**FRIEDRICH-SCHILLER-UNIVERSITÄT JENA**


# JENAER SCHRIFTEN

## ZUR

# MATHEMATIK UND INFORMATIK

**Eingang: 30.11.2015 Math/Inf/07/2015 Als Manuskript gedruckt**

"User-Extensible Predicates to Incorporate Organizational Facts into GRL Models and an Ontological, Rule-Based Analysis"

Frederik Schulz
Johannes Meißner
Wilhelm Rossak

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Institut für Informatik
Ernst-Abbe-Platz 2
07743 Jena

# Abstract

Software Engineering Projects are strongly influenced by a plurality of technical, organizational and even social factors. The Goal-oriented Requirements Language (GRL) is generally suitable for capturing the interdependencies between actors, their goals and intentions, tasks, they have to perform, and resources, they have to provide to each other. We utilize these capabilities to model the context of software projects including the work breakdown structure, team compositions and soft factors like motivations or social team roles. For this purpose, we extended the GRL by specific elements to represent project participants and their particular contributions. This domain specific language is called Organizational Context Analysis (OrCA). To incorporate further hard facts like technical decisions, developer skills, budgets or realization costs, OrCA was added a new predicate element. It allows to enrich a model with qualitative and quantitative statements about other model entities. Moreover, we defined extension points, that enable an user to add custom domain- and application-specific predicate types. As these models can become very complex, they are split into different views. Each view shows exclusively a section of these predicate types to examine a particular aspect, e. g. only cost relevant information. To ensure a model's consistency on a global level, including all information that is spread over these views, it is transferred into an OWL ontology. Hereby, user-defined integrity rules can be formalized as SPARQL queries, that are applied to a model's ontology to reveal possible conflicts or problems. This work explains the concept of OrCA predicates in detail, outlines the transfer of OrCA models into the ontology and illustrates the concepts for the specification of user-defined predicate types and integrity rules.

# 1 Introduction

## 1.1 The Goal-oriented Requirements Language (GRL)

The organizational context of software development projects has a strong influence on the software architecture. For example, *Conway's Law* [1] emphasizes the correlation between an organizational structure and the structure of the developed products. These interdependencies concern a lot of soft factors, e. g. stakeholder intentions or developer experiences that are hard to measure. The *i*-Framework* [28] or the *Goal Requirements Language* (GRL) [8] that is based on i*, are capable of capturing soft factors in a system of actors, goals and other intentional elements and their relationships among each other. An abstract example, covering a selection of these elements, is shown in Fig. 1.

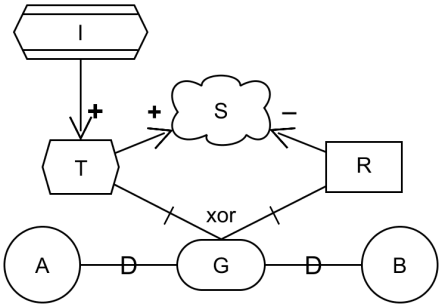


Figure 1: This abstract example illustrates a selection of GRL elements: An actor A depends on an actor B to fulfill the goal G. It can be achieved in two alternative ways: The execution of the task T or the provision of the resource R. Because of the negative impact of resource R on the softgoal S, the execution of task T would be preferable. An indicator I represents a measurable real-world value that influences the performance of task T.

The evaluation of GRL models is based on *evaluation values*. These are quantitative values ($[-100, \ldots, 100]$) [8, Sec. 7.5.3 c] or qualitative values (Denied, Unknown, Satisfied, ...) [8, Sec. 7.5.4] that express the degree of satisfaction of an element, e. g. the level of fulfillment of a goal. A *strategy* is an initial assignment of evaluation values to a subset of intentional elements; these values are propagated to other model elements according to the links, connecting them [8, Sec. 7.5].

## 1.2 Organizational Context Analysis (OrCA)

In our previous work [23] we introduced the basic concepts for our GRL-based *Organizational Context Analysis* (OrCA) framework. It captures the various interdependencies between a project's organizational context on the one hand and the technical decisions on the other hand. See the simple example in Fig. 2 that gives an overview of the most important elements in OrCA. The work breakdown structure, including the assignment of development tasks to project participants, is structured in three layers: The bottom *Deliverables* layer contains

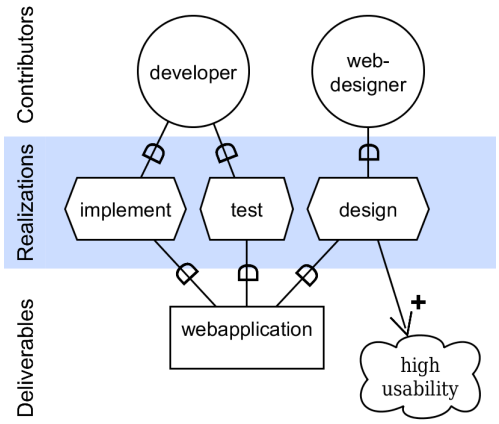


Figure 2: An example for an OrCA model. Two contributors are working on a webapplication. Particularly, the UI design task has a positive influence on the general project softgoal to provide a high usability.

special GRL resource elements that represent the achieved project results. The example presents only one *deliverable* webapplication. All realization efforts that are required to achieve these deliverables are represented by *realization* elements that are a derivation of the GRL tasks. They are contained in the intermediate *Realizations* layer. The example shows three separate tasks that are required to be performed for the realization of the webapplication: The programming effort implement, the quality assurance effort test and the user interface development effort design. The latter one has a positive contribution to the achievement of the softgoal to ensure the high usability of the webapplication. Last, the assignment of realizations to responsible project participants is done in the topmost *Contributors* layer. Every participant is represented by a *contributor* element that is a specialization of the GRL actor. Our example includes two contributors: A developer that is responsible for the implementation and testing of the webapplication, and a webdesigner for the UI design, respectively. Each realization element must be connected to exactly one contributor and exactly one deliverable, to prevent ambiguities.

An OrCA model can be enriched with a variety of additional facts like a contributor's skills, a contributor's social team roles, a realization effort's cost, allocated budgets etc. For this purpose, the OrCA meta model provides a *predicate* element. It is used to make statements regarding such organizational, technical or social facts. This concept is described in detail in the following section.

## 1.3 Motivation

It is explicitly not our objective to replace any existing tool or method for project, personnel or budget planning nor for software architectural design. Our approach targets to assist in an early project planning phase, to align organizational and technical key factors and to examine the feasibility of a project. A user should be supported by OrCA to create early drafts of the project setup, to reason about possible obstacles or functional problems. The multitude of

heterogeneous hard facts that influence a software project and their interdependencies with soft facts like intentions, individual goals and motivations often remains unspoken. OrCA should assist in bringing these issues to mind, committing them to paper and discussing them explicitly. This work introduces and refines the following aspects of our work:

1. OrCA predicates: Basic concepts, predicate types, predicate categories and views (Section 2).
2. Integrity check concepts: Evaluation steps and requirements for a rule-based, user-extensible evaluation of OrCA models (Section 3).
3. OWL: Ontology: Transfer of meta models and model instances to an OWL representation to provide a specification and a semantic underpinning (Section 4).
4. Integrity check implementation: Utilizing OWL class expressions and SPARQL queries to analyze and evaluate OrCA models (Sections 5, 6).

# 2 User-extensible OrCA-predicates

## 2.1 Ternary statements using the predicate element

To enrich OrCA diagrams with further hard facts from the organizational context, we introduced ternary statements with a *subject–predicate–object* structure. A subject can be an actor, a task, a resource or any of their OrCA derivations contributor, realization and deliverable. The object has to be a GRL resource, a GRL indicator or a fix value represented by an OrCA *data literal*. Subject and object are connected over an OrCA-specific *predicate element*. Its *predicate type* defines the semantic of the relationship between subject and object. For example, a predicate of the type has skill can be used to indicate that a contributor (the subject) has experience with some technology (the object). Predicates are a generic concept that can be adapted to any domain. Further application examples are predicates to map a deliverable to the technology, it is based on, to map a deliverable to its associated software license or even to assign a contributor to a social team role as illustrated in our previous work in [12]. Predicates are represented by triangles that are uniformly labeled with the particular predicate type. The connections from subject to predicate and from predicate to object are depicted by arrows with filled arrowheads, an example is shown in Fig. 3. Every predicate must be connected to exactly one subject and exactly one object.

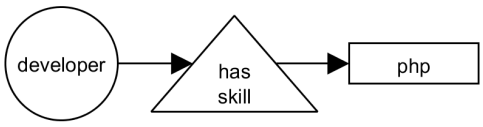


Figure 3: Example for an OrCA-predicate: The developer (subject) has skills (predicate) in the programming language php (object).

## 2.2 Predicate categories

OrCA distinguishes three categories of predicates:

- **Entity predicates**: The object of an entity predicate is always a GRL resource element. These predicates are used to express the semantic of the relationship between the subject element and some physical or informational entity or some concept or technology. The has skill predicate in Fig. 3 is an example for this category. Entity predicate elements can be assigned a GRL evaluation value and are allowed to be the source or destination of GRL dependency, decomposition or contribution (types *Make* and *Break* only) links. Hence, they can be taken into account in a GRL evaluation. The semantic of such a connection is outlined in subsection 2.6.

- **Data predicates**: The object of a data predicate must be an OrCA data literal. In contrast to an indicator that represents a *variable* real-world value, a data literal represents a *fix* value of a specific data type. The concept of OrCA data literals is explained in detail in the following subsection. A statement using a data predicate expresses the semantic of the relationship between the subject and the object's value. For example, a realization element can be connected over a has cost predicate to a data literal, to express the cost value of this realization. In contrast to entity predicates, a data predicate element must neither be the source nor the target of a contribution, dependency or decomposition link and is not assigned a GRL evaluation value. Hence, data predicate elements do not effect a GRL evaluation.

- **Indicator predicates**: An indicator predicate is the same as a data predicate, but connects to a GRL indicator instead of an OrCA data literal. Examples for data and indicator predicates are shown in Fig. 4.

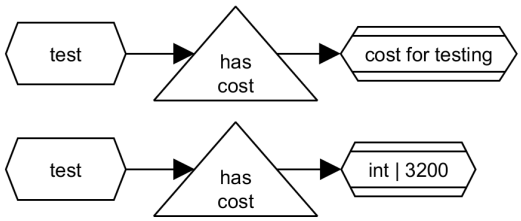


Figure 4: Above: An example for an indicator predicate. The test effort (subject) will have the cost (predicate) that is represented by the cost for testing indicator (object). This indicator is a variable placeholder for the actual value. Below: The same example, using a data literal as object that represents the fix integer value 3200.

## 2.3 Data literals

Like GRL indicators, OrCA data literals represent measurable real-world values, for example costs, budgets, technical parameters etc. Both concepts are closely related, but differ in a in a number of points:

- GRL indicators [8, Sec. 7.6] are used as variables that can be bound to a real-world data-source like a web-service, to provide their actual value. In contrast, an OrCA data literal is confined to a *fix* value that is known or specified a priori by the user.
- For GRL indicators a conversion method is used to map their associated real-world value to their *evaluation value* [8, Sec. 7.6.1 c]. This evaluation value is used for the algorithmic evaluation of a GRL model according to the specification in [8, Sec. 11.1]. In contrast, OrCA data literals are not to be considered by this GRL-specific evaluation mechanisms and thus are not assigned any evaluation value.
- GRL indicators can be connected by GRL contribution, dependency and decomposition links. In contrast, OrCA data literals can exclusively be connected to a predicate in the role of an object in an OrCA statement.
- Data literal elements use the same symbol as indicators (a hexagon with two additional horizontal lines, [8, Sec. 7.8.5 b]). An indicators meaning can be expressed in it's label, e. g. "costs for testing". In contrast, an OrCA data literals are labeled with a data type ( e. g. integer or string) and a fix value, separated by a "|"-symbol, e. g. int | 3200. The meaning of a data literal is determined by the type of the predicate, it is connected to.

The data literal concept provides a method that is less powerful, but easier to use to enrich a model with concrete data. A user can add constant values to a model that are visualized directly in its diagram. This reflects the purpose of OrCA, to help an user to develop first drafts of a software project configuration in the early planning phase in a simple way.

## 2.4 Predicate types

The meaning or semantic of a predicate is determined by its type. A predicate type definition is composed of the following parameters that can be specified by a user:

- **Name**: A name that ideally reflects a predicate's purpose, e. g. has cost to assign a cost value to some element or has skill to express a sufficient level of experience of an actor in some technology.
- **Identifier**: A technical representation of the name as string of at least one character, including only capital and lower case letters. It is used to refer to the predicate type in OWL ontologies, e. g. `HasCost`.
- **Category assignment**: Every predicate type belongs either to one of the categories entity, data or indicator predicates.
- **Domain**: A subset of the elements {Actor, Resource, Task, Contributor, Realization, Deliverable} that may be used as subject for predicates of this type. For example, a has skill predicate expresses a level of experience and thus should only be applied to a GRL actor or an OrCA contributor. Declaring the subset {Actor, Contributor} as domain of this predicate type, prevents the incorrect usage with other element types like GRL resource or GRL task.

- **Range**: The set of objects that are allowed for predicates of this type. The following cases can be distinguished:
  - If the predicate belongs to the entity predicate category, the range is generally the GRL resource element.
  - If the predicate belongs to the indicator predicate category, the range is generally the GRL indicator element.
  - If the predicate belongs to the data predicate category, the range can be specified as data type. For example, the has cost predicate type can be restricted to be connected to integer values. This excludes nonsensical connections to data literals with other data types like date, time, boolean or string. For the sake of simplicity, in this work the set of permitted data types is limited to the XML Schema Definition types `xsd:integer` and `xsd:string` as defined in [20] that can be used in OWL.

## 2.5 Views

In contrast to our brief introductory example, OrCA models that involve a large set of predicate types can become very complex. To preserve the comprehensibility of the diagrams, OrCA-models are separated into a *basic-model* and several *views*. Basic-models include the three aforementioned OrCA layers, comprising all contributors, realizations and deliverables, and further GRL elements like general project goals as shown in Fig. 2. Such a basic-model is used as scaffolding for a set of views. Each of these views enriches the basic-model with statements that are limited to a particular subset of predicate types. For example, our *cost view* shows only statements using the predicate types has cost and has budget, to express realization costs and contributor budgets, respectively. In other words, the cost-view restricts to all cost-relevant information. Hence, a statement using a has skill predicate will not appear in this view. Views may overlap, i. e. they are allowed to share predicate types. Figure 5 shows an example for a cost view and a skill view on our introductory example.

## 2.6 Semantic

We specify the truth value of an OrCA statement using an entity predicate type either to be *true*, or to be *false* or to be *unknown*. For example, the webapplication in Fig. 5 is based on php (statement is true) or it is not (statement is false) or information about this fact is missing – there is no intermediate value. To take this into account in a GRL model evaluation, we utilize the GRL evaluation values. As representative for the whole statement, we map its truth value to the evaluation value of the predicate element. The following cases can be distinguished:

- A statement is **false**. This is equal to a predicate element's evaluation value of $-100$ (quantitative) or Denied (qualitative), respectively.
- A statement's truth value is **unknown**. This is equal to a predicate element's evaluation value of 0 (quantitative) or Unknown (qualitative), respectively.

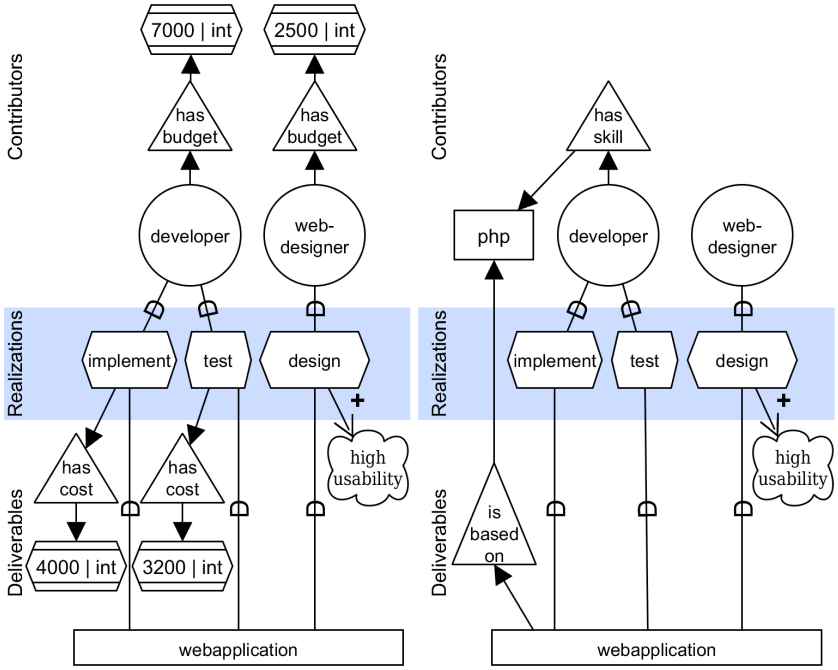


Figure 5: Left: A cost view on the introductory example: The budget of contributor developer is 7000. The costs of the realization efforts, the developer is responsible for, are 4000 and 3200. Clearly, the developer's budget is exceeded: $4000 + 3200 = 7200$. The web-developer's budget is 2500, but cost information for the design task is missing in this model. Right: A diagram showing a skill view on the same OrCA model. The developer has sufficient php skills that are required for the development of the webapplication. In contrast, the webdesigner lacks experience with php.

- A statement is **true**. This is equal to a predicate element's evaluation value of 100 (quantitative) or Satisfied (qualitative), respectively.

In a GRL evaluation, these evaluation values are propagated from or to other elements over contribution, dependency or decomposition links that are connected to the predicate element. The semantic of such links was described in [23, Sec. IV B] and is outlined in Fig. 6.

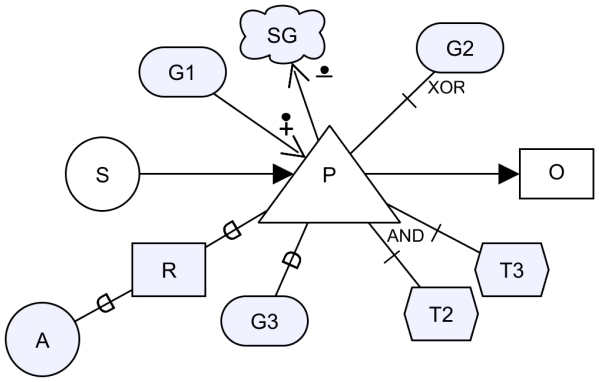


Figure 6: Combining statements and GRL links: The satisfaction of the statement (S, P, O) **1.)** ... is made by the satisfaction of goal G1 **2.)** ... breaks the satisfaction of softgoal SG **3.)** ... is one alternative mean for the satisfaction of goal G2 **4.)** ... can be achieved by task T2 and task T3 **5.)** ... is required for the satisfaction of goal G3 **6.)** depends on resource R provided by actor A (cf. [23, Sec. IV B, Fig. 7]).

Of course, this semantic could be applied to data or indicator predicates, as well, but from our perspective, this lacks any practical benefit. For example, the second statement in Fig. 4 expresses that the cost of the test effort is 3200. Evaluating the truth-value of this statement is equal to deliberating whether the cost is *exactly* 3200 or not. For example, an actual cost value of 3201 would falsify this statement. To express a non-binary, continuous correlation between a concrete value and the satisfaction level of an intentional element, a combination of the predicate concept and the native evaluation mechanisms for GRL indicators are the better choice. An example for such a construct is given in Fig. 7.

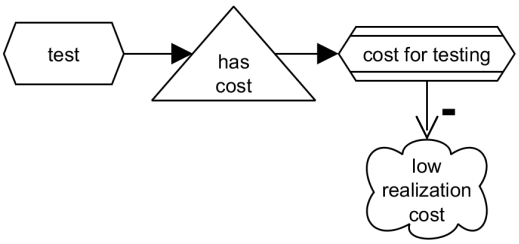


Figure 7: The indicator predicate has cost expresses the semantic relationship between the test effort and its cost value. A continuous negative influence of this value on a softgoal to keep the costs down, is represented by a negative contribution link.

To reveal possible conflicts in an OrCA model, we provide a rule-based evaluation mechanism. This concept is complementary to the GRL evaluation that is based on evaluation values and strategies. For example, such a rule can be applied to an OrCA model to detect missing skills of a contributor, exceeded budgets or adverse team compositions relating to social team roles. Therefore, an OrCA model is transferred to an OWL ontology. Thereby, predicate-specific rules can be formalized as SPARQL queries that are applied directly to this ontology. These rules reflect the domain-specific, functional semantic of a predicate and assist a user in creating a consistent model. Our ontological, rule-based evaluation concept is described in detail in the following section.

## 2.7 Extensibility

An user must be enabled to adapt the OrCA meta model to the specific needs of the particular domain, the software engineering project is embedded in. Possible domains are scientific projects, joint industrial projects, projects in a clinical environment or any other conceivable software engineering context. Thereto, OrCA provides three extension points that can be customized stepwise:

1. Creation or reuse of custom predicate type definitions.
2. Definition or reuse of custom views that comprise these predicate types.
3. Formulating of custom SPARQL queries to enable global automated, complex and domain-specific integrity checks.

This adaption process is complex, effortful and requires a deep understanding of the domain, the OrCA meta model and the involved technologies OWL and SPARQL. Nevertheless, as predicate types and rules are reusable, such an adaption is only a one-time task.

# 3 Integrity checks

The variety of organizational influence factors can lead to complex models even in simple project setups with few participants. Thus, model inconsistencies or functional problems can easily be overlooked. Furthermore, the entirety of information in an OrCA model is spread over a set of views. Obviously, changes in one view can lead to conflicts in another one. Therefore a user needs assistance to preserve a model's integrity on a global level. To guide the user in the early planning phase, we aim at an automated functional evaluation of the OrCA models. This complements the GRL specific evaluation mechanisms that are based on evaluation values and strategies. We will illustrate our concepts by three simplified evaluation examples in the following.

1. Check for an exceeded budget: The evaluation has to detect all cases where the sum of the costs of all realizations that are assigned to a contributor exceed this contributor's budget. The cost view in Fig. 5 contains two has budget predicates that connect both contributors to the fix values 7000 and 2500, respectively. The realization costs

of implement (4000) and test (3200) are denoted by two has cost predicates. In this particular case, the sum of the costs for contributor developer is higher than its budget.

2. Check for missing cost data: The prerequisite for such a check for exceeded budgets is the completeness of the cost information in the model. In our example, the realization design is missing a has cost predicate. Thus an overrun budget cannot be detected reliably.

3. Check for missing skills: The skill view in Fig. 5 extends the basic-model by a has skill predicate to indicate that contributor developer is experienced with the php technology. Furthermore, it adds an is based on predicate to determine php as technological choice for the deliverable webapplication. Hence, the developer has sufficient skills to work on the php based deliverable. In contrast, the webdesigner lacks experience with php. This is a possible deficiency in the organizational setup of this project.

The evaluation is divided in four steps:

1. **Step I: Inference**. New information has to be inferred from a given model instance. For example, we want to seek out all elements in the realization-layer that are connected to a has cost predicate. This is an intermediate result that will be processed further in step III.

2. **Step II: Structural consistency checks**. We are going to validate the model's consistency. For instance: Is the destination of each has contributor link exclusively a member of the contributor class?

3. **Step III: Data completeness checks**. A set of integrity checks has to assure that the quantity of the information that is contained in a model is sufficient. In the example in Fig. 5 missing cost information hinders the check for exceeded budgets.

4. **Step IV: Checks for functional conflicts**. An additional set of integrity constraints will detect complex functional problems like missing skills or exceeded budgets. The prerequisites are a consistent model and data completeness according to steps II and III. The user-defined set of predicate types and integrity rules is highly dependent on the project's domain.

According to these evaluation steps and our objectives, we impose the following requirements:

1. The evaluation has to facilitate the use of arithmetic aggregate functions. For example, the check for overrun budgets requires the totaling of all realization costs of a contributor.

2. The evaluation has to facilitate the use of logical expressions. For example, the check for missing cost data is equivalent to the computation of the set of *all* realizations that are *not* connected to any has cost predicate. This requires the evaluation of quantifiers, logical conjunctions and negation.

3. Due to the strong heterogeneity of software engineering projects, it would not be suitable to provide one universal catalog of integrity rules. As explained in the introduction, the set of rules has to be user-extensible and adaptable to customize the framework according to the specific needs of a project's domain.

4. For the sake of usability and user acceptance, the analysis framework has to be built upon established and standardized notations and languages.

For these purposes, the following sections transfer the OrCA model into an OWL ontology. Generic OWL axioms provide extension points for users to extend this ontology by custom predicates types in a standardized way. Additional OWL class expressions are used to infer helpful information from an OrCA ontology (evaluation step I). Integrity rules are formalized as SPARQL queries that are applied to the ontology to reveal structural inconsistencies (step II) and missing data (step III) and to detect functional conflicts (step IV).

# 4 Building an OrCA-Ontology

## 4.1 Utilizing the Web Ontology Language

In this section, we transfer the OrCA model to an ontology. Therefore, we utilize the *OWL 2 Web Ontology Language* as specified by the *W3C* in [16]. This serves multiple purposes:

1. **Specification**: An OWL ontology provides a formal structure and an unambiguous semantic underpinning.

2. **Automated analysis**: The OWL syntax is machine readable. New information can be inferred by reasoning tools. Query languages can be used to extract specific information.

3. **Tool support**: OWL enables the utilization of a broad range of well-engineered tools like the graphical development environment *Protégé* [14] or reasoners like *Pellet* [24].

4. **Integration**: The OWL API allows for an easy integration with Java applications.

The basic elements of OWL ontologies are *classes* and *properties*. Similar to object-oriented languages, classes are used to subsume things of the same type. For example, the class `OrCA:Contributor` can be derived from the OrCA model. According to the example in Fig. 5, this class has two members `OrCA:developer` and `OrCA:webdesigner` that are called *individuals*. Particularly, classes may overlap, i. e. they can share one or more individuals. Furthermore, subclasses can be defined as a subset of another class. To define the relations between individuals, OWL uses *object properties* that assign a particular individual as certain property value of another individual. For instance, an object property `OrCA:hasContributor` can be used to connect the individual `OrCA:design` to the individual that represents its contributor, namely `OrCA:webdesigner`. In the following, the OrCA ontology is constructed incrementally, accompanied by introductions of the underlying OWL concepts. A comprehensible overview and deeper explanation of these and further OWL concepts is given in [10]. From several syntax variants that are available for OWL, this work utilizes the *Functional-Style Syntax* (FSS) that is also the basis for the structural specification of OWL in [16].

## 4.2 Classes

In FSS, a named class is declared by an *axiom*, e. g. `Declaration(Class(OrCA:Deliverable))`. The colon-separated prefix `OrCA` refers to a separate ontology file that contains all OrCA specific elements and serves as namespace. Likewise, all GRL elements are prefixed by `GRL`. This is explained in detail in the context of the ontology file import structure in subsection 4.7. Furthermore, we follow the conventions in [9, S. 9] and [6, S. 17] to begin a class name with a capital letter and to use *CamelCase*. We construct a corresponding OWL class for each GRL and OrCA class. This ontology is intended to enable the functional evaluation of OrCA models. Therefor, it is limited to an essential subset of the GRL concepts that is relevant for this purpose.

Particularly, the OrCA predicates are represented by an `OrCA:Predicate` class. Furthermore, we declare three additional subclasses `OrCA:EntityPredicate`, `OrCA:IndicatorPredicate` and `OrCA:DataPredicate` to represent all predicate categories. Likewise, OrCA's Contributors are a subclass of the GRL actor element, the Realizations are specializations of the GRL tasks and the Deliverables are specific GRL resources, respectively. In an ontology this can be expressed by means of *subclass* axioms. A subclass is simply defined as a subset of the individuals of another class. Moreover, a class is allowed to be the subclass of multiple other classes. As an example, we define the Contributor class as subclass of the GRL actor with the following axiom: `SubClassOf(OrCA:Contributor GRL:Actor)`. Besides, the OWL specification defines all classes to be implicitly subsets of the top-level class owl:Thing.

Each user-defined predicate type like has skill or has cost is represented by a respective class in a separate namespace, e. g. `HasSkill`. It is declared to be a subclass of `OrCA:EntityPredicate`, `OrCA:IndicatorPredicate` or `OrCA:DataPredicate`, respectively and has the same class name, for example `HasSkill:EntityPredicate`. To integrate user-defined predicate types in a uniform manner, we specify the following generic axioms. They contain the placeholders $e$ for entity, $i$ for indicator and $d$ for data predicate type identifiers as defined in subsection 2.4. These axioms have to be added to the ontology, one for each user-defined predicate type; the placeholder is to be substituted by the respective identifier, e. g. $e$=`HasSkill` or $d$=`HasCost`. The entire *taxonomy*, i. e. the hierarchy of classes and subclasses, is diagrammed in Fig. 8.

*For each entity predicate identifier* $e$:
`Declaration(Class(`$e$`:EntityPredicate))`
`SubClassOf(`$e$`:EntityPredicate OrCA:EntityPredicate)`

*For each indicator predicate identifier* $i$:
`Declaration(Class(`$i$`:IndicatorPredicate))`
`SubClassOf(`$i$`:IndicatorPredicate OrCA:IndicatorPredicate)`

*For each data predicate identifier* $d$:
`Declaration(Class(`$d$`:DataPredicate))`
`SubClassOf(`$d$`:DataPredicate OrCA:DataPredicate)`

As mentioned before, OWL classes are allowed to overlap. E. g., without additional declarations, some OWL individual could be a member of both OWL classes, `OrCA:Contributor` and `OrCA:Deliverable`. Of course, such a duplicate class membership was pointless and violated

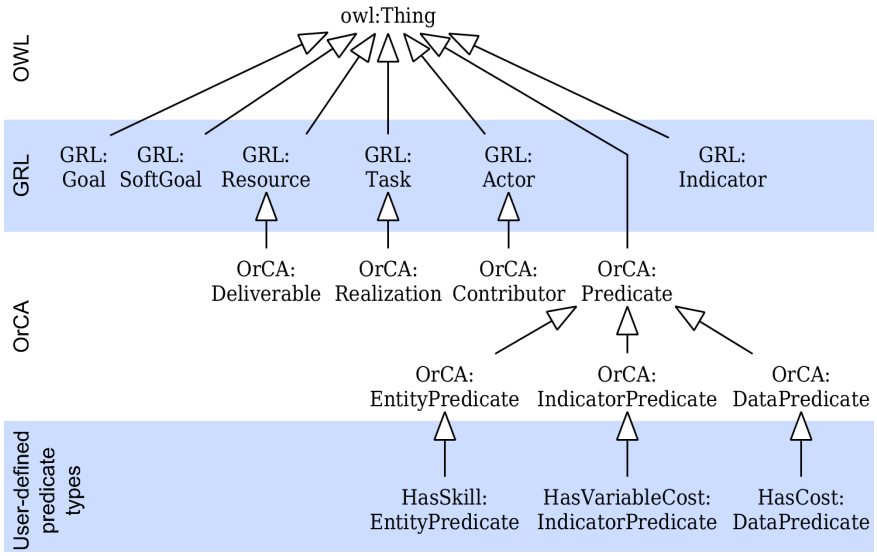


Figure 8: The taxonomy, including the GRL and OWL classes and user-defined predicate type classes.

the three-layered structure of OrCA. To prevent overlapping classes in OWL, they can be declared as disjoint. Hereby, neither the affected classes nor their subclasses can share the same individuals. To preserve the valid structure of OrCA models, we declare all classes to be disjoint that have a common super class by means of the two following axioms:

```
DisjointClasses(GRL:Goal GRL:Softgoal GRL:Resource GRL:Task
                GRL:Actor GRL:Indicator OrCA:Predicate)
DisjointClasses(OrCA:EntityPredicate OrCA:IndicatorPredicate
                OrCA:DataPredicate)
```

Likewise, all user-defined entity, indicator and data predicate classes have to be declared as disjoint, respectively. Therefore, we provide two more generic axioms. A list of placeholders $e_1,\ldots,e_n$ is used, to represent the identifiers of the entire set of user-defined entity predicate types, likewise $i_1,\ldots,i_n$ and $d_1,\ldots,d_n$ represent all indicator and data predicate type identifiers, respectively.

*For all entity predicate identifiers* $e_1,\ldots,e_n$
`DisjointClasses(`$e_1$`:EntityPredicate ...` $e_n$`:EntityPredicate)`

*For all indicator predicate identifiers* $i_1,\ldots,i_n$
`DisjointClasses(`$e_1$`:IndicatorPredicate ...` $e_n$`:IndicatorPredicate)`

*For all data predicates identifiers* $d_1,\ldots,d_n$
`DisjointClasses(`$d_1$`:DataPredicate ...` $d_n$`:DataPredicate)`

## 4.3 Object Properties

Relations among OWL individuals can be modeled by *object properties*. Hence, each GRL and OrCA link is represented by a corresponding object property. We follow the conven-

tion in [6, S. 26] to begin all object property identifiers with a lower case letter and to use *mixedCase*. Furthermore, we enhance the readability by prefixing all identifiers by one of the verbs `is` or `has` and appending appropriate prepositions. For example, the GRL dependency links are implemented by an object property `GRL:hasDependencyOn`. It is declared by `Declaration(ObjectProperty(GRL:hasDependencyOn))`. The connections from OrCA Deliverables in the bottom layer to Contributors in the top layer are an OrCA specific derivation of these dependency links. They are represented by two separate object properties `OrCA:hasRealization` and `OrCA:hasContributor`. The connection from subject to predicate in an OrCA statement is implemented by `OrCA:hasPredicate`, entity predicates are linked to their object by `OrCA:hasEntity`. A connection from a data predicate to an indicator is represented by `OrCA:hasIndicator`. Like classes, object properties can be arranged hierarchically. For example, the axiom `SubObjectPropertyOf(OrCA:hasRealization GRL:hasDependendencyOn)` declares the set of `OrCA:hasRealization` links to be a subset of the dependency links. Besides, all object properties are implicitly subsets of the `owl:topObjectProperty`.

For each user-defined predicate type, the ontology is extended by a specific variant of `hasPredicate` in the respective namespace, e. g. `HasSkill:hasPredicate`. It is declared as sub-object property of `OrCA:hasPredicate`. This is done by using the following generic axioms:

*For each predicate identifier* $p$:
`Declaration(ObjectProperty(`$p$`:hasPredicate))`
`SubObjectPropertyOf(`$p$`:hasPredicate OrCA:hasPredicate)`

The object property hierarchy is shown in Fig. 9. For the sake of brevity, it is restricted to an essential subset of the GRL concepts. For example, further sub-object properties to represent different contribution link types are omitted in this work.

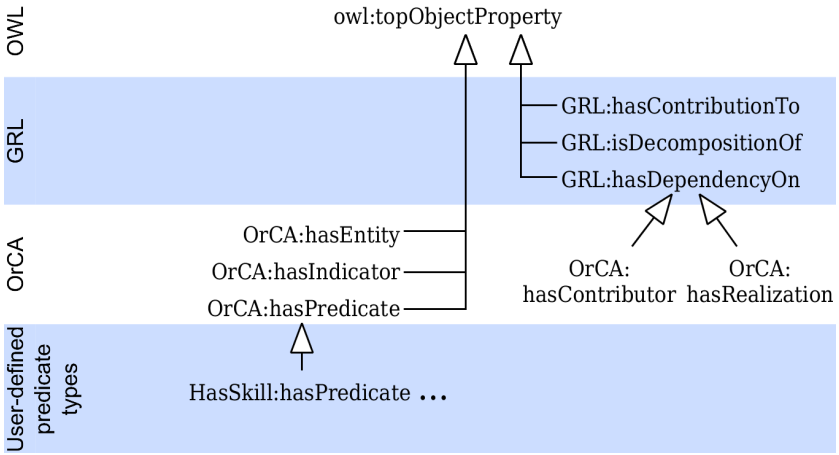


Figure 9: The hierarchy of object properties.

OWL allows for the axiomatic declaration of various mathematical object property characteristics like *symmetry* or *transitivity*. For the sake of brevity, this work confines to the declaration of the *domain* and the *range* of properties. For example, the axiom

`ObjectPropertyRange(OrCA:hasRealization OrCA:Realization)` states that the object property `OrCA:hasRealization` generally links to individuals of the realization class. Analogously, an `ObjectPropertyDomain` axiom defines the possible source class of such a link. In cases in which the range or the domain consists of more than one class, their union is expressed by using the `ObjectUnionOf` keyword. For example, the range of `OrCA:hasPredicate` is set to `ObjectUnionOf(OrCA:EntityPredicate OrCA:IndicatorPredicate OrCA:DataPredicate)`.

The domains of the GRL and OrCA object properties in our ontology are represented as matrix in Table 1. A matrix showing all ranges is contained in Table 2.

Table 1: Overview over the object property domains. A checkmark means, the class in the respective column is part of the domain of the object property in the respective row.

| | GRL:Actor | GRL:Task | GRL:Resource | GRL:Goal | GRL:Softgoal | GRL:Indicator | OrCA:Contributor | OrCA:Realization | OrCA:Deliverable | OrCA:EntityPredicate | OrCA:IndicatorPredicate |
|---|---|---|---|---|---|---|---|---|---|---|---|
| GRL:hasContributionTo | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| GRL:hasDependencyOn | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| GRL:isDecompositionOf | | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| OrCA:hasContributor | | | | | | | | ✓ | | | |
| OrCA:hasRealization | | | | | | | | | ✓ | | |
| OrCA:hasPredicate | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ | | |
| OrCA:hasEntity | | | | | | | | | | ✓ | |
| OrCA:hasIndicator | | | | | | | | | | | ✓ |

Table 2: Overview over the object property ranges.

| | GRL:Actor | GRL:Task | GRL:Resource | GRL:Goal | GRL:Softgoal | GRL:Indicator | OrCA:Contributor | OrCA:Realization | OrCA:Deliverable | OrCA:EntityPredicate | OrCA:IndicatorPredicate | OrCA:DataPredicate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GRL:hasContributionTo | | ✓ | | ✓ | ✓ | ✓ | | ✓ | | ✓ | | |
| GRL:hasDependencyOn | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| GRL:isDecompositionOf | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | |
| OrCA:hasContributor | | | | | | | ✓ | | | | | |
| OrCA:hasRealization | | | | | | | | ✓ | | | | |
| OrCA:hasPredicate | | | | | | | | | | ✓ | ✓ | ✓ |
| OrCA:hasEntity | | | ✓ | | | | | | | | | |
| OrCA:hasIndicator | | | | | | ✓ | | | | | | |

Likewise, the domain and range of the specific variants of `hasPredicate` for each user-defined predicate type must be refined according to its intended purpose. For example, only actors and contributors should be connected to a has skill predicate. Thus, the domain of `HasSkill:hasPredicate` is set to the union of both classes, `ObjectUnionOf(GRL:Actor OrCA:Contributor)`. Additionally, the range of such a specific `hasPredicate` variant can be restricted to the corresponding predicate class `HasSkill:EntityPredicate`. In general terms: For a predicate type with the identifier $p$ let $D(p)$ be a *class expression* that describes the union of all classes that are selected as the domain as specified in subsection 2.4. Then

- the domain of the `hasPredicate` variant of the predicate type identified by $p$ must be set to $D(p)$;
- the range of the `hasPredicate` variant of the predicate type identified by $p$ must be set to the corresponding `EntityPredicate` variant.

Therefore, we specify the following generic axioms:

*For each entity predicate identifier* $e$:
`ObjectPropertyDomain(`$e$`:hasPredicate` $D(e)$`)`
`ObjectPropertyRange(`$e$`:hasPredicate` $e$`:EntityPredicate)`

*For each indicator predicate identifier* $i$:
`ObjectPropertyDomain(`$i$`:hasPredicate` $D(i)$`)`
`ObjectPropertyRange(`$i$`:hasPredicate` $i$`:IndicatorPredicate)`

*For each data predicate identifier* $e$:
`ObjectPropertyDomain(`$d$`:hasPredicate` $D(d)$`)`
`ObjectPropertyRange(`$d$`:hasPredicate` $d$`:DataPredicate)`

## 4.4 Data Properties

The *data property* `OrCA:hasData` connects data predicate individuals to a concrete value that represents an OrCA data literal. It is declared by `Declaration(DataProperty(OrCA:hasData))`. The domain of `OrCA:hasData` is the class `OrCA:DataPredicate`. The range must be specified for each data predicate type and is either one of the two exemplary data types `xsd:integer` or `xsd:string`. Therefore, a predicate type specific variant of `OrCA:hasData` is declared in the respective namespace as *sub-data property* of `OrCA:hasData`. Its domain is set to the corresponding predicate class and its range is set to the intended data type. For example, to restrict has cost predicates to integer values, the data property `HasCost:hasData` is added to the ontology, its domain is set to `HasCost:DataPredicate` and its range is set to `xsd:integer`. In general terms: For a data predicate type with the identifier $d$ let $R(d)$ be the data type that is selected as the range according to subsection 2.4. Then

- the domain of the `hasData` variant of the predicate type identified by $d$ must be set to the corresponding `DataPredicate` variant;

- the range of the `hasData` variant of the predicate type identified by $d$ must be set to $R(d)$.

Therefore, we specify the following generic axioms:

*For each data predicate identifier* $d$:
`Declaration(DataProperty(`$d$`:hasData))`
`SubDataPropertyOf(`$d$`:hasData OrCA:hasData)`
`DataPropertyDomain(`$d$`:hasData` $d$`:DataPredicate)`
`DataPropertyRange(`$d$`:hasData` $\mathrm{R}(d)$`)`

## 4.5 Individuals

Basing on the terminology of classes and properties, the next step is to record a particular model instance, including all model elements and their relationships among each other. All elements in an OrCA model can be represented by *named individuals* in OWL. To enhance readability, all individuals are declared in the namespace of their class. For example, the axiom `Declaration(NamedIndividual(OrCA:developer))` declares an individual that represents the developer. The assignment of `OrCA:developer` to its respective class is declared by the axiom `ClassAssertion(OrCA:Contributor OrCA:developer)`. Predicates in an OrCA diagram are anonymous instances of their predicate type. To distinguish the corresponding OWL individuals, they are uniformly named "predicate" and suffixed by a number, e. g. `HasCost:predicate1`. The following table lists all classes and their respective individuals according to the example in Fig. 5:

Table 3: Class assertions according to the example in Fig. 5.

| | |
|---|---|
| `OrCA:Contributor` | `OrCA:developer`, `OrCA:webdesigner` |
| `OrCA:Realization` | `OrCA:implement`, `OrCA:test`, `OrCA:design` |
| `OrCA:Deliverable` | `OrCA:webapplication` |
| `GRL:Softgoal` | `GRL:highUsability` |
| `GRL:Resource` | `GRL:php` |
| `HasSkill:EntityPredicate` | `HasSkill:predicate1` |
| `IsBasedOn:EntityPredicate` | `IsBasedOn:predicate1` |
| `HasBudget:EntityPredicate` | `HasBudget:predicate1`,`HasBudget:predicate2` |
| `HasCost:EntityPredicate` | `HasCost:predicate1`,`HasCost:predicate2` |

## 4.6 Object and data property assertions

*Object property assertions* define the actual connections among individuals. An object property can be interpreted as a relation, containing all pairs of individuals that it connects. E. g., the pair (`HasSkill:predicate1`, `GRL:php`) is connected by the object property `OrCA:hasEntity`. In FSS this is declared by `ObjectPropertyAssertion(OrCA:hasEntity HasSkill:predicate1 GRL:php)`. Likewise, *data property assertions* define the actual assignments of concrete XSD

values to individuals. The table below lists all data and object properties in combination with their respective pairs of individuals or concrete values, respectively, according to the example case in Fig. 5:

Table 4: Object and data property assertions according to the example in Fig. 5.

| | | |
|---|---|---|
| `OrCA:hasRealization` | (`OrCA:webapplication`, | `OrCA:implement`) |
| | (`OrCA:webapplication`, | `OrCA:test`) |
| | (`OrCA:webapplication`, | `OrCA:design`) |
| `OrCA:hasContributor` | (`OrCA:implement`, | `OrCA:developer`) |
| | (`OrCA:test`, | `OrCA:developer`) |
| | (`OrCA:design`, | `OrCA:webdesigner`) |
| `GRL:hasContributionTo` | (`OrCA:design`, | `GRL:highUsability`) |
| `HasBudget:hasPredicate` | (`OrCA:developer`, | `HasBudget:predicate1`) |
| | (`OrCA:webdesigner`, | `HasBudget:predicate2`) |
| `HasCost:hasPredicate` | (`OrCA:implement`, | `HasCost:predicate1`) |
| | (`OrCA:test`, | `HasCost:predicate2`) |
| `HasSkill:hasPredicate` | (`OrCA:developer`, | `HasSkill:predicate1`) |
| `IsBasedOn:hasPredicate` | (`OrCA:webapplication`, | `IsBasedOn:predicate1`) |
| `OrCA:hasEntity` | (`HasSkill:predicate1`, | `GRL:php`) |
| | (`IsBasedOn:predicate1`, | `GRL:php`) |
| `HasBudget:hasData` | (`HasBudget:predicate1`, | `"7000"^^xsd:integer`) |
| | (`HasBudget:predicate2`, | `"2500"^^xsd:integer`) |
| `HasCost:hasData` | (`HasCost:predicate1`, | `"4000"^^xsd:integer`) |
| | (`HasCost:predicate2`, | `"3200"^^xsd:integer`) |

## 4.7 Ontology import structure

OWL is designed to formalize distributed knowledge in ontological structures in the *Semantic Web*. Therefore, ontologies and their elements are referenced by an IRI and can be imported from remote locations. This mechanism enables a modular ontology structure that holds all GRL and OrCA specific elements in separate ontology parts (see Fig. 10):

1. The basis is the ontology file /metamodel/GRL/primitive.owl that contains the GRL taxonomy, i. e. classes and object properties that represent all GRL elements. Examples are the `GRL:Actor` class or the `GRL:hasContributionTo` object property.

2. The GRL taxonomy is imported by /metamodel/OrCA/primitive.owl that extends it by OrCA specific classes and properties, e. g. the `OrCA:Contributor` class and the `OrCA:hasContributor` object property.

3. Specific classes and properties for each user-defined predicate type are contained in a respective ontology file, e. g. /userextension/HasSkill/primitive.owl that imports the OrCA taxonomy.

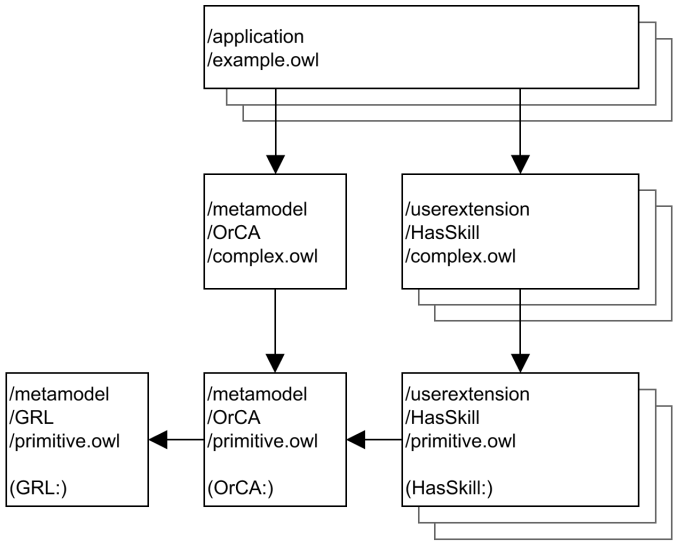


Figure 10: The import structure of the OrCA ontology. The ontology files in the primitive layer are assigned to the prefixes `GRL`, `OrCA` and `HasSkill`, respectively.

4. The term "primitive" in these file names refers to a concept for a modular, three-layered ontology structure that is proposed in [3]. A primitive layer contains only the taxonomy, i. e. class and object property declarations. Based on this, a complex layer adds further axioms that describe their characteristics and properties. On top, a third layer adds axioms that are specific to the particular case of application. We adopted this concept to modularize the OrCA ontology. Thus, respective ontology files for the complex layer /metamodel/OrCA/complex.owl and /userextension/HasSkill/complex.owl import the files of the primitive layer to add further axioms to the underlying taxonomy like property domains and ranges.

5. Finally, the complex OrCA ontology and the complex ontologies for all predicate types that are used in a specific case of application are imported in the top layer, e. g. /application/example.owl. Hence, this top layer ontology integrates all GRL and OrCA elements plus all elements for user-defined predicate types. Moreover, additional classes for analysis purposes are declared in this layer.

To refer to its elements, an imported ontology can be assigned a specific namespace prefix. It is separated from the name of an element of this ontology by a colon, e. g. `GRL:Actor`. Thereby, it is easy to see that this class represents a GRL specific concept. Obviously, an appropriate selection of prefixes can improve the readability and comprehensibility of ontology axioms and evaluation rules. As mentioned above, the entire taxonomy including all classes and properties is declared in the primitive layer. Thus, only this layer contains elements that should be referred by an expressive prefix. As depicted in Fig. 10, we chose `GRL`, `OrCA` and a predicate type's identifier as prefix for the respective primitive layer.

# 5 Analysis in OWL: capabilities and limits

An important objective of this work is a stepwise evaluation of OrCA models. In the following, we evaluate the native means for such an analysis that are provided by OWL. Furthermore, their respective limits are pointed out by our example cases.

## 5.1 Step I: Inference

Beginning with analysis *step I*, we illustrate how the reasoning capabilities of OWL can be employed to infer new information. In particular, we utilize *class expressions*. Their descriptive attributes can be used to "pick out" certain individuals with specific characteristics. For example, the class expression

```
EquivalentClasses(
    example:RealizationWithCost
    ObjectIntersectionOf(
        OrCA:Realization                    All realizations
        ObjectSomeValuesFrom(             that are connected
            HasCost:hasPredicate            by hasPredicate
            HasCost:DataPredicate     to a HasCost predicate
)))
```

defines a class named `RealizationWithCost` that contains all realizations that are connected to a has cost predicate as required for a detection of exceeded budgets. Or put another way, all realizations that do not lack information about their cost. This is an intermediate step to search for missing cost data. In general, similar classes can be added to the ontology by an user according to the particular domain and to individual evaluation objectives. A reasoner like *Pellet* is capable of the interpretation of these class expressions and will compute the actual class memberships. In our example case, `OrCA:implement` and `OrCA:test` are identified as members of `example:RealizationWithCost` but `OrCA:design` is not.

## 5.2 Step II: Structural consistency checks

This evaluation step examines structural inconsistencies. For example, the OrCA ontology declares restrictions to the domain and the range of the object properties. According to its range definition, the target of an `OrCA:hasEntity` property must be an element of the `GRL:Resource` class. Intuitively one would assume a reasoner to report an error or inconsistency, if this range axiom was violated. But this is interfered by OWL's semantic nature that does not adopt the Open World Assumption (OWA). The OWA generally assumes the possible incompleteness of the ontology against the reality. I. e. the missing of a fact in the ontology will not implicitly be interpreted as its negation but only as an absence of information in the model. Unfortunately, this can lead to counter-intuitive results in the context of a structural analysis. Assume the target of an `OrCA:hasContributor` link to be incorrectly set to the `OrCA:webapplication` individual of the class `OrCA:Deliverable`. This "range violation" alone is not sufficient for a reasoner to detect an inconsistency, as one could

expect. Instead, following the OWA, a reasoner will infer that `webapplication` is a member of the class `OrCA:Contributor` that is defined as range – even if this class assertion is not contained in the ontology. In other words, if the target of an object property is out of range, then it is added to the range:

```
    ObjectPropertyRange(OrCA:hasContributor OrCA:Contributor)
∧   ObjectPropertyAssertion(  OrCA:hasContributor
                              OrCA:design OrCA:webapplication)
⇒   ClassAssertion(OrCA:Contributor OrCA:webapplication)
```

Consequently, the webapplication is identified as member of both classes, `OrCA:Contributor` and `OrCA:Deliverable`. Of course, such a duplicate class membership contradicts the three-layered OrCA meta model. Therefore, this is prevented by the axioms in subsection 4.2 that explicitly declare all sibling classes to be disjoint. Hence, in a second step, a reasoner will detect an inconsistency as expected:

```
  ClassAssertion(OrCA:Contributor OrCA:webapplication)
∧ ClassAssertion(OrCA:Deliverable OrCA:webapplication)
∧ DisjointClasses(OrCA:Contributor OrCA:Deliverable)   ↯
```

In this particular case, declaring all sibling classes to be disjoint solves the issues that arise from the OWA. The pairwise disjunction of all GRL and OrCA classes is possible because our meta model contains a finite and relative small number of classes. However, some objectives of our analysis are impeded by the OWA to such an extent that they cannot be accomplished by using solely native OWL concepts. Such a case is illustrated in the next evaluation step.

## 5.3 Step III: Data completeness checks

We want to detect all individuals that are member of the Realization class but that are *not* connected to a has cost predicate. In step I, this was prepared by the declaration of the class `example:RealizationWithCost` that comprises all realizations that *are correctly connected* to such a predicate. Next, we declare a second class `RealizationWithoutCost` that contains all realizations that *are not* connected to a `has cost` predicate as required. Therefore, we make use of the `ObjectComplementOf` concept that describes the complement of a class:

```
EquivalentClasses(
    example:RealizationWithOutCost
    ObjectIntersectionOf(
        OrCA:Realization                          All realizations
        ObjectComplementOf(                       that are not
            ObjectSomeValuesFrom(                 connected
                HasCost:hasPredicate              by hasPredicate
                HasCost:DataPredicate             to a HasCost predicate
))))
```

This class description comprises all realizations that miss a cost predicate. Clearly, this is the case for the design task in Fig. 5. Unfortunately, the Open World Semantic obstructs this conclusion by assuming an incomplete ontology: `OrCA:design` *could* be connected to a cost predicate, although the ontology does not reflect this. Thus, the merely absence of cost information is not sufficient to detect a lack of information, so an appropriate data quality check for this purpose cannot be constructed by means of native OWL concepts.

## 5.4 Step IV: Checks for functional conflicts

The analysis under the OWA is limited for the same reasons in *step IV*. While it is possible to compute all contributors that have all of the skills that required for the realization of the deliverables they are responsible for, we cannot infer negative information like *contributor* webdesigner *needs skill* php *but is not connected to this by any* has skill *predicate*.

A completely different issue is the evaluation of arithmetic expressions. To check a contributor for exceeded budgets, the costs of its realizations have to be totaled. I. e., a contributors budget has to be compared the sum of all concrete values that

- ... are connected by `HasCost:hasData` to a has cost predicate that
- ... is connected by `HasCost:hasPredicate` to a realization that
- ... is connected to this particular contributor by `OrCA:hasContributor`.

A suitable mechanism for the computation of such an arithmetic aggregate function is not provided by OWL, so unfortunately native OWL concepts are not sufficient in this case, too.

## 5.5 Solution possibilities

Our analysis checks in step III and IV can be viewed as a set of *integrity constraints*. Without CWA and UNA, the OWL semantic is hardly capable of formalizing such integrity constraints to assure the model's consistency or to report functional problems. This is a known problem that is put in a nutshell in [25]: "*What triggers a constraint violation in closed world systems leads to new inferences in standard OWL systems*". In [15] the OWA semantic is compared to the CWA based semantic of relational databases. It is worked out that OWL offers no possibility "*to formalize database-like integrity constraints*". To eliminate these shortcomings, the authors introduce an *extended DL knowledge base* that extends the ontology by a finite set of integrity constraint axioms. These are evaluated separately under the CWA. Hence, the semantic of the ontology and its expressiveness are preserved. This approach is adopted in [25] where integrity constraints are translated to *SPARQL*-Queries [5]. In the following section, our work applies these principles to facilitate the implementation of the analysis steps III and IV with the OrCA ontology.

# 6 Utilizing SPARQL-Queries

We have considered several methods to overcome the deficiencies of the analysis in OWL ontologies as described in the previous chapter. The *Semantic Web Rule Language* (SWRL)

[7] enables the application of rules according to the *Datalog Rule Markup Language* in OWL. Based on this, the *Semantic Query-Enhanced Web Rule Language* (SQWRL) [19] allows to formalize queries against an OWL ontology as SWRL rules that are evaluated under CWA and UNA and that even provide arithmetic aggregate functions. However, both approaches have not met our requirements according to section 3 as neither SWRL nor SQWRL have become established technologies. Another option is the transformation of the ontology to a relational database (RDB). In [27] entire ontologies are recorded in a RDB to utilize the scalability capabilities of relational database systems for the reasoning. The inference rules are formalized as *SQL* statements. Likewise, our integrity constraints could be formalized by SQL statements that are interpreted under CWA and UNA and that support powerful arithmetic capabilities, as well. This approach combines the expressiveness of ontologies and relational databases, though it requires the implementation of an automated transformation between both. Furthermore, it constraints the user to work with two separated and heterogeneous systems in parallel. Eventually, we have chosen to adopt the approach in [25] and to build our analysis upon integrity constraints formalized as *SPARQL*-queries. SPARQL is a RDF query language. Since OWL is technically based on the RDF syntax, SPARQL-queries can operate on the ontology directly without any previous transformation. The syntax is very similar to SQL; an overview is given in [5]. The following simple query selects all individuals of the class `OrCA:Realization`. Precisely, it results in a table with only one column for the possible values of the placeholder `?r` that match to the search pattern in the `WHERE` clause: `SELECT ?r WHERE {?r a OrCA:Realization}`.

## 6.1 Analysis under the CWA

The queries are evaluated under the CWA, thus enabling the consideration of negative facts. In the following, this is applied to complete our analysis steps III and IV. The keyword `MINUS` removes all elements of the result set that fit into the subsequent search pattern. The class `RealizationWithCost` includes all realizations connected to a `HasCost` predicate. The SPARQL-query in Table 5 selects all realizations and subsequently removes those being an element of `RealizationWithCost`. The result contains all realizations without any cost information. As expected, in the case of our example in Fig. 5 the result contains only the single individual `OrCA:design`.

Table 5: SPARQL-query for the detection of missing cost data

| | |
|---|---|
| `SELECT ?r` | *Select all individuals* `?r` |
| `WHERE {?r a OrCA:Realization .` | *. . . in Realization* |
| `MINUS {?r a example:RealizationWithCost}}` | *. . . but not in RealizationWithCost* |

Using `FILTER NOT EXISTS` the result set can be reduced to all values that do *not* match to the subsequent search pattern. The query in Table 6 performs a check on step IV and detects a functional conflict by computing all pairs of contributors and skills where the contributor takes part in the realization of a deliverable that requires a specific skill; next the result set is reduced to those cases in which the required skill is *not* connected to the

contributor by an `HasSkill` predicate. In other words: the query selects all contributors that lack a required skill. Applied to the example case in Fig. 5, the result is the single pair (`OrCA:webdesigner`,`GRL:php`), as expected.

Table 6: SPARQL-query for the detection of missing skills

```
SELECT DISTINCT ?c ?skill WHERE {    Select all contributors ?c and skills ?skill
  ?d OrCA:hasRealization ?r .              ... where ?c participates in ?d by ?r
  ?r OrCA:hasContributor ?c .
  ?d IsBasedOn:hasPredicate ?pIBO .   ... and ?d is built with technology ?skill.
  ?pIBO OrCA:hasEntity ?skill
FILTER NOT EXISTS {                          Reduce the result to all ?c without
  ?c HasSkill:hasPredicate ?pHS .            ... skills for the technology ?skill
  ?pHS OrCA:hasEntity ?skill}}
```

## 6.2 Analysis with arithmetic expressions

To get back to the analysis of overrun budgets in step IV, we query for all contributors with the sum of their realization costs exceeding their budget. For this purpose, we utilize the SQL-like capabilities of SPARQL and group the result set by contributor to aggregate the values. Like its SQL pendant, the `GROUP BY` keyword groups the result by the specified columns. The sum of the grouped data values is calculated by `SUM()`. These aggregated results can be filtered by a `HAVING` statement. The query in Table 7 selects all contributors plus their respective budget value as determined by a has budget predicate; this is combined with each contributors' respective realization costs determined by has cost statements; these values are grouped by each contributor and totaled; these total costs of each contributor are then compared to the respective budget to determine the final result set. Applied to our example in Fig. 5 with cost values of 4000 and 3200 and a budget of 7000, the expected result is the triple (`OrCA:developer`,`"7200"^^xsd:integer`,`"7000"^^xsd:integer`).

Table 7: SPARQL-query fo the detection of exceeded budgets

```
SELECT ?c (SUM(?cost) AS ?totalCost) ?budget WHERE {
  ?c HasBudget:hasPredicate ?pHB .     Contributor ?c is be assigned to ?budget.
  ?pHB HasBudget:hasData ?budget .
  ?r OrCA:hasContributor ?c .                  Realization ?r is assigned to ?c
  ?r HasCost:hasPredicate ?pHC .                ... and ?r has the cost ?cost.
  ?pHC HasCost:hasData ?cost}
GROUP BY ?c ?budget                  Group the result by contributor and budget
HAVING (SUM(?cost)>?budget)   ... and filter those with costs exceeding the budget.
```

Further possibilities for the application of integrity rules on OrCA models are illustrated in our previous work in [13]. We extracted a set of rules to support the building of successful

teams according to the *Belbin team role model*. These rules can be formalized as SPARQL queries to reveal unfavorable team compositions.

# 7 Related work

There is existing work that addresses the ontological foundation of the language *i** that forms the basis of GRL. In [4] an extract of the i* concept is transferred exemplarily to the *Unified Foundational Ontology*. The objective of this work is not an analysis but a well-defined specification of the underlying semantic of i*. This enables a common semantic basic for the variety of heterogeneous i* advancements and derivations. A similar goal is achieved in [18], where principles of the *Model Driven Engineering* are adopted to define three modeling layers in OWL. The foundation is the OWL meta-model itself. Based on this, a meta ontology named *OntoiStar* is constructed for the purpose of the integration of the various i* dialects. This is again the basis for the OntoiStar model's instances. The approach is advanced in [17], where the possibilities of querying and reasoning on the ontology are underlined but not addressed explicitly. The mapping of i* language elements to OWL concepts according to these works resembles our approach, but focuses on the comprehensive integration of complete i* variants in a meta ontology to facilitate their interoperability. In contrast, our work concentrates on an essential subset of GRL elements that is although important to the automation of the model analysis. Generally, we prefer a small ontology that is limited to relevant GRL and OrCA concepts to simplify the task of building custom integrity rules. Nevertheless, in a future advancement of our work the integration of both approaches by utilizing this elaborate meta ontology as a foundation for our analysis framework is conceivable. In [26] those i* ontologies are combined with external ontologies to annotate the models with domain-specific knowledge. We share the requirement for an incorporation of domain specific ontologies. However, we do not focus on a further elaboration of a special integration mechanism but resort to OWL's import capabilities.

The framework in [21] enables the monitoring and optimization of processes based on key performance indicators (KPI). Indicators act as variables that are bound to information sources like data warehouses that provide actual measured values. A measured value is mapped to the satisfaction value of the affected indicator element and is taken into account by the native GRL evaluation mechanisms. These mechanisms are complemented by our rule-based approach that enables a user to define a set of custom predicate types and to formalize a set of integrity rules, to check a model for complex and domain specific problems like exceeded budgets. However, OrCA predicates are binary and do not provide a continuous effect of concrete values on satisfaction values. For that purpose, indicators and the GRL evaluation mechanisms are the better choice. Nevertheless, we introduced indicator predicates, to enrich a model with statements about the semantic relationships between indicators and other elements. The indicator concept is part of the last URN specification document [8, Sec. 7.6]. [22] presents an approach for the aggregation of KPIs. The actual values of a set of KPIs serve as input parameters for a mathematical formula that calculates their combined effect on a higher level KPI. This approach integrates with the native GRL evaluation mechanisms and is intended to enable a much more precise modeling of the interdependencies between KPIs.

A similarity to our approach is that users can specify formulas to aggregate sets of values. A difference arises from the usage of the calculated results: In [22] the results are evaluated based on the GRL evaluation mechanisms whereas OrCA uses the aggregated results as comparative values in user-defined integrity rules. To provide a minimal and consistent system of KPIs, [2] constructs a KPI ontology. This approach is based on OWL and captures KPIs and their mathematical definitions. The purpose of this ontology is to serve as a KPI reference.

A completely different approach to the analysis of i* models is illustrated in [11]: the i* models are translated into a set of *Prolog* facts. Afterwards, the detection of possible problems can be done completely in Prolog. The examples in this work concentrate more on a syntactical level, e. g. the validity of the relationships between elements is checked. This can be classified as analysis according to step II or III in section 3.

As mentioned above, the deficiencies of OWL regarding the CWA and UNA are addressed in detail in [15]. This work illustrates the counter-intuitive results of a reasoner when OWL is used to define database-like integrity constraints. The provided solution is an extension of OWL by a set of integrity constraints that are specified in OWL but evaluated separately under a different semantic. This is adopted in [25], where operators and rules are introduced which are interpreted under the CWA to assure the model integrity, for example to check uniqueness and typing constraints. The implementation is done with SPARQL. We applied these concepts to facilitate our analysis steps III and IV.

# 8 Conclusion

OrCA is an approach to capture the organizational context of software engineering projects. It extends the GRL by predicates that enable the creation of statements to describe the semantic of the relationships between model elements. The purpose of a predicate is stated by its predicate type. We distinguish three categories of predicate types: Entity predicate types connect an element to a GRL resource element, indicator predicates have a GRL indicator as object and data predicate connect to a concrete value in form of an OrCA data literal. Thus, our approach allows for both: Statements that concern informational or logical entities and statements that concern concrete values. To adapt the framework to an user's specific domain, custom predicate types can be added. To preserve the understandability, OrCA diagrams are split into a set of views, each of them showing only a subset of predicate types.

Furthermore, we transferred these OrCA models to an OWL ontology. This provides a well-defined semantic foundation. Inference and consistency checks can be accomplished under the OWA by means of a common reasoner. Additional constraints can be formalized as SPARQL queries to verify the data completeness and to detect functional problems basing on the CWA. We attained our targets for an evaluation as defined in section 3:

1. Arithmetic functions and aggregate functions are supported. This has been illustrated by a check for overrun budgets in an example case.

2. Exemplary checks for missing cost predicates and missing skills demonstrated the concepts for an evaluation of logical expressions.

3. The set of predicate types and views can easily be extended to adapt the meta-model to a specific application context. Custom SPARQL-queries can be constructed as needed in a particular project to factor the custom predicates into the evaluation.

4. We restricted our approach to the use of well-established languages to ensure a comprehensible framework that is easy to fit to the specific analysis requirements that arise from a particular project context.

# References

[1] Conway, M.E.: How Do Committees Invent (1968), http://www.melconway.com/research/committees.html

[2] Diamantini, C., Genga, L., Potena, D., Storti, E.: Collaborative Building of an Ontology of Key Performance Indicators. In: On the Move to Meaningful Internet Systems: OTM 2014 Conferences. pp. 148–165 (2014)

[3] Dumontier, M., Villanueva-Rosales, N.: Three-Layer OWL Ontology Design. In: Second international workshop on modular ontologies (WoMO) (2007)

[4] Franch, X., Guizzardi, R.S.S., Guizzardi, G., López, L.: Ontological Analysis of Means-End Links. In: iStar 2011 – Proceedings of the 5th International i* Workshop. pp. 37–42 (2011)

[5] Harris, S., Seaborne, A.: SPARQL 1.1 Query Language: W3C Recommendation 21 March 2013 (2013), http://www.w3.org/TR/sparql11-query/

[6] Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., Wroe, C.: A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition1. 2. The University of Manchester (2009)

[7] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML: W3C Member Submission 21 May 2004 (2004), http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[8] International Telecommunication Union: Z.151 User requirements notation (URN) - Language definition (10/2012) (2012-10)

[9] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL plugin: An open development environment for semantic web applications. In: The Semantic Web–ISWC 2004, pp. 229–243. Springer (2004)

[10] Kuba, M.: OWL 2 and SWRL Tutorial (2012), http://dior.ics.muni.cz/~makub/owl/

[11] Laue, R., Storch, A.: A Flexible Approach for Validating i∗ Models. In: iStar 2011 – Proceedings of the 5th International i* Workshop. pp. 32–36 (2011)

[12] Meissner, J., Schulz, F.: Social Team Characteristics and Architectural Decisions: a Goal-oriented Approach. In: iStar 2014 - Proceedings of the Seventh International i* Workshop. CEUR Workshop Proceedings, vol. Vol-1157 (2014)

[13] Meissner, J., Schulz, F., Rossak, W.: Analyse der sozialen Teamstruktur in Softwareprojekten. In: GI-Edition - Lecture Notes in Informatics (LNI). vol. P-239, pp. 171–176 (2015)

[14] Mizoguchi, R., Kozaki, K.: Ontology Engineering Environments. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 315–336. International Handbooks on Information Systems, Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-540-92673-3_14

[15] Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. Web Semantics: science, services and agents on the World Wide Web 7(2), 74–89 (2009)

[16] Motik, B., Patel-Schneider, P.F., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition): W3C Recommendation 11 December 2012 (2012), http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

[17] Najera, K., Martinez Alicia, Perini, A., Estrada Hugo: An Ontology-Based Methodology for Integrating i* Variants. In: iStar 2013 – Proceedings of the 6th International i* Workshop. CEUR Workshop Proceedings, vol. Vol-978, pp. 1–6 (2013)

[18] Najera, K., Perini, A., Martinez Alicia, Estrada Hugo: Supporting i* model integration through an ontology-based approach. In: iStar 2011 – Proceedings of the 5th International i* Workshop. pp. 43–48 (2011)

[19] O'Connor, M., Das, A.: SQWRL: a Query Language for OWL. In: OWLED (2009)

[20] Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C. M., Thompson, Henry S.: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes: W3C Recommendation (2012), http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/

[21] Pourshahid, A., Amyot, D., Peyton, L., Ghanavati, S., Chen, P., Weiss, M., Forster, A.J.: Business process management with the user requirements notation. Electronic Commerce Research 9(4), 269–316 (2009)

[22] Pourshahid, A., Richards, G., Amyot, D.: Toward a goal-oriented, business intelligence decision-making framework. In: E-Technologies: Transformation in a Connected World, pp. 100–115. Springer (2011)

[23] Schulz, F., Meissner, J., Rossak, W.: Tracing the interdependencies between architecture and organization in goal-oriented extensible models. In: Proceedings of the Third Eastern European Regional Conference on the Engineering of Computer Based Systems. pp. 25–32 (2013)

[24] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web 5(2), 51–53 (2007)

[25] Sirin, E., Tao, J.: Towards Integrity Constraints in OWL. In: OWLED (2009)

[26] Vazquez, B., Estrada Hugo, Martinez Alicia, Morandini, M., Perini, A.: Extension and integration of i* models with ontologies. In: iStar 2013 – Proceedings of the 6th International i* Workshop. CEUR Workshop Proceedings, vol. Vol-978, pp. 7–12 (2013)

[27] Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 1239–1248 (2008)

[28] Yu, E.: Modelling strategic relationships for process reengineering. Social Modeling for Requirements Engineering 11 (2011)