Knowledge-based Implementation of Metalayers - The Reasoning-Driven Architecture

Lothar Hotz and **Stephanie von Riegen**[1]

Abstract. The *Meta Principle*, as it is considered in this paper, relays on the observation that some knowledge engineering problems can be solved by introducing several layers of descriptions. In this paper, a knowledge-based implementation of such layers is presented, where on each layer a knowledge-based system consisting, as usual, of a knowledge model and separated inference methods is used for reasoning about the layer below it. Each layer represents and infers *about* knowledge located on a layer below it.

1 Introduction

Typically, knowledge engineering has the goal to create a model of knowledge of a certain domain like car periphery supervision [33], drive systems [29], or scene interpretation [16]. For knowledge-based tasks, like constructing or diagnosing a specific car periphery system, a strict separation is made into *domain model* which covers the knowledge of a certain domain and a *system model* which covers the knowledge of a concrete system or product of the domain. The domain model and the system models are represented with a *knowledge-modeling language* which again is interpreted, because of a defined semantic, through a *knowledge-based system*. Examples are a terminology box (TBox) as a domain model representing e.g. knowledge about an animal domain; and an assertional box (ABox) as a system model representing e.g. a specific animal. The TBox and ABox are represented with a certain Description Logic [5] which is again interpreted through a Description Logic reasoner like PELLET or RACER. In this paper, we use configuration systems (*configurators*) as knowledge-based systems. In such systems, a domain model (also called *configuration model*) is used for constructing a system model (also called *configuration*). Configurators typically combine a broad range of inference mechanisms like constraint solving, rule-based inference, taxonomical reasoning, and search mechanisms.

The domain model and the system model constitute two layers: In the domain model all knowledge *about* possible systems is expressed, in the system model all knowledge *about* one real system is expressed. Often, these two layers are sufficient for expressing the knowledge needed for a certain knowledge-based task. However, in some domains more layers may be needed for adequately representing the domain knowledge.

We take the biological classification as an example for multiple layers. In [32], a detailed discussion of alternative modeling for biological classification is supplied. Figure 1 presents an extract of the traditional biological classification of organisms established by Carolus Linnaeus. Each biological rank joins organisms according to shared physical and genetic characteristics. We conceive of a rank as knowledge about the next layer. The main ranks are kingdom, phylum, class, order, genus, species, and breed, which again are divided into categories. Each category unifies organisms with certain characteristics. For instance, the `Mammal` class includes organisms with glands only, thus a downward specialisation from `Mammal` to its subcategories is depicted in Figure 1. For clarity reasons, only extracts of ranks and categories are given, for example the rank of kingdom contains more than the category animal, among others plants, bacteria, and fungi. The ranks are representing an additional layer ($BioCl^M$) above the domain model of the biological classification. The categories of the ranks form the domain model layer ($BioCl^D$) and each of them is an instance of the correspondent rank. The system model layer ($BioCl^S$) is covering specific individuals (also called *domain objects*), e.g. Tux the penguin. By the given classification, the need for multiple layers becomes directly evident: It is understandable that a `King Penguin` is an instance of `Breed`. But it would be improper to denote `Tux` as a `Breed`, which would hold if `King Penguin` would be a specialization of `Breed`.

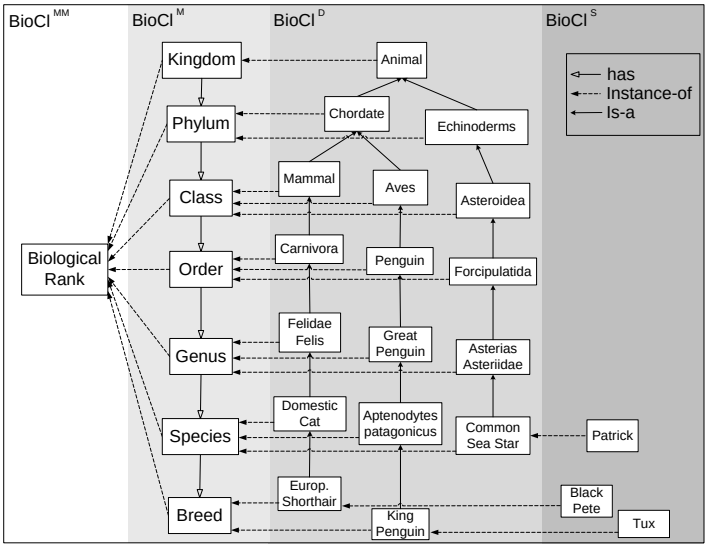

Figure 1. Biological Classification represented with several layers. Figure inspired by [4].

Because each layer specifies knowledge about knowledge in the layer below it, we also speak of a *metalayer approach* or of *metaknowledge* because as [28] pointed out: "Metaknowledge is knowledge about knowledge, rather than knowledge from a specific domain such as mathematics, medicine or geology." From a knowledge engineering viewpoint, metaknowledge is being created at the same time

[1] HITeC e.V. c/o Fachbereich Informatik, Universität Hamburg, Germany, email: {hotz,svriegen}@informatik.uni-hamburg.de

as knowledge [27]. For supporting metaknowledge, representation facilities are needed that allow the adequate representation of these types of knowledge, in here called *metaknowledge models*. A strict separation of these knowledge types from each other and the encapsulation of metalayer facilities are further requirements for a metalayer approach [6]. Furthermore, for facilitating the use and maintenance of the layers, if possible, each layer should be realized with the same modeling facilities. For being able to reason about the models on a metalayer and not only to specify them, a declarative language with a logic-based semantic should be used for implementing each layer. For allowing domain specific models on the layers, each layer should be extensible.

In this paper, we propose a Reasoning-Driven Architecture, RDA (rooted at the Model-Driven Architecture, MDA, see [23] and Section 2). The RDA consists of an arbitrary number of layers. Each layer consists of a model and a knowledge-based system. Both represent the facilities used for reason about the next lower layer.

For this task, we consider the Component Description Language (CDL) as a knowledge-modeling language which was developed for representing configuration knowledge [15]. CDL combines ontology reasoning similar to the Web Ontology Language (OWL) [2] with constraint reasoning [30], and rules [13] (see Section 3). Because knowledge-based configuration can be seen as model construction [8, 16, 15], these technology provides a natural starting point for implementing the modeling layers of RDA. Typically, a configuration model (located at the domain model layer) generically represents concrete configurations which themselves are located on the system model layer. The crucial point of RDA is to provide each layer with a model that represents knowledge about the next lower layer (see Section 4) and uses a knowledge-based configuration system to infer about this layer (see Section 5). By introducing a configuration system on each layer of the RDA, we enable reasoning tasks like consistency check, model construction and enhancement on each layer, i.e. also on metalayers.

An application of such an RDA is naturally to support the knowledge acquisition process needed for knowledge-based systems. In a first phase of a knowledge acquisition process, the typically tacit knowledge about a domain is extracted by applying knowledge elicitation methods and high interaction between a knowledge engineer and the domain expert (*knowledge elicitation phase*). A model sketch is the result, which in turn is formalized during the *domain representation phase*. During this phase a domain model is created. The domain model has to be expressed with the facilities of a knowledge-modeling language. The RDA can e.g. be used to check such knowledge models for being consistent with the knowledge-modeling language.

2 Reasoning-Driven Architecture

For defining the Reasoning-Driven Architecture (RDA), we borrow the notion of layers from the Model-Driven Architecture [24, 22, 12, 25]. In MDA, the main task is to specify modeling facilities that can be used for defining models (*metamodeling*), see for example [25]: "A metamodel is a model that defines the language for expressing a model". Or compiled to terms used here: "A metaknowledge model (called *Meta-CDL-KB*, see below and Section 3) is a knowledge model that defines CDL, which in turn is used for expressing a domain model". MDA provides four layers for modeling (see Figure 2, *MDA view*): $M2$ is the language layer represented by a metamodel, which is realized by (or *is a (linguistic, s.b.) instance-of*) a metametamodel located on the $M3$ layer. The language is used for creating a model of a specific system on the $M1$ layer. The system model represents a system which is located in the reality ($M0$ layer; not shown in the figure for brevity) [9]. Please note, that each layer contains elements which are instances of classes of the layer above. Typically a specific implementation in a tool ensures that a system model on $M1$ conforms to a metamodel on $M2$ and a metamodel on $M2$ conforms to a metametamodel on $M3$.

For clarifying our approach, we will use $R1$, $R2$, $R3$ for RDA which roughly correspond to $M1$, $M2$, $M3$ in MDA, respectively. Ri stands for "Reasoning Layer i". We separate $R1$ in several reasoning layers, because for knowledge-based systems one single model on this layer is not sufficient. This is due to the above mentioned separation of domain model and system model. $R1$ consists of a domain model specified with concepts and constraints of CDL (denoted by $R1_C$) and knowledge instances (denoted $R1_I$) representing the system model. Furthermore, corresponding reasoning facilities of CDL allow to reason about entities on layer $R1$ (see Figure 2, *CDL view*).

The RDA itself consists of multiple copies of the CDL view for representing and reason about distinct types of metaknowledge on different layers. These layers are denoted with $R1^{Mi}$ ($i \geq 0$), $R1^D$, $R1^S$ for an arbitrary number of metamodel layers, one domain model layer, and one system model layer, respectively. Because each of these layers are realized with same knowledge-based facilities, i.e. CDL concepts and instances, we do not extend CDL with the notion of a metaclass, which has instances that act as classes and can again have instances (e.g. like OWL Full [2] or like MDA/UML implementations with stereotypes [4]). Thus, the layers of $R1$ are not in one system but are clearly separated into several systems, here called *Knowledge Reflection Servers*. Each server is realized with the typical concept/instance scheme. Hence, each server can be realized with a typical knowledge-based system like Description Logic systems, rule-based systems, or as in our case a configuration system based on CDL. Through a mapping between those servers, concepts on one layer are identified with instances of the next higher layer. This mapping is a one-to-one mapping and based on a metaknowledge model (see Figure 2, *RDA view* and Section 4).

Following [4], we distinguish between a *linguistic* and an *ontological instance-of* relation. However, we explicitly name the internal *implementation instance-of* relation as such, which is a simple UML type-instance relation in [4]. The implementation instance-of relation is provided through the instantiation protocol of the underlying implementation language Common Lisp and its Common Lisp Object System (CLOS) [20, 21] (see Figure 2, classes are instances of the predefined metaclass `standard-class`). The linguistic instance-of relation is originated in the notion of classes and objects known from programming languages. In case of CDL, this relation is realized with the macroexpansion facilities of Common Lisp. Beside others, Figure 2 depicts concept definitions (`define-concept`) of CDL and their linguistic instance-of expansion to `defclass` of CLOS. The ontological instance-of relation represents relationships between knowledge elements, in CDL between concepts and instances (see above).

As we will see in Section 3, the main feature of CDL is given by the use of its inference techniques like constraint propagation. By representing the knowledge of a domain with modeling facilities of CDL these inference techniques can be applied for model construction. This representation is basically a generic description of domain objects of a domain at hand, i.e. CDL is used for specifying a domain model. For the representation of concrete domain objects, this description is instantiated and a system model is constructed. The created instances are related to each other through relations. Further-

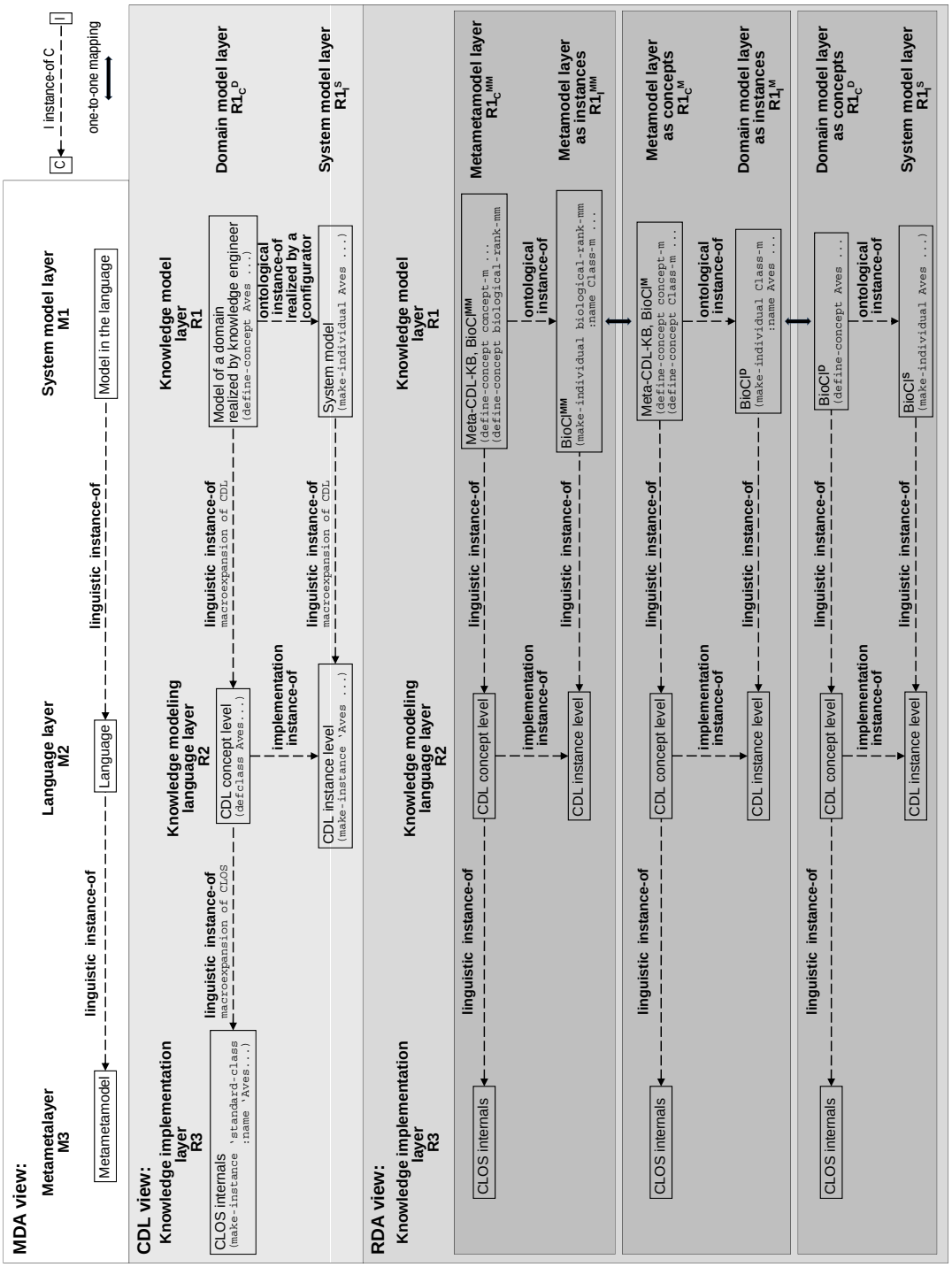

Figure 2. Comparing MDA, CDL, and RDA. The MDA view is refined when applying CDL with its domain and system model on $R1$ ($R1_C$, $R1_I$). The CDL view is three times copied for using RDA for the biological classification domain.

more, instances can be checked for concept membership.

A configuration system supports mainly three tasks (see Figure 2, *CDL view*):

1. It enables the expression of a configuration model that is consistent with the configuration language, which the system implements. For this task, the configuration system performs consistency checks of given configuration models (or parts of it) with the language specification. As a result, the configuration model is conform to the configuration language. However, the creation of the configuration model is done manually during knowledge acquisition.
2. On the basis of the configuration model, the configuration system supports the creation of configurations (system models) that are consistent with the configuration model. For this task, the system interprets the logical expressions of the configuration model and

creates configurations according to these definitions. As a result, the configurations are conform to the configuration model.

3. The configuration model can be defined in textual form or supported by a graphical user interface that enables the creation of concepts and constraints. Thus, the configuration system supplies a user interface for expressing the configuration model and for guiding the configuration process.

Thus, a configuration system supplies means for supporting the step from a domain model ($R1_C^D$) to a system model ($R1_I^S$) (see Figure 2, *CDL view*) and it can check configuration models represented with the configuration language of the system for being compliant with the language. However, the development of the configuration model is not supported with general inference techniques but is system dependent.

In RDA, this instantiation facility is used for supporting the step from the configuration language to the domain model. By applying the configuration system to a domain model that contains every model of a language, i.e. by applying it to a *metaknowledge model* (the Meta-CDL-KB), the configuration of a domain model for any specific domain is supported (Figure 2, *RDA view*, $R1_C^M$). This is achieved because of the general applicability of the language constructs of CDL, which are based on logic (see Section 3). Furthermore, other advantages of configuration systems, like a declarative representation of the configuration model, or the use of inference techniques can be applied to the Meta-CDL-KB. Thus, the construction of metamodel-compliant domain models is supported with this approach. However, the question arises: How can CDL be represented with CDL? (see Section 4.1).

3 Comprehensive Knowledge Representation

This section is organized as follows. First a brief overview of the knowledge representation language CDL (Component Description Language) [15] will be given in Section 3.1. Section 3.2 presents parts of the metamodel of CDL which have to be modeled on $R1^M$. CDL will be used in the following sections to realize the envisioned metalevels through knowledge-based implementations.

3.1 A Sketch of CDL

The CDL mainly consists of two modeling facilities: a concept hierarchy and constraints. Models consisting of concepts and constraints belong to $R1^D$, for the biological classification domain this layer is named $BioCl^D$ (see Figure 1).

The **Concept Hierarchy** contains *concepts*, which represent domain objects, a *specialization hierarchy* (based on the `is-a` relation), and *structural relations*. Concepts gather all properties, a certain set of domain objects has, under a unique name. A specialization relation relates a *super-concept* to a *sub-concept*, where the later inherits the properties of the first. The structural relation is given between a concept c and several other concepts r, which are called *relative concepts*. With structural relations a compositional hierarchy based on the `has-parts` relation can be modeled as well as other structural relationships. For example, the structural relation `has-differences` connects a species with its differentiating characteristics which is not a decomposition, to be precise. Parameters specify the attributes of a domain object with value intervals, values sets (enumerations), or constant values. Parameters and structural relations of a concept are also referred to as *properties* of the concept. *Instances* are instantiations of concepts and represent concrete domain objects. When instantiated, the properties of an instance are initialized by the values or value ranges specified in the concept. Figure 3 gives examples for concept definitions. The structural relation `has-differences` is defined, which relates one biological class with several characteristics and one characteristic with several classes. The concept for a biological class (`Mammal`) is defined with such a relation including number restricted structural relations. The right side of the operator `:>` consists of the super-concept of all relative concepts and the minimal and maximal number of those concepts. The left side restricts the total number of instances in the relation. The characteristics `Hair` and `Fur` are optional and only one of them can be used for describing a mammal, because exactly 6 characteristics are needed for specifying a mammal. Except of the metaconcept specification typical ontological definitions are given.

Constraints summarize conceptual constraints, constraint relations, and constraint instances. *Conceptual constraints* consists of a condition and an action part. The condition part specifies a *structural situation* of instantiated concepts. If this structural situation is fulfilled by some instances (i.e. the instances *match* the structural situation), the *constraint relations* that are formulated in the action part are instantiated to *constraint instances*.

Constraint relations can represent restrictions between properties like `all-different-p` or `ensure-relation`. The constraint relation `ensure-relation` establishes a relation of a given name between two instances. It is used for constructing structural relations and thus provides main facilities for creating resulting constructions. Before establishing a relation between given instances, `ensure-relation` checks whether the relation already exists. The constraint relation `all-different-p` ensures that all objects in a given `set` are of a different type. Please note, that such kind of constraints extend typical constraint technology, which is based on primitive datatypes like numbers or strings [19].

Constraints are multi-directional, i.e. they are propagated regardless of the order in which constraint variables are instantiated or changed. At any given time, the remaining possible values of a constraint variable are given as structural relations, intervals, value sets or constant values.

Constraint relations are used in the action part of conceptual constraints. Figure 6(c) gives also an example of such a conceptual constraint in CDL, however, already on a metalayer. It shows, how instances, which are selected through the structural situation, can be checked for being of a different type. When this check is fulfilled this constraint would be consistent otherwise inconsistent.

A *configuration system* performs knowledge processing on the basis of logical mappings like they are given in [31] for a predecessor of CDL. Thus, the configuration system applies *inference techniques* such as taxonomical reasoning, value-related computations like interval arithmetic [18], establishing structural relations, and constraint propagation. The structural relation is the main mechanism that causes instantiations and thus leads to an extended configuration: If such a relation is given between a concept c and several relative concepts r, depending on what exists first as instances in the configuration (c or one or more of the relative concepts r), instances for the other part of the relation may be created and the configuration increases. This capability together with the fact that descriptions, i.e. models, of systems are constructed lead to the use of configuration systems for constructing models. For a detailed description of CDL and its use in a configuration system, we refer to [15].

a)

```
(define-relation :name has-differences
  :inverse differentiate
  :domain Class-m
  :range Characteristics
  :mapping m-n)

(define-concept :name Animal
  :specialization-of domain-root
  :metaconcept Kingdom-m)

(define-concept :name Chordate
  :specialization-of Animal
  :metaconcept Phylum-m)

(define-concept :name Mammal
  :specialization-of Chordate
  :has-differences
   ((:type Characteristic :min 6 :max 6)
     :>
    (:type Glands :min 1 :max 1)
    (:type Hair :min 0 :max 1)
    (:type Fur :min 0 :max 1)
    (:type MiddleEarBones :min 3 :max 3)
    (:type WarmBlooded :min 1 :max 1))
  :metaconcept Class-m)
```

b)

```
(define-concept :name Characteristic
  :specialization-of domain-root
  :metaconcept Characteristic-m)

(define-concept :name Glands
  :specialization-of Characteristic
  :metaconcept Characteristic-m)

(define-concept :name Hair
  :specialization-of Characteristic
  :metaconcept Characteristic-m)

(define-concept :name Fur
  :specialization-of Characteristic
  :metaconcept Characteristic-m)

(define-concept :name MiddleEarBones
  :specialization-of Characteristic
  :metaconcept Characteristic-m)

(define-concept :name WarmBlooded
  :specialization-of Characteristic
  :metaconcept Characteristic-m)
```

Figure 3. Example of CDL concept definitions from the domain of biological classification.

3.2 Parts of the Metamodel of CDL

Languages are typically defined by describing their abstract syntax, their concrete syntax, and consistency rules. For describing CDL's abstract syntax, we introduce three metalevel facilities: a *knowledge element*, a *taxonomical relation* between knowledge elements, and a *compositional relation* between knowledge elements. These facilities are not to be mixed up with the above mentioned CDL facilities: *concepts*, *specialization relations*, and *structural relations*. The abstract syntax for concepts and conceptual constraints of CDL is given in Figure 5. A concrete syntax for CDL is given in Figure 3 for example.

A CDL concept is represented with a knowledge element of name $concept$ (see Figure 4 (a)), and a CDL structural relation is represented with the knowledge element $relation\text{-}descriptor$. The fact that CDL concepts can have several structural relations is represented with a compositional relation with name $has\text{-}relations$. Parameters are represented similarly.

Structural relations are defined in a further part of the metamodel (see Figure 4(b)). The fact that a concept is related by a structural relation of other concepts (the relative concepts) is represented with three knowledge elements and three compositional relations in a cyclic manner.

Figure 4(c) provides the metamodel for a conceptual constraint with its structural situation and action part. A structural situation consists of a concept expression which in turn consists of a variable and a conditioned concept. The action part consists of a number of constraint relations that should hold if the structural situation is fulfilled.

Several consistency rules define the meaning of the syntactic constructs. For example, one rule for the structural relation defines that the types of the relative concepts of a structural relation have to be sub-concepts of the concept on the left side of the operator `:>` ($rule$-5). Additionally, consistency rules are given that check CDL instances, e.g. one rule defines when instances match a conceptual constraint ($rule$-6). All rules are given in [15].

4 Metamodels

In this section, the metamodels needed for $R1^M$ and $R1^{MM}$ (Section 4.1 and [14]) and their extensions for modeling the biological classification domain (Section 4.2) are presented.

4.1 CDL in CDL

As discussed in Section 2, $R1^M$ and $R1^{MM}$ will be realized with CDL. Thus, the goal is to define the metamodel of CDL (as sketched in Section 3.2) using CDL itself. In fact, CDL provides all knowledge representation facilities needed for this purpose. The result of this is a metaconfiguration model called Meta-CDL knowledge base (Meta-CDL-KB). In Section 3.2, parts of the metamodel of CDL are defined using three modeling facilities, namely *knowledge elements*, *taxonomical relation*, and *compositional relation*. These modeling facilities are mapped to the CDL constructs *concept*, *specialization relation*, and *structural relation* respectively. Figure 5 shows how the knowledge elements shown in Figure 4 (a) can be represented with the *meta*-concepts $concept\text{-}m$, $relations\text{-}descriptor\text{-}m$, and $parameter\text{-}m$. The consistency rules of CDL have to be represented also. This is achieved by defining appropriate constraints. In Figure 5, a conceptual constraint is represented, which checks the types of a structural relation.[2]

Furthermore, instances can be represented on the metalevel by including the metaconcept $instance\text{-}m$. Having instances available, conceptual constraints and their matching instances can be represented (see Figure 5). The fact that instances fulfill a certain conceptual constraint is represented through establishing appropriate relations using the constraint relation `ensure-relation`. Please note that self references can be described also, e.g. a $concept\text{-}m$ is related to itself via the $has\text{-}superconcept\text{-}m$ relation (compare the loop in Figure 4 with Figure 5).

Other approaches also use metalevels for defining their language, e.g. UML [34, 24, 26, 25]. In contrast to these approaches, we use

[2] For a complete mapping of the CDL consistency rules to conceptual constraints see [15].

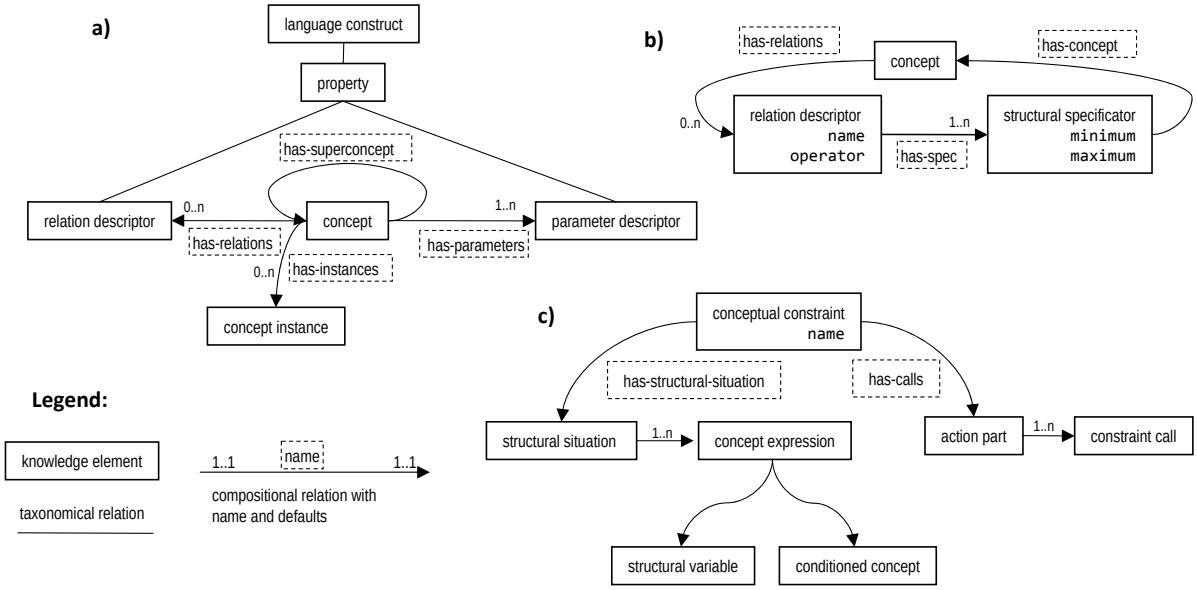

Figure 4. Metamodel for a) a concept, b) a structural relation, and c) a conceptual constraint.

```
(define-concept :name concept-m
  :specialization-of named-domain-object-m
  :concept-of-dom-m (:type domain-m)
  :superconcept-of-m
   (:type concept-m :min 0 :max inf)
  :in-some-m (:type structural-spec-m
              :min 0 :max inf)
  :has-superconcept-m
   (:type concept-m :min 0 :max 1)
  :has-relations-m
   (:type relation-descriptor-m :min 0 :max inf)
  :has-parameters-m
   (:type parameter-m :min 0 :max inf)
  :has-instances-m
   (:type instance-m :min 0 :max inf))

(define-concept :name relation-descriptor-m
  :specialization-of named-domain-object-m
  :relation-of-m (:type concept-m)
  :has-left-side-m (:type structural-spec-m
                    :min 1:max 1)
  :has-right-side-m (:type structural-spec-m
                     :min 0:max inf)
  :has-relation-definition-m
   (:type relation-definition-m :min 1:max 1))

(define-concept :name structural-spec-m
  :specialization-of domain-object-descriptor-m
  :parameters ( (lower-bound [0 inf])
                (upper-bound [0 inf]))
  :in-relation-left-m
   (:type relation-descriptor-m)
  :in-relation-right-m
   (:type relation-descriptor-m)
  :some-of (:type concept-m))

(define-concept :name instance-m
  :specialization-of named-domain-object-m
  :instance-of-dom-m (:type domain-m)
  :instance-of-m (:type concept-m)
  :matching-instance-of-m
   (:type conceptual-constraint-m)
  :has-relations-m
   (:type relation-descriptor-m :min 0 :max inf)
  :has-parameters-m
   (:type parameter-m :min 0 :max inf))
```

```
(define-concept :name conceptual-constraint-m
  :specialization-of named-domain-object-m
  :structural-situation
   (:type concept-expression-m :min 1 :max inf)
  :constraint-calls
   (:type constraint-call-m :min 1 :max inf)
  :matching-instances
   (:type instance-m :min 0 :max inf))



(define-conceptual-constraint :name consistency-rule-5
  :structural-situation
   ((?c    :type concept-m)
    (?rd   :type relation-descriptor-m
           :relation-of-m ?c)
    (?svt  :type structural-spec-m
           :in-relation-left-m ?rd)
    (?stdi :all :type structural-spec-m
           :in-relation-right-m ?rd))
  :constraint-calls
   ((all-isp ?stdi ?svt)))

(define-conceptual-constraint
  :name instance-consistency-rule-6
  :structural-situation
   ((?cc :type conceptual-constraint-m)
    (?i  :type instance-m
         :self (:condition
                (instance-matches-cc-p *it* ?cc))))
  :constraint-calls
   ((ensure-relation ?i matching-instance-of ?cc)
    (ensure-relation ?cc matching-instances ?i)))
```

Figure 5. Formalizing the knowledge elements shown in Figure 4(a) and some consistency rules with CDL concepts.

a knowledge representation language with a logic-based semantic on the metalevel, i.e. CDL instead of UML derivates like EMOF [24]. Doing so, inference techniques provided by the knowledge representation language can be used, e.g. constraint propagation. This enables the realisation of the Knowledge Reflection Server as introduced in the next section.

4.2 Extension of Meta-CDL-KB for Biological Classification

Because the metalayers are also realized with a knowledge-modeling language (here CDL) they can be extended by simply adding concept and constraint definitions to the metaknowledge base. Thus, the modeling facilities provided by such languages can not only be used for specifying the languages itself (see Section 4.1) but also for domain-specific extensions on the metalayers. In Figure 6 the extensions of $R1^{MM}$ (a) and $R1^{M}$ (b and c) for the biological classification domain are sketched, yielding to $BioCl^{MM}$ and $BioCl^{M}$ respectively. $BioCl^{MM}$ consists simply of one concept which specifies a biological rank (Figure 6 (a)). On $BioCl^{M}$, beside the concept definitions that define the metaconcepts used in $BioCl^{D}$ (see Figure 3 and 1), the conceptual constraint `Specific-Characteristics` is defined on the metalayer. This conceptual constraint checks every combination of biological classes for having specific characteristics by comparing their differences models on $BioCl^{D}$. Thus, with this conceptual constraint on $BioCl^{M}$ it is specified that a biological class should have a unique combination of characteristics. With the constraint, also classes of different phylums are tested (e.g. chordate and echinoderms). On $BioCl^{D}$, these kinds of constraints are hard to define because they are typlically not related to one specific concept but to several. Furthermore, such constraints are usually part of some modeling guidelines, e.g. for biological classification such documents state that the definitions of biological classes should be unique. Thus, by the approach presented here a *modeling of modeling guidelines* on the metalayer is achieved.

5 Knowledge Reflection Servers

Each layer described in Section 2 is realized through a Knowledge Reflection Server (KRS). Every server monitors the layer below it and consists of the appropriate model and a configuration system which interprets the model. This has the advantage of using declarative models at each metalayer as well as the possibility to apply inference techniques like e.g. constraint programming at the metalayer. Each server supplies knowledge-based services that can be called by a server below it for obtaining a judgement of its own used models. For example, a KRS monitors the activities during the construction of the domain model $BioCl^{D}$, i.e. during the domain representation phase. If e.g. a concept c of the domain is defined with `define-concept` on $R1^{D}$ the KRS on $R1^{M}$ is informed (see Figure 7) for checking its consistency. Furthermore, a KRS

- supplies services like $check\text{-}knowledge\text{-}base$, $add\text{-}conceptual\text{-}constraint$,
- creates appropriate instances of metaconcepts of the Meta-CDL-KB, e.g. $concept\text{-}m$ or $conceptual\text{-}constraint\text{-}m$,
- uses constraint propagation for checking the consistency rules,
- applies the typical model configuration process for completing the configuration, e.g. adds mandatory parts,
- checks consistency of created domain specific concepts, e.g. of $BioCl^{D}$,
- can supply new concepts for the layer below, which may be computed by machine learning methods,
- monitors the reasoning process, e.g. for evaluating reasoning performance, and thus, makes reasoning explicit,
- can create and use explanations,
- may solve conflicts that occur during the domain representation phase,
- may apply domain-specific metaknowledge, e.g. "ensuring specific differentiating characteristics of biological classes" with metacontraints as shown in Figure 6.

We implemented parts of the KRS services based on the configuration system KONWERK [10], but have not yet finished the extensive evaluation. The Meta-CDL-KB and its extensions were used for checking versions of knowledge bases for the biological classification domain. Thereby, a mapping of concept definitions of $BioCl^{D}$ to instance descriptions of $BioCl^{M}$ was realized, i.e. concept definitions on one layer are instance definitions on the next upper layer. Furthermore, the concrete syntax for defining concepts in $BioCl^{D}$ was extended, thus, metaproperties can be specified in $BioCl^{D}$. Both implementation issues as well as interfaces for the server functionality could be realized straight forward because of the flexibility of the underlying implementation language Common Lisp. However, the reasoning facilities provided by KONWERK could directly be used for scrutinizing the layers.

The validation of this approach was shown with the help of the following three scenarios, which also illustrate the use of the KRS:

- First, metaknowledge modeling can be adequately enabled, to this no workarounds with specialisations, like in [32] are needed.
- Checking domain dependent constraints: In the event of the introduction of a new biological class in $BioCl^{D}$, the KRS advises according to the specific characteristics of the constraint (e.g. see Figure 6 (c)) on $BioCl^{M}$ whether it is a biological class or not.
- Checking domain independent consistency rules: Nonrelevant to the kind of domain the domain independent rules, like the number restrictions (see Figure 5) will be checked at all times. For example, when a new kind of mammal will be introduced the defined restrictions, like the number of middle ear bones has to be conform.

6 Discussion and Related Work

Main properties of the Reasoning-Driven Architecture realized with the Knowledge Reflection Servers (KRS) as described in the previous sections are:

- the introduction of a model on one layer that represents the knowledge facilities used on the layer below it (i.e. metaknowledge models).
- the use of existing knowledge-based systems with their reasoning facilities on each layer, especially on metalayers. This enables reflection about knowledge.
- the mapping of concepts of one layer to instances of the next higher layer. This approach has the potential of using more tractable instance related inference methods instead of concept reasoning.
- the support of declarative knowledge modeling on several metalayers. This enables the modeling of knowledge and metaknowledge at the same time. Metaknolwledge is typically specified in modeling guidelines. Thus, the described approach enables the modeling of modeling guidelines.

```
(define-concept :name Kingdom-m
  :specialization-of concept-m
  :metaconcept Biological-Rank-mm)

(define-concept :name Phylum-m
  :specialization-of Kingdom-m
  :metaconcept Biological-Rank-mm)

(define-concept :name Class-m
  :specialization-of Phylum-m
  :has-relations-m
   (:type Difference-descriptor-m :min 1 :max 1)
  :metaconcept Biological-Rank-mm)
```

b)

```
(define-concept :name Difference-descriptor-m
  :specialization-of relation-descriptor-m
  :relation-of-m (:type Class-m)
  :has-left-side-m
    (:type Characteristic-some-m :min 1:max 1)
  :has-right-side-m
    (:type Characteristic-some-m :min 0:max inf))

(define-concept :name Characteristic-some-m
  :specialization-of some-m
  :in-relation-left-m
   (:type Difference-descriptor-m)
  :in-relation-right-m
   (:type Difference-descriptor-m)
  :some-of (:type Characteristic-m))

(define-concept :name Characteristic-m
  :specialization-of concept-m
  :has-relations-m
   (:type Difference-descriptor-m :min 1 :max 1)
  :metaconcept Biological-Rank-mm)
```

c)

```
(define-conceptual-constraint
  :name Specific-Characteristics
  :structural-situation
  ((?c1 :name Class-m)
   (?c2 :name Class-m
        :self #'(not-eq *it* ?c1))
   (?r1 :name Difference-descriptor-m
        :relation-of-m ?c1)
   (?r2 :name Difference-descriptor-m
        :relation-of-m ?c2)
   (?d1 :all :name Characteristic-some-m
        :in-relation-right-m ?r1)
   (?d2 :all :name Characteristic-some-m
        :in-relation-right-m ?r2)
  :action-part
  ((all-different ?d1 ?d2)))
```

a)

```
(define-concept :name Biological-Rank-mm
  :specialization-of concept-m)
```

Figure 6. Extending Meta-CDL-KB with concepts and conceptual constraints for the domain of biological classification, i.e. parts of $BioCl^M$.

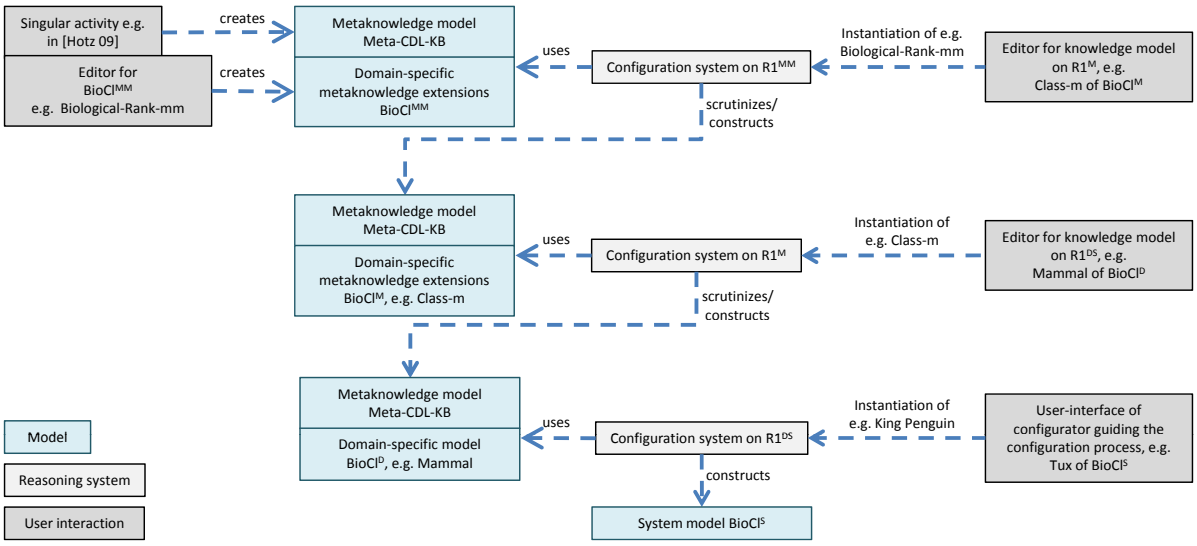

Figure 7. Monitoring the construction of metamodels and domain models through Knowledge Reflection Servers realized with configuration systems. A server using $BioCl^M$ scrutinizes $BioCl^D$ and a server using $BioCl^{MM}$ scrutinizes $BioCl^M$.

The use of reasoning methods in RDA is achieved by replacing the Meta Object Facility (MOF) of MDA [23], which is based on UML, with CDL. Other knowledge-representation languages like the Web Ontology Language (OWL) [2] could also be considered for being used on the layers. However, CDL is quite expressive, e.g. also constraints can be expressed on each layer. For realizing the KRS, even more important for us was the possibility to add server technologies to the knowledge-representation language CDL. However, by replacing the Meta-CDL-KB with a metamodel for OWL (e.g. the Ontology Definition Metamodel (ODM) [26]) one could use RDA for scrutinizing the construction of domain models written in OWL. However, with the Meta-CDL-KB a knowledge-based implementation of

a metamodel is provided and was used from a knowledge-based system (here a configurator). [3] and [11] present also approaches that include semantics on the metalayer, similar as the Meta-CDL-KB metamodel does. However, these approaches do not emphasise the use of reasoning methods on each layer as well as the capability to define domain-specific extensions on the metalayer. Furthermore, the RDA presented in this paper allows for an arbitrary number of metalayers. By introducing a configurator which allows the definition of procedural knowledge for controling the used reasoning techniques, in our approach the realization of metastrategies on the metalayers can be considered (see also [17]). (See Section 2 for further comparison to MDA and OWL.)

The creation of a metamodel for CDL with the aid of CDL has its tradition in self-referencing approaches like Lisp-in-Lisp [7] or the metaobject protocol, which implements CLOS (the Common Lisp Object System) with CLOS [21]. Such approaches demonstrated the use of the respective language and provide reflection mechanisms. With our approach such reflection mechanisms are extended from object-oriented reflection (e.g. about introspection of methods) to knowledge-based reflection (e.g. about used concepts and constraints for modeling a domain). Thus, our approach provides reflection about knowledge and a way to self-awareness of agents.

A Knowledge Reflection Server is basically an implementation of a configuration tool on the basis of the Meta-CDL-KB, i.e. of a configuration model. A typical configuration tool is implemented with a programming language and an object model implemented with it. During this implementation one has to ensure correct behavior of model construction and the inference techniques. By using CDL, this behavior (e.g. the consistency rules) is declaratively modeled, not procedurally implemented. The bases for this declarative realization are of course the procedural implementation of the inference techniques, so to speak, as a bootstrapping process. However, our approach gives indications how to open up the implementation of configuration systems or other knowledge-based systems for allowing domain-specific extensions and extensions to the inference methods and the used knowledge-modeling language.

The approach of the Metacognitive Loop presented in [1] considers the use of metalevels for improving learning capabilities and human-computer dialogs. Similar to our approach, it points out the need for enhancing agents with capabilities to reason about their cognitive capabilities for gaining self-awareness and a basis for decisions about what, when, and how to learn. However, our approach stems more from the ontology and technical use point of view and supports the idea of using metalevels from that side.

7 Conclusion

This paper presents a technology for using knowledge-based systems on diverse metalayers. Main ingredients for this task are models about knowledge (metamodels). Through the use of knowledge-based systems as they are, a Reasoning-Driven Architecture is provided. It enables reasoning facilities on each metalayer, opposed to the Model-Driven Architecture which focusses on transformations. The Reasoning-Driven Architecture is realized through a hierarchy of Knowledge Reflection Servers based on configuration systems. Future work will include meta strategies for conducting reasoning methods on the metalayers, a complete implementation of the servers, and industrial experiments in the field of knowledge engineering.

REFERENCES

[1] Michael L. Anderson and Donald R. Perlis, 'Logic, Self-awareness and Self-improvement: the Metacognitive Loop and the Problem of Brittleness', *J. Log. and Comput.*, **15**(1), 21–40, (2005).

[2] Grigoris Antoniou and Frank Van Harmelen, 'Web Ontology Language: OWL', in *Handbook on Ontologies in Information Systems*, pp. 67–92. Springer, (2003).

[3] T. Asikainen and T. Männistö, 'A Metamodelling Approach to Configuration Knowledge Representation', in *Proc. of the Configuration Workshop on 22th European Conference on Artificial Intelligence (IJCAI-2009)*, Pasadena, California, (2009).

[4] Colin Atkinson and Thomas Kühne, 'Model-Driven Development: A Metamodeling Foundation', *IEEE Softw.*, **20**(5), 36–41, (2003).

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook*, Cambridge University Press, 2003.

[6] Gilad Bracha and David Ungar, 'Mirrors: design principles for meta-level facilities of object-oriented programming languages', in *OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 331–344, New York, NY, USA, (2004). ACM.

[7] R.A. Brooks, R.P. Gabriel, and L. Steele Jr., 'Lisp-in-Lisp: High Performance and Portability', in *Proc. of Fifth Int. Joint Conf. on AI IJCAI-83*, (1983).

[8] M. Buchheit, R. Klein, and W. Nutt, 'Constructive Problem Solving: A Model Construction Approach towards Configuration', Technical Report TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, (January 1995).

[9] Dragan Gaŝević, Dragan Djuric, Vladan Devedzic, and Bran Selic, *Model Driven Architecture and Ontology Development*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] A. Günter and L. Hotz, 'KONWERK - A Domain Independent Configuration Tool', *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).

[11] P. Haase, R. Palma, and d'Aquin M., 'Updated Version of the Networked Ontology Model', Project Deliverable D1.1.5, Neon Project, (2009). www.neon-project.org.

[12] W. Hesse, 'More Matters on (Meta-)Modelling: Remarks on Thomas Kühne's "Matters"', *Journal on Software and Systems Modeling*, **5**(4), 369–385, (2006).

[13] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean, 'SWRL: A Semantic Web Rule Language Combining OWL and RuleML', W3c member submission, World Wide Web Consortium, (2004).

[14] L. Hotz, 'Construction of Configuration Models', in *Configuration Workshop, 2009*, eds., M. Stumptner and P. Albert, Workshop Proceedings IJCAI, Pasadena, (2009).

[15] L. Hotz, *Frame-based Knowledge Representation for Configuration, Analysis, and Diagnoses of technical Systems (in German)*, volume 325 of *DISKI*, Infix, 2009.

[16] L. Hotz and B. Neumann, 'Scene Interpretation as a Configuration Task', *Künstliche Intelligenz*, **3**, 59–65, (2005).

[17] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor, *Configuration in Industrial Product Families - The ConIPF Methodology*, IOS Press, Berlin, 2006.

[18] E. Hyvönen, 'Constraint Reasoning based on Interval Arithmetic: the Tolerance Propagation Approach', *Artificial Intelligence*, **58**, 71–112, (1992).

[19] U. John, *Konfiguration and Rekonfiguration mittels Constraint-basierter Modellierung*, Infix, St. Augustin, 2002. In German.

[20] Sonya E. Keene, *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS*, Addison-Wesley, 1989.

[21] J. Kiczales, D. G. Bobrow, and J. des Rivieres, *The Art of the Metaobject Protocol*, MIT Press, Cambridge, MA, 1991.

[22] T. Kühne, 'Matters of (Meta-)Modeling', *Journal on Software and Systems Modeling*, **5**(4), 369–385, (2006).

[23] OMG, *MDA Guide Version 1.0.1, omg/03-06-01*, Object Management Group, 2003.

[24] OMG, *Meta Object Facility Core Specification, version 2.0, formal/2006-01-01*, Object Management Group, 2006.

[25] OMG, *Unified Modeling Language: Infrastructure, version 2.1.1, formal/07-02-06*, Object Management Group, 2007.

[26] OMG, *Ontology Definition Metamodel, Version 1.0*, Object Management Group, 2009.
[27] Gilbert Paquette, 'An Ontology and a Software Framework for Competency Modeling and Management', *Educational Technology & Society*, **10**(3), 1–21, (2007).
[28] J. Pitrat, 'Metaconnaissance, avenir de l'Intelligence Artificielle.', Technical report, Hermes, Paris, (1991).
[29] K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt, 'A Structure-Based Configuration Tool: Drive Solution Designer DSD', *14. Conf. Innovative Applications of AI*, (2002).
[30] D. Sabin and E.C. Freuder, 'Configuration as Composite Constraint Satisfaction', in *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 153–161. AAAI Press, (1996).
[31] C. Schröder, R. Möller, and C. Lutz, 'A Partial Logical Reconstruction of PLAKON / KONWERK', in *DFKI-Memo D-96-04, Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96*, ed., F. Baader, (1996).
[32] Stefan Schulz, Holger Stenzhorn, and Martin Boeker, 'The ontology of biological taxa', in *ISMB*, volume 24, pp. 313–321, (2008).
[33] K. Wolter, T. Krebs, and L. Hotz, *Mass Customization - Challenges and Solutions*, chapter Model-based Configuration Support for Software Product Families, 43–62, Springer, 2006.
[34] Dong Yang, Ming Dong, and Rui Miao, 'Development of a Product Configuration System with an Ontology-Based Approach', *Comput. Aided Des.*, **40**(8), 863–878, (2008).