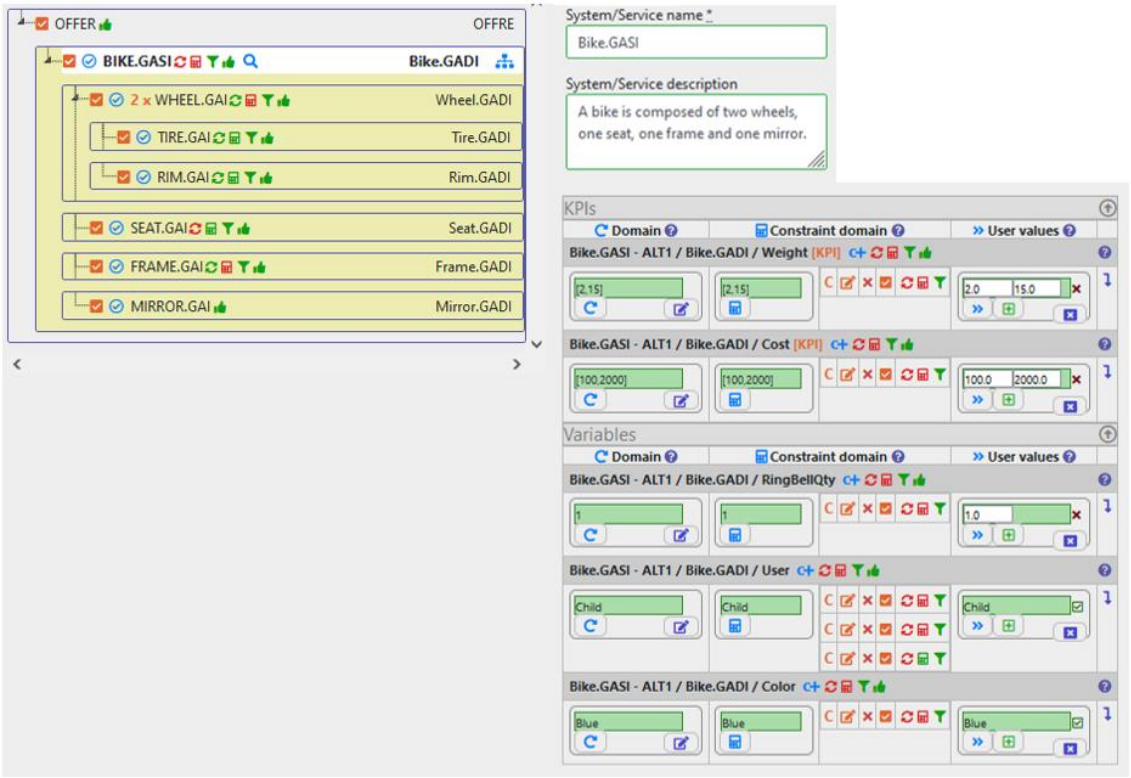

Figure 81. *ETO.Bike.GASI* $^{(2)}_{1}$

- Step 5: Modify *ETO.Bike.GASI* $^{(2)}_{1}$: we take the example of Figure 51 in chapter 4. However, we only consider cases 3 and 5. We modify *ETO.Bike.GASI* $^{(2)}_{1}$ as represented in Figure 82. Since the user wants a bike with a mirror, we add the *ETO.Mirror.GAI* with quantity '1' in *ETO.Bike.GASI* $^{(2)}_{1}$. It is necessary to add the attribute 'MirrorQty' in *ETO.Bike.GASI* $^{(2)}_{1}$. Moreover, the user wants to define a relation between the quantity of mirrors and the quantity of seats. Therefore, we define a new table of compatibility in which the first value corresponds to the value of '*SeatQty'* and the second one corresponds to the value of '*MirrorQty'*.

Figure 82. Modified *ETO.Bike.GASI*$^{(2)}_1$

- Step 6: Define the quantity of all GAI: the user wants a bike with two wheels, one seat, one frame and one mirror. Note that if we have one mirror, we must have one seat due to the added compatibility table within *ETO.Bike.GASI* $^{(2)}_1$.

- Step 7: Select *CTO.Wheel.GAI*$^{(2)}$ and configure it: the user is interested in configuring both the descriptive view and structural view of *CTO.Wheel.GAI*$^{(2)}$ without any modifications (CTO configuration). At the end of the CTO configuration, the result of configuring *CTO.Wheel.GADI*$^{(2)}$ is (Figure 83): Diameter = {16}, Material = {Aluminum}, Weight = {[0.2, 3]}, Cost = {[50, 600]}. The result of configuring *CTO.Wheel.GASI*$^{(2)}_1$ is (Figure 83): *CTO.Wheel.GASI*$^{(2)}_1$ is composed of one *CTO.Rim.GAI*$^{(2)}$ and one *CTO.Tire.GAI*$^{(2)}$. There is a constraint representing that the diameters of wheel, rim and tire are equal. The *CTO.Rim.GAI*$^{(2)}$ is configured as follows: Diameter = {16}, Weight = {[0.2, 0.6]}, Cost = {[25, 200]}. The *CTO.Tire.GAI*$^{(2)}$ is configured as follows: Diameter = {16}, Weight = {[0.2, 0.8]}, Cost = {[30, 50]}.

The weight and cost of *CTO.Wheel.GADI*$^{(2)}$ are aggregated as follows:

Weight = {[0.4, 1.4]} and Cost = {[55, 250]}.

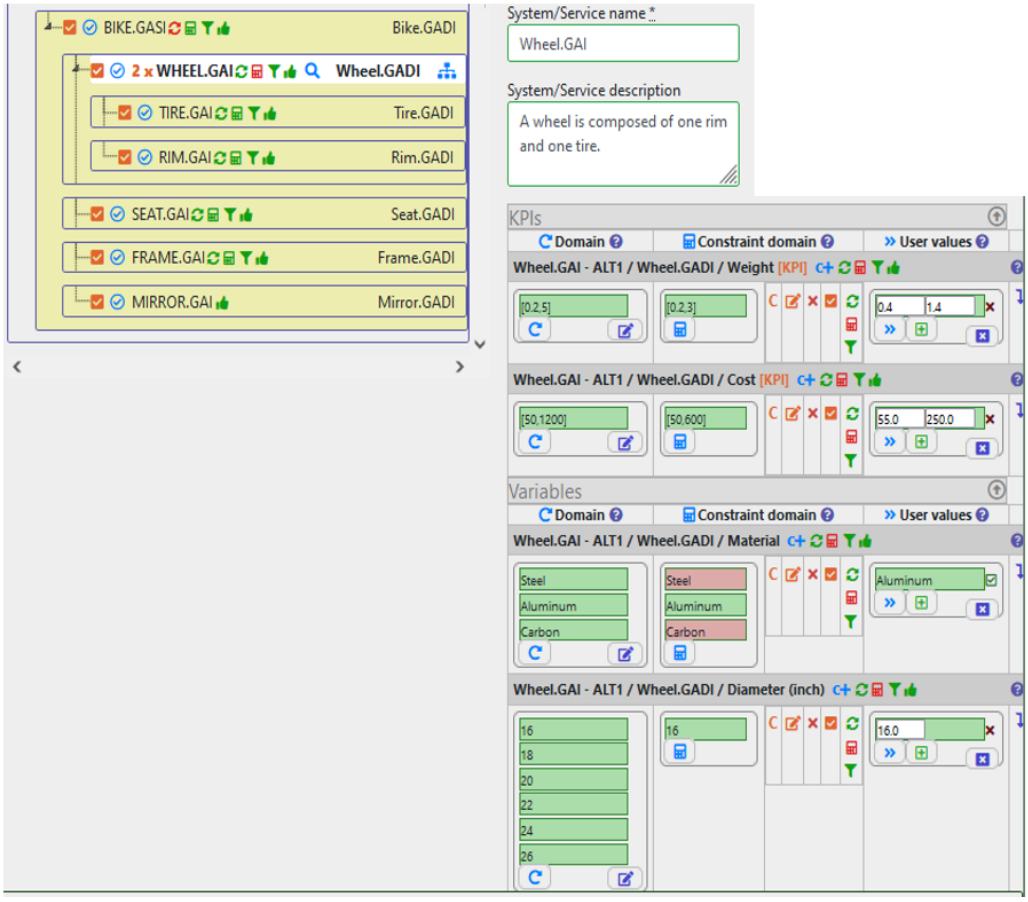

Figure 83. CTO solution for *CTO.Wheel.GASI*$^{(2)}$

- Step 8: Select *CTO.Frame.GAI*$^{(2)}$ and configure it without modifications (CTO configuration) (Figure 84): the user selects the color 'Blue' for the seat. Due to the compatibility table defined within *CTO.Frame.GADI*$^{(2)}$, the color 'Blue' is only compatible with materials 'Aluminum' and 'Carbon' which the user selects 'Carbon'. Then, the Weight and Cost are manually restricted to {[2, 7]} and {[300, 600]}.

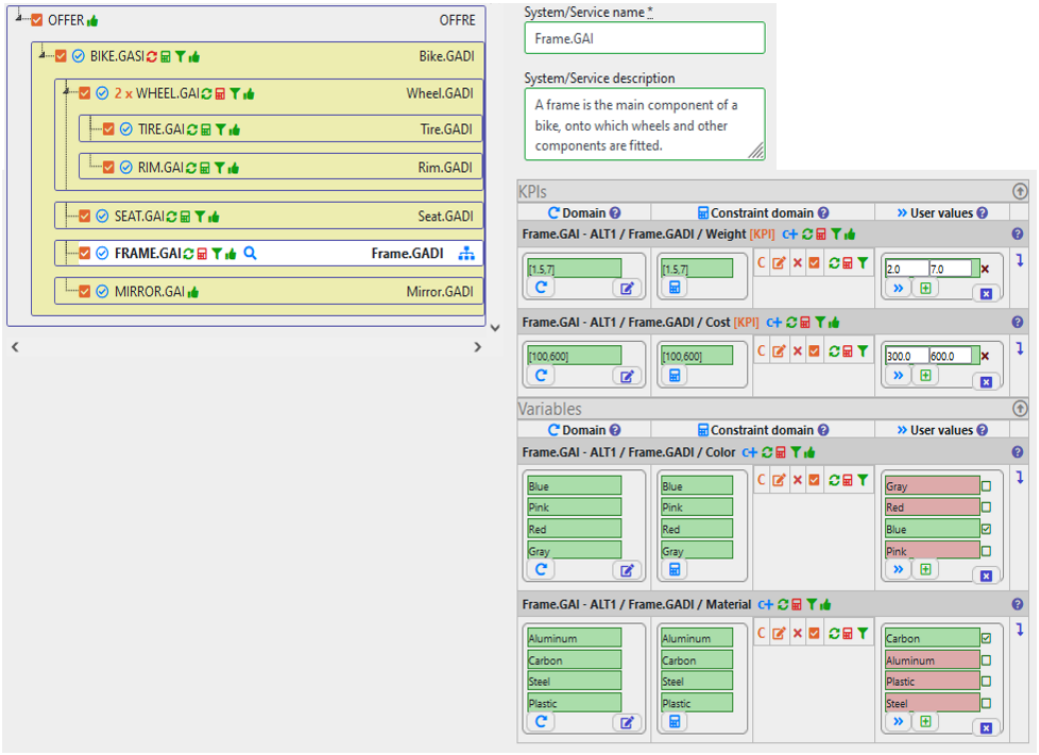

Figure 84. CTO solution for *CTO.Frame.GAI*$^{(2)}$

- Step 9: Select and configure *CTO.Seat.GAI*$^{(2)}$ (Figure 85): Since the user is 'Child', the material of *CTO.Seat.GADI*$^{(2)}$ is 'Plastic' due to the first constraint defined in *ETO.Bike.GASI* $^{(2)}{}_{1}$. Then, considering the compatibility table defined within *CTO.Seat.GADI*$^{(2)}$, material 'Plastic' is compatible with colors 'Pink', 'Blue' and 'Gray' which the user selects 'Pink'. Then, the weight and cost are manually restricted to {[0.5, 0.8]}, {[5, 100]}. Since the domain of all attributes has been reduced to singletons, we propose a CTO solution for *CTO.Seat.GADI*$^{(2)}$ in the Figure 85.

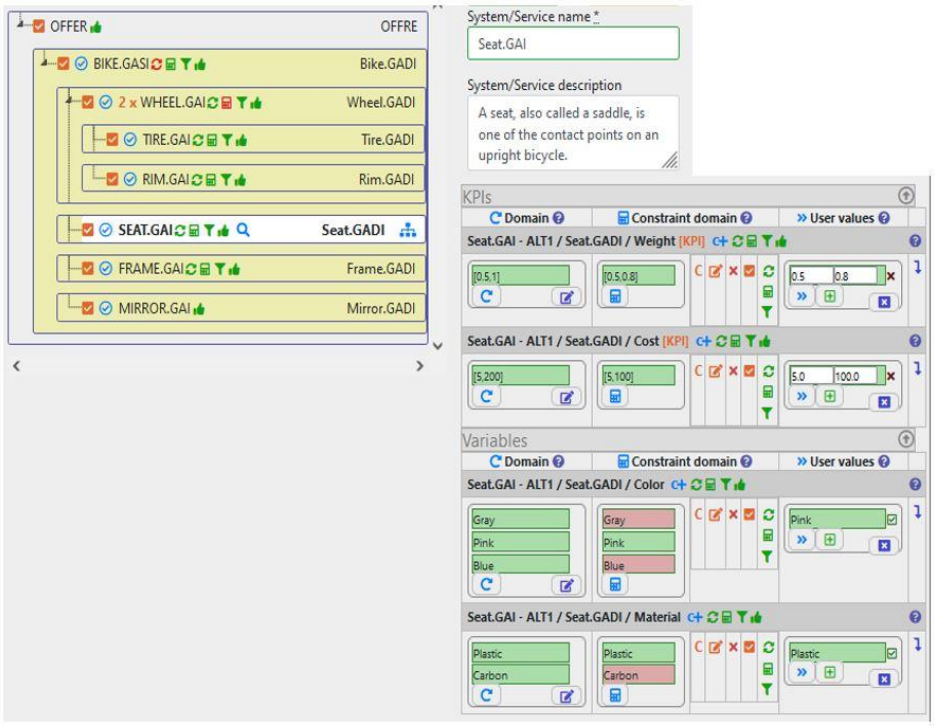

Figure 85. CTO solution for *CTO.Seat.GAI*[(2)]

- Step 10: Select and configure *ETO.Mirror.GAI* (Figure 86): as explained before *ETO.Mirror.GAI* and *ETO.Mirror.GADI* are created. The user decides to only configure the descriptive view of *ETO.Mirror.GAI*, not its structural view. The user selects the value '30' for the diameter of the mirror among the ranges of {[10, 80]}. The weight and cost remain {[0.2, 0.9]} and {[30, 70]}. This corresponds to an ETO solution for the mirror.

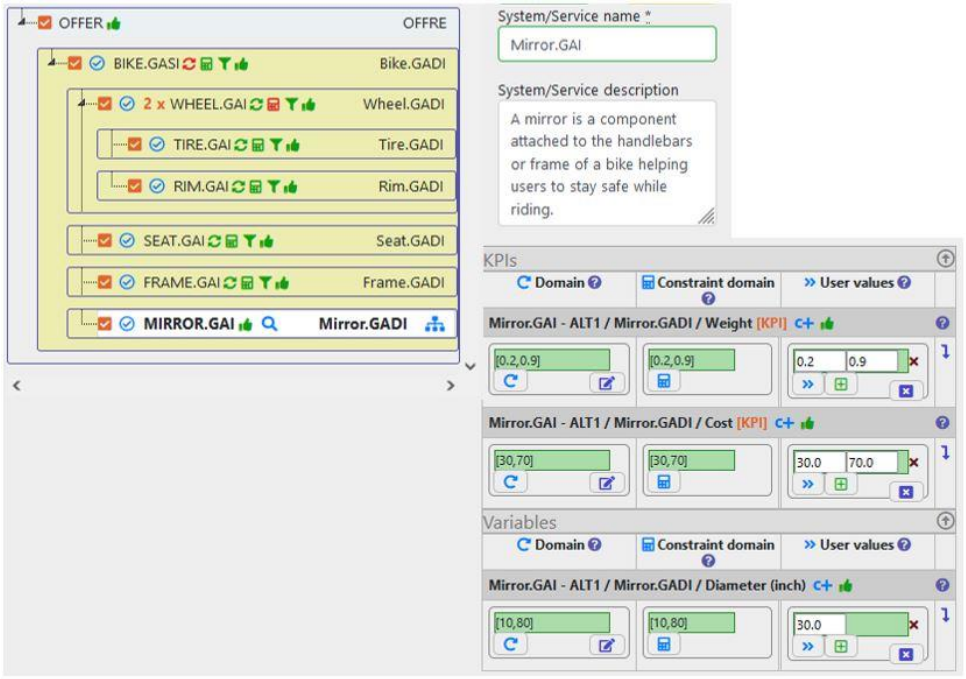

Figure 86. ETO solution for *ETO.Mirror.GAI*[(i)]

- Step 11: Manually aggregate the KPIs for *ETO.Bike.GASI* $^{(2)}_{1}$*:* the user aggregates the weight and cost and then obtains respectively the following domains: {[3.5, 11.5]} and {[445, 1270]}.
- Step 12: Propose an ETO solution: the solution is the integration of all previous solutions.

In this illustration, one can remark that it mixes CTO configuration with ETO configuration. For some parts of the system to configure, no modifications were necessary and the CTO process were carried out. But for other parts, modifications were necessary and the ETO configuration process were carried out.

5.4. Synthesis

This chapter is devoted to the illustration of our proposals explained in chapters 3 and 4 using the OPERA software, in order to check and validate them. We therefore apply our two contributions on a simple but realistic case study for the configuration of bicycles.

In section 5.1, we introduced the OPERA software and we presented the use case. In section 5.2, we defined our proposals concerning the knowledge formalization for system configuration. In OPERA, we can define GADMs at different levels of abstraction. Moreover, we can define different GASMs. As well as we can configure them to ensure their consistency. In section 5.3, we defined our proposals related to knowledge reuse for system configuration. In OPERA, we are able to define GADIs and GASIs, configure them, and meet ETO requirements by modifying the instances. In the last section, we mixed CTO configuration and ETO configuration in order to fulfill the requirements: the formalized knowledge on some parts of the bicycle had to be modified using ETO configuration process and other ones didn't necessitate any modifications and were configured using the CTO configuration process. Then, we have shown that it is possible to associate both processes.

In the next chapter, we present the conclusions of this PhD thesis and scientific perspectives.

6. Conclusions and scientific perspectives

In this chapter, we provide an overview of our research and we propose ideas and directions for future research. First, in section 6.1, we address the conclusions of this Ph.D. We then conclude with some scientific perspectives in section 6.2.

6.1. Conclusions

This PhD focuses on formalization and reuse of knowledge for system configuration. Chapter 1 serves as an initial introduction to this PhD thesis. The chapter describes briefly the two main steps of knowledge formalization and knowledge reuse. The dissertation's research questions and principal contributions are then introduced in this first chapter, followed by an overview of its structure and main concepts.

In Chapter 2, the focus is on the specifics of system configuration and presents a comprehensive analysis of the state of the art in system configuration. Based on the literature review, we have identified two scientific gaps: 1) formalize generic models at different levels of abstraction for system configuration distinguishing descriptive view and structural view (topic of our first research question), and 2) meet the user's ETO requirements during the configuration process, These ETO requirements can be either descriptive or structural and can operate at varying abstraction levels. For this bibliographic study, in order to evaluate the existing works and approaches, eight criteria have been identified. Four criteria are related to the knowledge formalization phase and four criteria are related to the reuse phase. Finally, the comparison of the different approaches according to the eight criteria led us to choose the association of ontologies and CSPs.

In this PhD, responses to the two research questions have been proposed:

RQ 1. “Is it possible to define an ontology of generic models to better manage knowledge, allowing a clear distinction between descriptive and structural views for system configuration?”

The first research question is answered in Chapter 3 which leads to our first contribution. This contribution targeted the phase of knowledge formalization. We formalized an ontology of generic models using the association of ontologies, CSPs, commonality, and inheritance principles. Our approach improves knowledge management by creating generic models hierarchically at various abstraction levels using ontologies, from the most general one to the most specialized one. This enables experts to easier maintain and update models. Our approach also enables a clear distinction to be made between descriptive and structural views of the same system. In the descriptive view, the key attributes, indicators of the system, and the relations between their values are formalized while avoiding details of its structure. This view is interesting for novice or non-expert users. In the structural view, the BOM of the system, KPIs aggregation methods, and the relations between the items composing it are formalized. This

view is interesting for expert users. In this way, we have defined the concept of Generic Artifacts at the level (i), notated $GA^{(i)}$, and its two distinct views: the descriptive view, notated $GADM^{(i)}$ and the structural view, notated $GASM^{(i)}_j$ using CSP to model several relations and ensure the consistency of the models. CSP allows for the formalization of relations between artifacts or between their characteristics using constraints and provides the opportunity to use constraint filtering to delete inconsistent values and only keep consistent ones. Then, based on the commonality of generic models, we formalized a new model at a higher level of abstraction using the generalization mechanism. This model only contains the common knowledge among models. Moreover, we formalized a new generic model at a lower level of abstraction using inheritance principles and the specialization mechanism. This model is refined and enriched based on what's specific for their new model. Therefore, in the same general model (regardless of the descriptive $GADM^{(i)}$ and structural $GASM^{(i)}_j$ view), knowledge may have either been inherited, denoted as 'I', or deliberately added, denoted as 'S'. This notation clearly indicates the quantity of both general and specific knowledge in a general model. Our approach makes the formalization process quicker and in a more structured way. Moreover, it facilitates maintenance and update allowing to keep generic models up to date. When we make a change to a high-level model, all related specific models will inherit those updates.

RQ 2. "How can ETO requirements be processed during configuration activity?"

The second research question is answered in Chapter 4 which leads to our second contribution. Our second contribution has focused on the phase of knowledge reuse. We proposed a knowledge reuse process by adapting the CTO configuration activity to fulfill ETO requirements. Therefore, our approach first covers CTO situations and we highlight what our proposals imply in CTO configuration. First, in the classical way, the user has to select a generic model, or $GA^{(i)}$ at a certain level of abstraction. An instance of his descriptive view is created, noted $GADI^{(i)}$ then configured. Once the descriptive view or $GADI^{(i),}$ has been configured, the user can decide whether or not to configure one of its structural views or $GASI^{(i)}_j$ to meet the CTO's requirements. If this is the case, the user enters a new configuration cycle for each item or GAI of the BOM. Configuration is complete when all the attributes of all the $GADI^{(i)}$ in the BOM are set to a single value. A CTO-solution has then been found. After that, our focus shifted to pure ETO situations. We proposed a generic process for knowledge reuse in ETO. The adaptation of configuration activity towards ETO involved either creating new instances or modifying existing ones to meet ETO requirements, which corresponds to performing engineering activities. We proposed several cases for modifying both GADI and GASI in order to formalize ETO requirements using CSP. During the creation or modification of GADI or GASI for ETO, we need to ensure their consistency. In the case of modification, constraint filtering cannot be used for consistency checks since it will result in deleting the added values or inconsistency. Instead, the consistency check is done by the expert user manually. Once the ETO configuration activity is complete, the resulting solutions are stored in an Experience Base (EB). Storing them allows for future reuse of solutions and update of formalized generic models. Moreover, we proposed a generic process for knowledge reuse in CTO- ETO which links between CTO and ETO and allows to fulfill both CTO and ETO requirements.

In Chapter 5, the thesis takes a practical turn by implementing some of our proposals. This is achieved by using a real bicycle example in the OPERA software. After briefly introducing the OPERA mock-up, our bike example focuses:

1) For the formalization phase: on the creation of generic artifacts $GA^{(i)}$ and their corresponding descriptive, $GADM^{(i)}$ and structural views, $GASM^{(i)}_j$ (sections 5.2.1 and 5.2.2). We also demonstrate the concepts of generalization (section 5.2.3) and specialization (section 5.2.4) of models, using the examples of wheel and bicycle families, respectively.

2) For the knowledge reuse phase: for CTO requirements, on selecting and configuring BOM components (sections 5.3.1 and 5.3.2), while for ETO requirements, on defining a new component during configuration (section 5.3.3), as well as modifying a component (section 5.3.4) and the structure of the BOM (section 5.3.5).

In conclusion, this thesis has made:

- two notable contributions to knowledge management in system configuration.
- one experiment of our proposals on a software mock-up.

Additionally, we have presented four papers at international conferences and one at a national oral presentation.

- Maryam Mohammad Amini, Michel Aldanondo, Élise Vareilles, Thierry Coudert. Twenty Years of Configuration Knowledge Modeling Research. Main Works, What To Do Next?. IEEM 2021 - International Conference on Industrial Engineering and Engineering Management, Dec 2021, Singapore, France. pp.1328-1332,
- Maryam Mohammad Amini, Thierry Coudert, Élise Vareilles, Michel Aldanondo. Integration of Ontologies and Constraint Satisfaction Problems for Product Configuration. IEEM 2021 - International Conference on Industrial Engineering and Engineering Management, Dec 2021, Singapore, France. pp.578-582,
- Maryam Mohammad Amini, Thierry Coudert, Élise Vareilles, Michel Aldanondo. System Configuration Models: Towards a Specialization Approach. MIM 2022 - 10th IFAC Conference on Manufacturing Modelling, Management and Control, Jun 2022, Nantes, France. pp.1189 - 1194, ⟨10.1016/j.ifacol.2022.09.551⟩,
- Élise Vareilles, Thierry Coudert, Michel Aldanondo, Maryam Mohammad Amini. Capitalisation de connaissances en configuration de biens et de services : vers une meilleure gestion de la communalité des modèles. CIGI QUALITA MOSIM 2023, Jun 2023, Trois-Rivières, Canada. 8 p.

6.2. Scientific perspectives

The two contributions summarized above open up various perspectives for future research. This section represents the main ones following each contribution.

Perspective 1: Considering multiple inheritance in knowledge formalization

In our knowledge formalization phase, we limited the specialization to the single inheritance, in which a child inherits the knowledge from one of the parents. However, when we delve deeper into knowledge formalization, the concept of multiple inheritance emerges as a potentially richer and more complex concept (Cardelli, 1984). This concept allows a child to inherit knowledge from at least two parents. For instance, let's consider we have formalized two generic models: one representing 'Child Bikes' and the other representing 'City Bikes'. Thanks to multiple inheritance we can create a new generic model for 'Child City Bikes' which inherits the features and structures of both 'Child Bikes' and 'City Bikes' models. The multiple inheritance facilitates the formalization of a richer model by inheriting features and structures from several parent models, resulting in a more detailed and specific child model. Efficiency is another key benefit since the formalized generic models can be used to define a new model rather than starting from scratch.

However, multiple inheritance is not without any challenges. One is the increased complexity that results from merging multiple parent models, which can make the child model more difficult to understand and maintain. Conflict resolution is another obstacle; when features or attributes from different parent models come into conflict, it can be difficult to determine which feature should take priority or how it should be merged. Moreover, it is essential to ensure that the child model does not inherit contradictory features from its parents, and therefore maintains the consistency of knowledge.

The idea would be to first formalize multiple inheritance for generic models (i.e. GA, GADM, and GASM multiple inheritance), then propose formalization processes. These processes should tackle challenges like dealing with complex situations and resolving conflicts. To do this, the following papers can be investigated as a good basis and their ideas can be adapted. (Simons, 2005) examined the theoretical challenges and resolution of conflicts related to multiple inheritance in object-oriented programming. Ducournau in (Ducournau et al., 1992) developed conflict resolution mechanisms in multiple inheritance in Object-Oriented Programming. (Ducournau & Privat, 2011) provided insights on the semantics of multiple inheritance.

Perspective 2: Considering intangible artifacts in knowledge formalization

In our knowledge formalization, we only considered tangible artifacts such as technical systems. However, nowadays configurations are not just limited to tangible artifacts; they also involve intangible artifacts like services. Services such as maintenance or training play an important role in customer satisfaction and loyalty. Moreover, customized services can provide a significant competitive advantage.

Future work should therefore focus on the formalization of generic models for intangible artifacts at different levels of abstraction and its link with the models for tangible artifacts. The idea is to create an ontology of generic models for services, in which generic models are structured from the most general one to the most specialized one. To achieve this, we must investigate how to define processes for both generalization and specialization of services models. Before that, we must first define the descriptive view and structural view of a family of services, therefore we need to adapt our proposals to the services. For instance, in the descriptive view, the features, KPIs, and the constraints linking their values can be defined while in the structural view the service components, their relations, and KPIs aggregation methods can be formalized. The following papers can be investigated as a good basis and their ideas can be adapted.

(Guillon, Ayachi, et al., 2021) studied Product-Service Systems, also called PSS which allows the combination of different types of artifacts (components, subsystems, service components, and modules) in a unique technical solution architecture. They also formalized processes for artifacts. (Dong et al., 2011) proposed an ontology-based approach for modeling knowledge for service products, adapting product configuration concepts to the service sector. They mentioned that based on the literature, components have properties, constraints, and resources. They employed languages like OWL and SWRL to formalize the knowledge of services. (Shen et al., 2012) proposed an ontology-based approach for formalizing configuration knowledge related to Product Extension Services (PESs). They began by creating meta-ontologies, which encompassed sub-ontologies for services, products, and customers, providing a foundation for the general PES configuration domain. The knowledge was then formalized using OWL, and SWRL.

Perspective 3: Measuring the level of ETO in knowledge reuse

In our knowledge reuse phase, we proposed processes for reusing knowledge in CTO, ETO, and CTO-ETO. However, we did not discuss the level of ETO. In reality, most of the time we deal with CTO-ETO in which user requirements are CTO requirements and some are ETO requirements. Among ETO requirements it can be related to Light ETO or Heavy ETO (Sylla, Guillon, Vareilles, et al., 2018). Therefore, it seems to be interesting to understand the quantity of engineering activities required to fulfill ETO requirements.

A third perspective concerns the ideas for the development of an indicator to measure the percentage of ETO within a configuration. The following works (Siddique et al., 1998) and (Thevenot & Simpson, 2006) can be considered as good references to give us some ideas. In their work, they proposed an index for measuring platform commonality, named Percent Commonality (%C) then measured it based on different views, which we explained in Chapter 2, page 24. We can adapt their proposals to our context.

In our proposals for knowledge formalization and knowledge reuse, we defined three distinct tags: ‘I’ for ‘Inherited’, ‘S’ for ‘Specific’, and ‘E’ for ‘ETO’. In ETO, all three tags ‘E’, ‘S’, and ‘I’ are used. While, in CTO, only the tags ‘I’ and ‘S’ are used. The idea would be to use these tags in order to calculate the level of ETO for GADI and GASI. To achieve the level of ETO for GADI, first, the level of ETO for its elements (i.e. attributes, attributes domains,

relations, and constraint tuples) must be calculated and then percentages must be integrated using a method. On the other hand, to obtain the level of ETO for GASI, first, it must be figured out how to compute the ETO level for each GAI composing it. Then, how to propagate the ETO level to the higher level of composition (e.g. from the component level to the subsystem level and then to the system level). Moreover, the level of ETO for other elements such as quantity domains, relations, and constraints tuples must be calculated as well. Finally, a method must be defined to integrate these percentages of ETO.

Bibliographic references

Aamodt, A., & Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, *7*(1), 39–59. https://doi.org/10.3233/AIC-1994-7104

Abdullah, R., Ibrahim, H., Atan, R., Napis, S., Selamat, M. H., Haslina, N., Hernazura, S., & Jamil, H. (2008). The Development of Bioinformatics Knowledge Management System with Collaborative Environment. *IJCSNS International Journal of Computer Science and Network Security*, *8*(2), 309–319.

Alavi, M., & Leidner, D. E. (2001). Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly: Management Information Systems*, *25*(1), 107–136. https://doi.org/10.2307/3250961

Aldanondo, M., Hadj-Hamou, K., Moynard, G., & Lamothe, J. (2003). Mass customization and configuration: Requirement analysis and constraint based modeling propositions. *Integrated Computer-Aided Engineering*, *10*(2), 177–189. https://doi.org/10.3233/ICA-2003-10207

Aldanondo, M., & Vareilles, E. (2008). Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *Journal of Intelligent Manufacturing*, *19*(5), 521–535. https://doi.org/10.1007/s10845-008-0135-z

Amini, M. M., Aldanondo, M., Vareilles, E., & Coudert, T. (2021). Twenty Years of Configuration Knowledge Modeling Research. Main Works, What To Do Next?. In *2021 IEEE International Conference on Industrial Engineering and Engineering Management*, 1328–1332. https://doi.org/10.1109/IEEM50564.2021.9673037

Amini, M. M., Coudert, T., Vareilles, E., & Aldanondo, M. (2021). Integration of Ontologies and Constraint Satisfaction Problems for Product Configuration. In *2021 IEEE International Conference on Industrial Engineering and Engineering Management*, 578–582. https://doi.org/10.1109/IEEM50564.2021.9672918

Amini, M. M., Coudert, T., Vareilles, E., & Aldanondo, M. (2022). System Configuration Models: Towards a Specialization Approach. *MIM 2022 - 10th IFAC Conference on Manufacturing Modelling, Management and Control*, *55*(10), 1189–1194. https://doi.org/10.1016/j.ifacol.2022.09.551

Andersen, B. (1993). Modelling Product Structures by Generic Bills-of-Materials. *Production Planning & Control*, *4*(3), 286–286. https://doi.org/10.1080/09537289308919447

Arana, J. (2007). Product Modeling and Configuration Experiences. *Mass Customization Information Systems in Business* (pp. 33–58). IGI Global. https://doi.org/10.4018/978-1-59904-039-4.ch002

Baker, K. R. (1985). Safety stocks and component commonality. *Journal of Operations Management*, *6*(1), 13–22. https://doi.org/10.1016/0272-6963(85)90031-2

Balmelli, L. (2007). An overview of the systems modeling language for products and systems development. *Journal of Object Technology*, *6*(6), 149–177. https://doi.org/10.5381/jot.2007.6.6.a5

Barták, R., Salido, M. A., & Rossi, F. (2010). Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing*, *21*(1), 5–15. https://doi.org/10.1007/s10845-008-0203-4

Beldiceanu, N., Carlsson, M., Demassey, S., & Petit, T. (2007). Global Constraint Catalogue: Past, Present and Future. *Constraints*, *12*(1), 21–62. https://doi.org/10.1007/s10601-006-9010-8

Beldiceanu, N., Carlsson, M., & Rampon, J.X. (2005). *Global Constraint Catalog*.

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, *30*(1), 257–261. https://doi.org/10.1145/274790.274308

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, *168*(1–2), 70–118. https://doi.org/10.1016/j.artint.2005.05.003

Bergmann, R., & Wilke, W. (1996). On the role of abstraction in case-based reasoning. In *European Workshop on Advances in Case-Based Reasoning*, 28–43. https://doi.org/10.1007/BFb0020600

Bessiere, C. (1991). Arc-consistency in dynamic constraint satisfaction problems. *AAAI Conference on Artificial Intelligence*, 221–226.

Bessière, C. (1994). Arc-consistency and arc-consistency again. *Artificial Intelligence*, *65*(1), 179–190. https://doi.org/10.1016/0004-3702(94)90041-8

Bettman, J. R., & Park, C. W. (1980). Effects of Prior Knowledge and Experience and Phase of the Choice Process on Consumer Decision Processes: A Protocol Analysis. *Journal of Consumer Research*, *7*(3), 234. https://doi.org/10.1086/208812

Blecker, T., & Friedrich, G. (2006). *Mass Customization: Challenges and Solutions* (Vol. 87). Springer Science & Business Media. https://doi.org/10.1007/0-387-32224-8

Bonev, M., & Hvam, L. (2013). Performance measures for mass customization strategies in an ETO environment. In *20th European Operations Management Association Conference*.

Borst, P., Akkermans, H., & Top, J. (1997). Engineering ontologies. *International Journal of Human-Computer Studies*, *46*(2–3), 365–406. https://doi.org/10.1006/ijhc.1996.0096

Brière-Côté, A., Rivest, L., & Desrochers, A. (2010). Adaptive generic product structure modelling for design reuse in engineer-to-order products. *Computers in Industry*, *61*(1), 53–65. https://doi.org/10.1016/j.compind.2009.07.005

Cao, J., & Hall, D. (2020). Ontology-based Product Configuration for Modular Buildings. *Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)*, *October*, 171–176. https://doi.org/10.22260/ISARC2020/0026

Cardelli, L. (1984). A semantics of multiple inheritance. In *International symposium on semantics of data types* (pp. 51–67). https://doi.org/10.1007/3-540-13346-1_2

Clark, T., & Evans, A. (1997). Foundations of the Unified Modeling Language. In *Proceedigs of the 2nd Northern Formal Methods Workshop*.Springer.

Colburn, T., & Shute, G. (2007). Abstraction in Computer Science. *Minds and Machines*, *17*(2), 169–184. https://doi.org/10.1007/s11023-007-9061-7

Collins, D. (1982). Bike. *Collins English Dictionary*. Retrieved from: https://www.collinsdictionary.com/dictionary/english/bike#google_vignette

Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM*, *32*(1), 9–23. https://doi.org/10.1145/63238.63239

DeBellis, M. (2021). A practical guide to building OWL ontologies using Protégé 5.5 and

plugins. URL: https://www. researchgate. net/publication/351037551_A_Practical _Guide_to_Building_OWL_Ontologies_Using_Protege_55_and_Plugins(дата звернення: 11.05. 2023).

Debruyne, R., & Bessiere, C. (1997). Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. *Proceedings 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 412–417.

Djefel, M., Vareilles, É., Aldanondo, M., & Gaborit, P. (2008). Vers le couplage de la conception produit et de la planification projet via une approche par contraintes. *JFPC 2008 - Quatriemes Journees Francophones de Programmation Par Contraintes*, 397–402.

Dobing, B., & Parsons, J. (2006). How UML is used. *Communications of the ACM*, *49*(5), 109–113. https://doi.org/10.1145/1125944.1125949

Dong, M., Yang, D., & Su, L. (2011). Ontology-based service product configuration system modeling and development. *Expert Systems with Applications*, *38*(9), 11770–11786. https://doi.org/10.1016/j.eswa.2011.03.064

Ducournau, R., Habib, M., Huchard, M., & Mugnier, M. L. (1992). Monotonic conflict resolution mechanisms for inheritance. *ACM SIGPLAN Notices*, *27*(10), 16–24. https://doi.org/10.1145/141937.141939

Ducournau, R., & Privat, J. (2011). Metamodeling semantics of multiple inheritance. *Science of Computer Programming*, *76*(7), 555–586. https://doi.org/10.1016/j.scico.2010.10.006

Elgh, F. (2011). Modeling and management of product knowledge in an engineer-to-order business model. In *18th International Conference on Engineering Design (ICED 11) - Impacting Society Through Engineering Design*.

Erens, F., & McKay, A. (1994). Product modelling using multiple levels of abstraction instances as types. *Computers in Industry*, *24*(1), 17–28. https://doi.org/10.1016/0166-3615(94)90005-1

Erens, F., & Wortman, H. (1996). Generic product modeling for mass customization. *Implementation Road Map, January*, 1–23.

Esheiba, L., Elgammal, A., Helal, I. M. A., & El-Sharkawi, M. E. (2021). A Hybrid Knowledge-Based Recommender for Product-Service Systems Mass Customization. *Information*, *12*(8), 296. https://doi.org/10.3390/info12080296

Falkner, A., Haselböck, A., Krames, G., Schenner, G., Schreiner, H., & Taupe, R. (2020). Solver Requirements for Interactive Configuration. *JUCS - Journal of Universal Computer Science*, *26*(3), 343–373. https://doi.org/10.3897/jucs.2020.019

Felfernig, A. (2007). Standardized configuration knowledge representations as technological foundation for mass customization. *IEEE Transactions on Engineering Management*, *54*(1), 41–56. https://doi.org/10.1109/TEM.2006.889066

Felfernig, A., Friedrich, G. E., & Jannach, D. (2000). UML as domain specific language for the construction of knowledge-based configuration systems. *International Journal of Software Engineering and Knowledge Engineering*, *10*(4), 449–469. https://doi.org/10.1016/S0218-1940(00)00024-9

Felfernig, A., Hotz, L., Bagley, C., & Tiihonen, J. (2014). *Knowledge-Based Configuration: From Research to Business Cases*. Newnes.

Felfernig, A., Jannach, D., & Zanker, M. (2000). Contextual Diagrams as Structuring Mechanisms for Designing Configuration Knowledge Bases in UML. In *International*

Conference on the Unified Modeling Language (pp. 240–254). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-40011-7_17

Forza, C., & Salvador, F. (2008). Application support to product variety management. *International Journal of Production Research*, *46*(3), 817–836. https://doi.org/10.1080/00207540600818278

Freuder, E. C. (1978). Synthesizing constraint expressions. *Communications of the ACM*, *21*(11), 958–966. https://doi.org/10.1145/359642.359654

Frohlich, P. H. (2002). Inheritance decomposed. *Inheritence Workshop at ECOOP*.

Gartner, R. (2016). The Taxonomic Urge. *Metadata: Shaping Knowledge from Antiquity to the Semantic Web* (pp. 65–75).

Ghedira, K. (2013). *Constraint satisfaction problems: csp formalisms and techniques*. John Wiley & Sons.

Golomb, S. W., & Baumert, L. D. (1965). Backtrack Programming. *Journal of the ACM*, *12*(4), 516–524. https://doi.org/10.1145/321296.321300

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220. https://doi.org/10.1006/knac.1993.1008

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human - Computer Studies*, *43*(5–6), 907–928. https://doi.org/10.1006/ijhc.1995.1081

Guillon, D. (2019). *Assistance à l'élaboration d'offres du produit au service : proposition d'un modèle générique centré connaissances et d'une méthodologie de déploiement et d'exploitation*. Doctoral dissertation, Ecole des Mines d'Albi-Carmaux.

Guillon, D., Ayachi, R., Vareilles, É., Aldanondo, M., Villeneuve, É., & Merlo, C. (2021). ProductΥservice system configuration: a generic knowledge-based model for commercial offers. *International Journal of Production Research*, *59*(4), 1021–1040. https://doi.org/10.1080/00207543.2020.1714090

Guillon, D., Sylla, A., Vareilles, E., Aldanondo, M., Villeneuve, E., Merlo, C., Thierry, C., & Geneste, L. (2017). Configuration and Response to calls for tenders : an open bid configuration model. In *19th Configuration Workshop*.

Guillon, D., Villeneuve, E., Merlo, C., Vareilles, E., & Aldanondo, M. (2021). ISIEM: A methodology to deploy a knowledge-based system to support bidding process. *Computers & Industrial Engineering*, *161*(June), 107638. https://doi.org/10.1016/j.cie.2021.107638

Hadzic, M., Wongthongtham, P., Dillon, T., & Chang, E. (2009). *Ontology-based multi-agent systems*. Germany: Springer Berlin Heidelberg.

Hanafi, R., Mejri, L., & Ghezala, H. H. Ben. (2018). Toward a domain ontology for computer projects resolution: Project memory challenge. In *KEOD - 10th International Conference on Knowledge Engineering and Ontology Developmen* (pp. 245-252)

Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, *14*(3), 263–313. https://doi.org/10.1016/0004-3702(80)90051-X

Hause, M. (2006). The SysML Modelling Language. *Fifteenth European Systems Engineering Conference, September*, 1-12.

Hegge, H. M. H., & Wortmann, J. C. (1991). Generic bill-of-material: a new product model.

International Journal of Production Economics, *23*(1–3), 117–128. https://doi.org/10.1016/0925-5273(91)90055-X

Hilpinen, R. (1992). On artifacts and works of art 1. *Theoria*, *58*(1), 58–82.

Hilpinen, R. (1999). Artifact. *Stanford Encyclopedia of Philosophy*.

Hutchinson, J., Whittle, J., & Rouncefield, M. (2014). Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, *89*, 144–161.

Janota, M., Botterweck, G., Grigore, R., & Marques-Silva, J. (2010). How to Complete an Interactive Configuration Process?. In *36th Conference on Current Trends in Theory and Practice of Computer Science* (pp. 528-539). https://doi.org/10.1007/978-3-642-11266-9_44

Jiao, J., & Tseng, M. M. (1999). An Information Modeling Framework for Product Families to Support Mass Customization Manufacturing. *CIRP Annals*, *48*(1), 93–98. https://doi.org/10.1016/S0007-8506(07)63139-4

Johnsen, S. M., & Hvam, L. (2019). Understanding the impact of non-standard customisations in an engineer-to-order context: A case study. *International Journal of Production Research*, *57*(21), 6780–6794. https://doi.org/10.1080/00207543.2018.1471239

Junker, U. (2006). Configuration. In *Handbook of Constraint Programming* (Vol. 2, Issue C, pp. 837–873). Elsevier. https://doi.org/10.1016/S1574-6526(06)80028-3

Kamsu Foguem, B., Coudert, T., Béler, C., & Geneste, L. (2008). Knowledge formalization in experience feedback processes: An ontology-based approach. *Computers in Industry*, *59*(7), 694–710. https://doi.org/10.1016/j.compind.2007.12.014

Kauffman, D. L. (1980). *Systems One: An Introduction to Systems Thinking*. Future Systems, Incorporated, 1–42.

Khoshafian, S., Blumer, R., & Abnous, R. (1991). Inheritance and generalization in intelligent SQL. *Computer Standards & Interfaces*, *13*(1–3), 213–220. https://doi.org/10.1016/0920-5489(91)90029-Y

Kim, D. H. (1999). *Introduction to systems thinking*. Waltham, MA: Pegasus Communications.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Kolodner, J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, *6*(1), 3–34. https://doi.org/10.1007/BF00155578

Kossiakoff, A., Sweet, W. N., Seymour, S. J., & Biemer, S. M. (2011). S*ystems Engineering Principles and Practice*. John Wiley & Sons

Kotha, S., & Pine, B. J. (1994). Mass Customization: The New Frontier in Business Competition. *The Academy of Management Review*, *19*(3), 588. https://doi.org/10.2307/258941

Kramer, J. (2007). Is abstraction the key to computing?. *Communications of the ACM*, *50*(4), 36–42. https://doi.org/10.1145/1232743.1232745

Krótkiewicz, M. (2018). A novel inheritance mechanism for modeling knowledge representation systems. *Computer Science and Information Systems*, *15*(1), 51–78. https://doi.org/10.2298/CSIS170630046K

Language, M., Breu, R., Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., & Thurner, V. (1997). Towards a Formalization of the Unified Modeling Language. In *ECOOP'97 Object-Oriented Programming: 11th European Conference Jyväskylä, Finland, 1997 Proceedings 11* (pp. 344-366).

Lee, H. J., & Lee, J. K. (2005). An effective customization procedure with configurable standard models. *Decision Support Systems*, *41*(1), 262–278. https://doi.org/10.1016/j.dss.2004.06.010

Lyu, G., Chu, X., & Xue, D. (2017). Product modeling from knowledge, distributed computing and lifecycle perspectives: A literature review. *Computers in Industry*, *84*, 1–13. https://doi.org/10.1016/j.compind.2016.11.001

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, *8*(1), 99–118. https://doi.org/10.1016/0004-3702(77)90007-8

Männistö, T., Peltonen, H., Soininen, T., & Sulonen, R. (2001). Multiple abstraction levels in modelling product structures. *Data & Knowledge Engineering*, *36*(1), 55–78. https://doi.org/10.1016/S0169-023X(00)00034-3

McLucas, A. C., & Ryan, M. J. (2005). Meeting Critical Real-World Challenges in Modelling Complexity: What System Dynamics Modelling Might Learn From Systems Enginee. *Proceedings of the 23rd International Conference of the System Dynamics Society, July*.

Ming, Z., Wang, G., Yan, Y., Dal Santo, J., Allen, J. K., & Mistree, F. (2017). An Ontology for Reusable and Executable Decision Templates. *Journal of Computing and Information Science in Engineering*, *17*(3), 1–13. https://doi.org/10.1115/1.4034436

Mittal, S., & Falkenhainer, B. (1990). Dynamic Constraint Satisfaction Problems. *Proceedings Eighth National Conference on Artificial Intelligence*, 25–32. http://link.springer.com/10.1007/978-81-322-3972-7_10

Mittal, S., & Frayman, F. (1989). Towards a Generic Model of Configuration Tasks. In *IJCAI International Joint Conference on Artificial Intelligence* (Vol. 89, pp. 1395-1401).

Mkhinini, M. M., Labbani-Narsis, O., & Nicolle, C. (2020). Combining UML and ontology: An exploratory survey. *Computer Science Review*, *35*, 100223. https://doi.org/10.1016/j.cosrev.2019.100223

Monge, L. G. (2019). *Knowledge-based configuration : a contribution to generic modeling , evaluation and evolutionary optimization*. Doctoral dissertation, Ecole des Mines d'Albi-Carmaux.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, *7*(C), 95–132. https://doi.org/10.1016/0020-0255(74)90008-5

Myrodia, A., Kristjansdottir, K., & Hvam, L. (2017). Impact of product configuration systems on product profitability and costing accuracy. *Computers in Industry*, *88*, 12–18. https://doi.org/10.1016/j.compind.2017.03.001

Nickerson, R. C., Varshney, U., & Muntermann, J. (2013). A method for taxonomy development and its application in information systems. *European Journal of Information Systems*, *22*(3), 336–359. https://doi.org/10.1057/ejis.2012.26

Noy, F. N., & McGuinness, D. L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. *Stanford Knowledge Systems Laboratory*.

Oddsson, G., & Ladeby, K. R. (2014). From a literature review of product configuration

definitions to a reference framework. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *28*(4), 413–428.

Ohira, Y., Hochin, T., & Nomiya, H. (2011). Introducing Specialization and Generalization to a Graph-Based Data Model. In *Knowledge-Based and Intelligent Information and Engineering Systems: 15th International Conference*. https://doi.org/10.1007/978-3-642-23866-6_1

Omg. (2012). *OMG Object Constraint Language (OCL) v2.3.1*. *03*(January), 246. https://www.omg.org/spec/OCL/2.3.1/PDF

Pedersen, C. H. (1989). Extending ordinary inheritance schemes to include generalization. In *Conference proceedings on Object-oriented programming systems, languages and applications* (pp. 407-417). https://doi.org/10.1145/74878.74920

Pérez, B., & Porres, I. (2019). Reasoning about UML/OCL class diagrams using constraint logic programming and formula. *Information Systems*, *81*, 152–177. https://doi.org/10.1016/j.is.2018.08.005

Petre, M. (2013). UML in practice. In *35th International Conference on Software Engineering* (pp. 722–731).

Pine, B. J., Victor, B., & Boynton, A. C. (1993). Making mass customization work. *Harvard Business Review*, 71(5), 108-11.

Pitiot, P., Aldanondo, M., & Vareilles, E. (2014). Concurrent product configuration and process planning: Some optimization experimental results. *Computers in Industry*, *65*(4), 610–621. https://doi.org/10.1016/j.compind.2014.01.012

Polenghi, A., Roda, I., Macchi, M., Pozzetti, A., & Panetto, H. (2022). Knowledge reuse for ontology modelling in Maintenance and Industrial Asset Management. *Journal of Industrial Information Integration*, *27*(July 2021). https://doi.org/10.1016/j.jii.2021.100298

Queralt, A., & Teniente, E. (2006). Reasoning on UML class diagrams with OCL constraints. In *International Conference on Conceptual Modeling*, 497–512. https://doi.org/10.1007/11901181_37

Reyes-Peña, C., & Tovar-Vidal, M. (2019). Ontology: Components and Evaluation, a Review. *Research in Computing Science*, *148*(3), 257–265. https://doi.org/10.13053/rcs-148-3-21

Riboni, D., & Bettini, C. (2011). OWL 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, *7*(3), 379–395. https://doi.org/10.1016/j.pmcj.2011.02.001

Rigger, E., Fleisch, R., & Stankovic, T. (2021). Facilitating configuration model formalization based on systems engineering. In *ConfWS*, 1–8.

Romero Bejarano, J. C., Coudert, T., Vareilles, E., Geneste, L., Aldanondo, M., & Abeille, J. (2014). Case-based reasoning and system design: An integrated approach based on ontology and preference modeling. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *28*(1), 49–69. https://doi.org/10.1017/S0890060413000498

Rossi, F., Van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Elsevier.

Rudberg, M., & Wikner, J. (2004). Mass customization in terms of the customer order decoupling point. *Production planning & control*, 15(4), 445-458.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). The Unified Modeling Language Reference Manual. In *Addison Wesley*.

Rüßmann, M., Lorenz, M., Gerbert, P., Waldner, M., Justus, J., Engel, P., & Harnisch, M. (2015). Industry 4.0: The future of productivity and growth in manufacturing industries. In *Boston consulting group*, 9(1), 54-89.

Sabin, D., & Weigel, R. (1998). Product configuration frameworks-a survey. *IEEE Intelligent Systems*, *13*(4), 42–49. https://doi.org/10.1109/5254.708432

Sage, A. P. (1992). *Systems engineering*. John Wiley & Sons.

Sciore, E. (1989). Object specialization. *ACM Transactions on Information Systems*, *7*(2), 103–122. https://doi.org/10.1145/65935.65936

Shaharin, S., Saad, A., Hashim, M., & Ubaidullah, N. H. (2019). Current Trends in the Integration of Case-Based Reasoning and Semantic Web. *International Journal of Academic Research in Business and Social Sciences*, *9*(14). https://doi.org/10.6007/IJARBSS/v9-i14/6518

Shen, J., Wang, L., & Sun, Y. (2012). Configuration of product extension services in servitisation using an ontology-based approach. *International Journal of Production Research*, *50*(22), 6469–6488. https://doi.org/10.1080/00207543.2011.652744

Siddique, Z., Rosen, D. W., & Wang, N. (1998). On the applicability of product variety design concepts to automotive platform commonality. *Proceedings of the ASME Design Engineering Technical Conference*. https://doi.org/10.1115/DETC98/DTM-5661

Simonis, H. (2007). Models for Global Constraint Applications. *Constraints*, *12*(1), 63–92. https://doi.org/10.1007/s10601-006-9011-7

Simons, A. J. H. (2004). The Theory of Classification, Part 10: Method Combination and Super-Reference. *The Journal of Object Technology*, *3*(1), 43. https://doi.org/10.5381/jot.2004.3.1.c4

Simons, A. J. H. (2005). The Theory of Classification Part 17: Multiple Inheritance and the Resolution of Inheritance Conflicts. *The Journal of Object Technology*, *4*(2), 15. https://doi.org/10.5381/jot.2005.4.2.c2

Soininen, T., Tiihonen, J., Männistö, T., & Sulonen, R. (1998). Towards a general ontology of configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *12*(4), 357–372. https://doi.org/10.1017/S0890060498124083

Stahl, A., & Bergmann, R. (2000). Applying Recursive CBR for the Customization of Structured Products in an Electronic Shop. In *European Workshop on Advances in Case-Based Reasoning*, 297–308.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, *25*(1–2), 161–197. https://doi.org/10.1016/S0169-023X(97)00056-6

Stumptner, M. (1997). An overview of knowledge-based configuration. *Ai Communications*, *10*(2), 111–125.

Sylla, A., Coudert, T., & Geneste, L. (2021). A case-based reasoning (CBR) approach for engineer-to-order systems performance evaluation. *IFAC-PapersOnLine*, *54*(1), 717–722. https://doi.org/10.1016/j.ifacol.2021.08.182

Sylla, A., Guillon, D., Ayachi, R., Vareilles, E., Aldanondo, M., Coudert, T., & Geneste, L.

(2018). How to deal with engineering-to-order product/system configuration?. In *ConfWS 2018-20th Configuration Workshop* (Vol. 2220, pp. 103-108).

Sylla, A., Guillon, D., Vareilles, E., Aldanondo, M., Coudert, T., & Geneste, L. (2018). Configuration knowledge modeling: How to extend configuration from assemble/make to order towards engineer to order for the bidding process. *Computers in Industry*, *99*, 29–41. https://doi.org/10.1016/j.compind.2018.03.019

Taivalsaari, A. (1996). On the Notion of Inheritance. *ACM Computing Surveys*, *28*(3), 438–479. https://doi.org/10.1145/243439.243441

Thevenot, H. J., & Simpson, T. W. (2006). Commonality indices for assessing product families. *Product Platform and Product Family Design: Methods and Applications*, 107–129. https://doi.org/10.1007/0-387-29197-0_7

Tidstam, A., Malmqvist, J., Voronov, A., & Knut, A. (2016). Formulating constraint satisfaction problems for the inspection of configuration rules. *AI EDAM*, 30(3), 313–328. https://doi.org/10.1017/S0890060415000487

Tiihonen, J., & Soininen, T. (1997). Product configurators - information system support for configurable products. Technical Report TKO-B137, Helsinki University of Technology, Finland.

Tsang, E. (1993). Foundations of Constraint Satisfaction. In *Academic Press Limited*. https://doi.org/10.1057/jors.1995.93

Tseng, H.E., Chang, C.C., & Chang, S.H. (2005). Applying case-based reasoning for product configuration in mass customization environments. *Expert Systems with Applications*, *29*(4), 913–925. https://doi.org/10.1016/j.eswa.2005.06.026

Tumnark, P., Cardoso, P., Cabral, J., & Conceição, F. (2019). An Ontology to Integrate Multiple Knowledge Domains of Training-Dietary-Competition in Weightlifting: A Nutritional Approach. *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, *12*(2), 140–152. https://doi.org/10.37936/ecti-cit.2018122.135896

Van Beek, P., & Dechter, R. (1997). Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, *44*(4), 549–566. https://doi.org/10.1145/263867.263499

Van Harmelen, F., Lifschitz, V., & Bruce Porter, (Eds.). (2008). *Handbook of knowledge representation*. Elsevier.

Van Heijst, G., Schreiber, A. T., & Wielinga, B. J. (1997). Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, *46*(2–3), 183–292. https://doi.org/10.1006/ijhc.1996.0090

Van Hertum, P., Dasseville, I., Janssens, G., & Denecker, M. (2016). The KB paradigm and its application to interactive configuration. *Theory and Practice of Logic Programming*, *17*(1), 91–117. https://doi.org/10.1017/S1471068416000156

Vareilles, É. (2005). *Conception et approches par propagation de contraintes : contribution à la mise en œuvre d' un outil d'aide interactif*. Doctoral dissertation, Toulouse: Institut Nationale Polytechnique de Toulouse .

Vareilles, É. (2015). *Configuration interactive et contraintes : connaissances , filtrage et extensions.* Mémoire d´HDR, Albi France: Écoles de Mines .

Vareilles, É., Aldanondo, M., Codet De Boisse, A., Coudert, T., Gaborit, P., & Geneste, L. (2012). How to take into account general and contextual knowledge for interactive aiding design: Towards the coupling of CSP and CBR approaches. *Engineering Applications of*

Artificial Intelligence, *25*(1), 31–47. https://doi.org/10.1016/j.engappai.2011.09.002

Vareilles, É., Coudert, T., Aldanondo, M., & Mohammad-amini, M. (2023). Capitalisation de connaissances en configuration de biens et de services : vers une meilleure gestion de la communalité des modèles. *CIGI QUALITA MOSIM 2023*.

Venkatraman, S., & Venkatraman, R. (2018). Communities of Practice Approach for Knowledge Management Systems. *Systems*, *6*(4), 36. https://doi.org/10.3390/systems6040036

Wand, Y., Monarchi, D. E., Parsons, J., & Woo, C. C. (1995). Theoretical foundations for conceptual modelling in information systems development. *Decision Support Systems*, *15*(4), 285–304. https://doi.org/10.1016/0167-9236(94)00043-6

Wesley, P. A. (2015). *Unified Modeling Language User Guide*. Pearson Education India.

Wortmann, J. C., Muntslag, D. R., & Timmermans, P. J. M. (1997). Customer-driven manufacturing. In *Customer-driven Manufacturing*. Springer Dordrecht. https://doi.org/10.1007/978-94-009-0075-2_4

Xie, H., Henderson, P., & Kernahan, M. (2005). Modelling and solving engineering product configuration problems by constraint satisfaction. *International Journal of Production Research*, *43*(20), 4455–4469. https://doi.org/10.1080/00207540500142381

Xu, Z. W., Sheng, Z. Q., & Xie, H. L. (2009). Product Configuration Knowledge Modeling Using Extend Object Model. *Applied Mechanics and Materials*, *16–19*, 394–398. https://doi.org/10.4028/www.scientific.net/AMM.16-19.394

Xuanyuan, S., Li, Y., Patil, L., & Jiang, Z. (2016). Configuration semantics representation: A rule-based ontology for product configuration. *SAI Computing Conference (SAI)*, 734–741. https://doi.org/10.1109/SAI.2016.7556062

Yang, D., & Dong, M. (2012). A constraint satisfaction approach to resolving product configuration conflicts. *Advanced Engineering Informatics*, *26*(3), 592–602. https://doi.org/10.1016/j.aei.2012.03.008

Yang, D., & Dong, M. (2013). Applying constraint satisfaction approach to solve product configuration problems with cardinality-based configuration rules. *Journal of Intelligent Manufacturing*, *24*(1), 99–111. https://doi.org/10.1007/s10845-011-0544-2

Yang, D., Dong, M., & Chang, X.-K. (2012). A dynamic constraint satisfaction approach for configuring structural products under mass customization. *Engineering Applications of Artificial Intelligence*, *25*(8), 1723–1737. https://doi.org/10.1016/j.engappai.2012.07.010

Yang, D., Dong, M., & Miao, R. (2008). Development of a product configuration system with an ontology-based approach. *Computer-Aided Design*, *40*(8), 863–878. https://doi.org/10.1016/j.cad.2008.05.004

Yang, D., Miao, R., Wu, H., & Zhou, Y. (2009). Product configuration knowledge modeling using ontology web language. *Expert Systems with Applications*, *36*(3), 4399–4411. https://doi.org/10.1016/j.eswa.2008.05.026

Zhai, Z., Martínez Ortega, J.F., Lucas Martínez, N., & Castillejo, P. (2018). A Rule-Based Reasoner for Underwater Robots Using OWL and SWRL. *Sensors*, *18*(10), 3481. https://doi.org/10.3390/s18103481

Zhang, J., Wang, Q., Wan, L., & Zhong, Y. (2005). Configuration-oriented product modelling and knowledge management for made-to-order manufacturing enterprises. *International Journal of Advanced Manufacturing Technology*, *25*(1–2), 41–52.

Zhang, L. L. (2014). Product configuration: A review of the state-of-the-art and future research. *International Journal of Production Research*, *52*(21), 6381–6398. https://doi.org/10.1080/00207543.2014.942012

Zhang, L. L., Vareilles, E., & Aldanondo, M. (2013). Generic bill of functions, materials, and operations for SAP 2 configuration. *International Journal of Production Research*, *51*(2), 465–478. https://doi.org/10.1080/00207543.2011.652745

Zhou, G., Lu, Q., Xiao, Z., Zhou, C., Yuan, S., & Zhang, C. (2017). Ontology-based cutting tool configuration considering carbon emissions. *International Journal of Precision Engineering and Manufacturing*, *18*(11), 1641–1657. https://doi.org/10.1007/s12541-017-0193-2

Integration of constraint satisfaction problems and ontologies for the formalization and exploitation of knowledge in system configuration

This thesis focuses on system configuration, a design activity involving the assembly of predefined subsystems and components. The key challenge is formalizing knowledge for system configuration at various levels of abstraction distinguishing descriptive and structural views and then reusing it to meet Engineer-To-Order (ETO) requirements. The first contribution is a knowledge formalization process for system configuration using ontologies, Constraint Satisfaction Problems (CSPs), commonality and inheritance principles. This process facilitates the creation of generic models at different abstraction levels, ensuring their consistency, maintenance, and update. Descriptive and structural views provide comprehensive system representations. The second contribution is a knowledge reuse process for system configuration by adapting a Configure-To-Order (CTO) configuration process to an ETO configuration process to meet ETO requirements. Instances of generic models are created, descriptive and/or structural, then configured by using a generic process to meet ETO requirements. Obtained solutions are capitalized in an experience base. Finally, proposals are illustrated on a bicycle example, implemented and tested on the OPERA platform, a system configuration software.

Keywords: System Configuration, Knowledge Formalization, Knowledge Reuse, Ontology, Constraint Satisfaction Problems

---

Intégration de problèmes de satisfaction de contraintes et d'ontologies pour la formalisation et l'exploitation de connaissances dans la configuration de systèmes

Cette thèse se concentre sur la configuration des systèmes, une activité de conception impliquant l'assemblage de sous-systèmes et de composants prédéfinis. Le principal défi consiste à formaliser des connaissances pour la configuration de systèmes à différents niveaux d'abstraction en distinguant les vues descriptives et structurelles, puis à les réutiliser pour répondre aux exigences de l'ingénierie ETO. La première contribution est un processus de formalisation des connaissances pour la configuration de systèmes utilisant les ontologies, les problèmes de satisfaction des contraintes (CSP) et les principes de communalité et d'héritage. Ce processus facilite la création de modèles génériques à différents niveaux d'abstraction, garantissant leur cohérence, leur maintenance et leur mise à jour. Les vues descriptives et structurelles fournissent des représentations complètes du système. La deuxième contribution est un processus de réutilisation des connaissances pour la configuration de systèmes en adaptant un processus de configuration à la commande (CTO) à un processus de configuration ETO pour répondre aux exigences ETO. Des instances de modèles génériques descriptives et/ou structurelles sont créées, puis configurées à l'aide d'un processus générique pour répondre aux exigences de l'ETO. Les solutions obtenues sont capitalisées dans une base d'expérience. Enfin, les propositions sont illustrées sur un exemple de bicyclette, mises en œuvre et testées sur la plate-forme OPERA, un logiciel de configuration de systèmes.

Mots Clés : Configuration de systèmes, Formalisation des connaissances, Réutilisation des connaissances, Ontologies, Problèmes de satisfaction de contraintes