

A Graphical Web Tool with DL-based Reasoning Support over Orthogonal Variability Models

Angela Oyarzun¹ and Germán Braun^{1,3}, Laura Cecchi¹, and Pablo Fillottrani^{2,4}

¹UNIVERSIDAD NACIONAL DEL COMAHUE ²UNIVERSIDAD NACIONAL DEL SUR

³Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

⁴Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC)

Abstract Variability Management is one of the most challenging tasks in a Software Product Line (SPL) development. This is reflected in the way software is developed, maintained and extended. Therefore, automatic variability analysis has emerged in order to validate models in early development stages, avoiding affecting derived products quality. In this work, we present *crowd-variability*, a novel graphical tool designed for modelling and validating Orthogonal Variability Models (OVM) using Description Logics (DL)-based reasoning services. We describe the tool and demonstrate the usage of the first prototype along with examples of use. Currently, we are working to release the first beta version of *crowd-variability*.

Keywords: Software Product Lines, Orthogonal Variability Models, Description Logics, Graphical tools for modeling variability

1 Introduction

Software Product Lines (SPL) development involves one of the most challenging and important activities of the domain and application engineering phases, Variability Management [1]. During the first phase, variability is defined, modelled, implemented and validated, while in the second phase, such variability is instantiated in order to derive new products. In this context, Variability Management has a great impact on the way software is developed, extended and maintained. Orthogonal Variability Models (OVM) are used to define variability and relate it to more traditional models like class diagrams, use cases, among others. OVM validation plays a decisive role in detecting faults in a SPL in early development stages that, on the contrary, can limit the quality of the products derived. Due to this fact, automated variability analysis [2, 3, 4, 5] has emerged, which focuses on a set of techniques for translating and validating variability models by considering the anomalies or mismatches these models might contain. Therefore, checking consistency of variability models is a critical problem.

There are several works about automated variability analysis that propose different techniques and methods [5, 6], many of them based on Feature Model analysis, which is another language for variability. However, approaches for validating OVM diagrams have also been undertaken. In this sense, FaMa-OVM [6] is a tool that works with OVM specified using a textual format that are later analysed by SAT-solvers, but as

well as different SAT-based approaches, it presents certain limitations related to more restrictive logics and thus fail to reflect the finer logical structure of variability models.

Therefore, our group presented a proposal [7] focusing on the use of Description Logics (DL) [8] for tackling this issue. DL are a family of decidable logics that have been widely proven to give support to software engineering in order to improve the automated analysis. Our encoding is provided on an OVM-based language from a previously defined SeVaTax framework [9], aiming at analysing variability models properties and deriving products from an SPL. Thus, its models are formalised in DL *ALCCZ* [10] guaranteeing EXPTIME reasoning on SeVaTax models.

In this work, we present *crowd-variability*, a graphical tool for designing, visualising and checking consistency of OVM diagrams. This novel client-server tool provides graphical support for users modelling their diagrams and is integrated with automatic DL-based reasoning [11], using OWLlink [12] as communication protocol for off-the-shelf reasoning systems. Thus, users will visualise their OVMs in an on-line manner while are being modelled and edited. A first Web prototype¹ of this tool already runs on the client-server architecture, supports OWLlink and enables the satisfiability checking process on OVM graphical diagrams. Such client-server architecture is based on an ontology engineering environment, named *crowd*² [13, 14]. *crowd* is a multi-views software for graphical editing of ontologies, using standard languages such as UML, EER and ORM, and DL reasoning in order to give support to ontology engineering tasks.

This work is structured as follows. Section 2 introduces orthogonal variability models and describes each of their components. Section 3 illustrates and details the architecture of this tool. Section 4 presents the first prototype developed together with a simple example of use. Section 5 exposes a preliminary evaluation and discussions. To conclude the paper, section 6 elaborates on final considerations and directions for future works.

2 Orthogonal Variability Models

Variability identifies and models variable characteristics of the products derived from Software Product Lines [1]. Products are applications generated by properly selecting services from a variability model and its constraints. Variability can be modelled as part of traditional models such as class diagrams, use cases, feature models or it can be represented by a separated model. In this regard, OVMs have emerged, which are in charge of defining variability of a SPL, through a separated model and relating it to other diagrams.

OVMs consist of two fundamental elements: variation points and variants. These elements relate to each other through variability dependencies and/or constraint dependencies. Both elements and interactions are depicted in Table 1.

Elements:

- **Variation Point:** it represents a variable object in real world.

¹ <http://crowd.fi.uncoma.edu.ar/ovm/>

² <http://crowd.fi.uncoma.edu.ar>

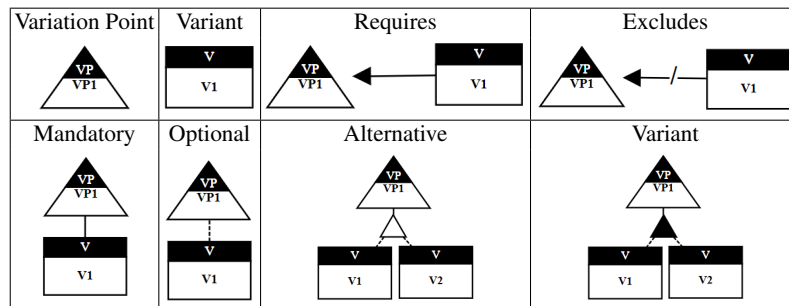


Table 1. OVM Graphical Components

- **Variant**: it denotes how a variation point varies.

Constraint Dependencies:

- **Requires**: it describes a relationship between variation points, variants or a variant and a variation point where the selection of an element requires the selection of the associated element to the first one.
- **Excludes**: it describes a relationship between variation points, variants or a variant and a variation point where the selection of an element excludes the selection of the associated element to the first one.

Variability Dependencies:

- **Mandatory**: it defines the selection of a variation point requires the selection of the associated variants.
- **Optional**: it determines the selection of a variation point can require, but not necessarily, the selection of the associated variants.
- **Alternative**: it specifies the selection of a variation point requires the selection of only one of the associated variants.
- **Variant**: it defines the selection of a variation point requires the selection of at least one of the associated variants.

Example 1. Fig.1 depicts mandatory and alternative variability dependencies modelling the variability of a Remote Robotics Laboratory(RRL), which is defined through a programming language(PL) and the type of robot (R): Frankestito[15, 16] or Multiplo N6 Max[17]. The RRL variation point allows configuring four possible products: {RRL, R, Multiplo N6 Max, PL, Python}, {RRL, R, Multiplo N6 Max, PL, Blockly}, {RRL,R, Frankestito, PL, Python} and {RRL, R, Frankestito, PL, Blockly}. Therefore, as at least one product can be derived, this model is considered *consistent* [7].

3 crowd-variability Overview

crowd-variability is a graphical modelling tool being supported by Universidad Nacional del Comahue and Universidad Nacional del Sur of Argentina. Its main purposes are to

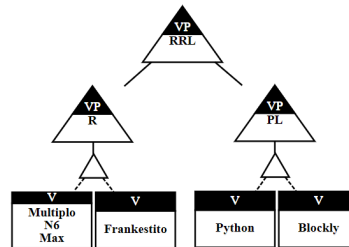


Figure 1. OVM example for a Remote Robotics Laboratory (RRL) modeled in *crowd-variability* tool.

enable users to design and visualise OVM diagrams, as well as validating them. This tool employs complete logical reasoning in order to verify the satisfiability of specifications. So as to apply automated analysis, a semantic definition of all elements of an OVM diagram must be provided. Therefore, each variant, variation point, variability dependency and constraint dependency is translated into a logic-based formalism. Finally, this automatic variability analysis is possible due to the fact that the tool is fully integrated with a powerful logic-based reasoning system acting as a background inference engine.

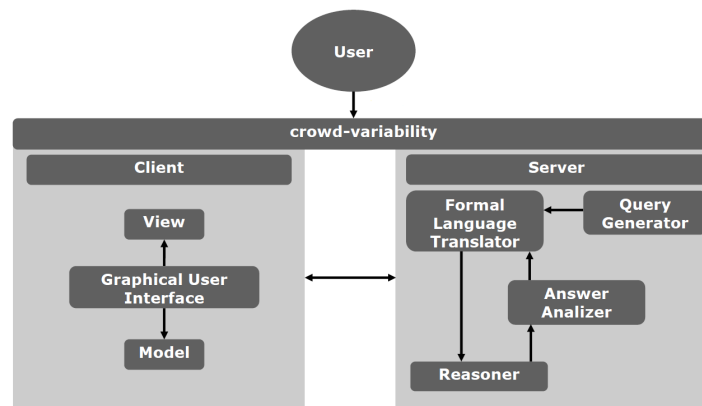


Figure 2. An overview of the *crowd-variability* client-server architecture.

An overview of the client-server architecture is shown in Figure 2. On the client side, users are able to create and edit their models through a graphical user interface (GUI). It offers a set of graphical primitives of the OVM language and functions required for manipulating these diagrams. The server component consists of modules in charge of translating graphical models into a formal language, generating queries to the reasoner, reason over OVM and processing the reasoner output. We explore each com-

ponent in turn.

Client. The Client side is composed of a View, a Model and a Graphical User Interface (GUI). The first two, View and Model are in constant interaction between them. The **View** possesses the visual representation of the OVM diagrams, widgets and events for handling them, while the **Model** contains an abstract representation of each primitive of the OVM language and is able to answer diverse methods that affect it. The corresponding models and its components are created by the external graphical library JointJS³, which is a powerful open source JavaScript. JointJS has been chosen among others because it offers a wide variety of functionalities and focuses on the model that represents each element in user diagrams and the responses to actions on the view. Moreover, it facilitates diagrams translation into a JSON format, uses Backbone⁴ to give structure to Web Applications by providing a model-view-controller architecture and it is capable of expanding its functions by creating and/or adding your own plug-ins.

On the other hand, the *crowd-variability* **Graphical User Interface** is an user-interface mashup [18], which is based on widgets aggregating sets of functionalities in a common graphical space. It is associated to a set of cross-view widgets, such as tool-bars and user management, and a set of specific-view ones, icons, among others. These widgets are orchestrated by the system, through event notifications generated by each one as the user interacts. Last but not least, the user interface component also provide us with functionalities such as user login, model store and interoperability with other tools through JSON importing and exporting in addition to OWLlink files.

Server. The Server side comprises a Formal Language Translator, a Query Generator, an Answer Analyzer and a Reasoner. The **Formal Language Translator** module takes a JSON representation of an user's model and translates it into a formal language and write it to an OWLlink file, which is later sent to the reasoner together with a set of queries. This queries are appended to the file after being generated by a **Query Generator** module, so that the reasoning system can determine the consistency of the model provided by the user. Afterwards, **Reasoner** module connects to an off-the-shelf reasoning system, p.e. Racer [19], for processing the OWLlink document and returns a new OWLlink document including the responses to these queries. Finally, the **Answer Analyzer** processes the reasoner output in order to give the conclusions to users. The output is a new JSON object indicating the consistency of the whole model and each one of this concepts.

4 First Prototype and Example of Use

The client side of *crowd-variability* first prototype runs in a Web browser and has been developed using CoffeeScript, a programming language that compiles into JavaScript, PHP and JointJS library, which has been used to build a new plug-in for OVM. The server side runs in an Apache server and has been developed using PHP. The prototype offers users the possibility of graphically visualising their OVM diagrams, while they are being designed and edited. It also supports the complete set of OVM primitives as

³ <https://www.jointjs.com/opensource>

⁴ <http://backbonejs.org/>

described in section 2. Once built the model, users may request its consistency evaluation. At the moment, *crowd variability* prototype allows OVM diagrams translation into *ALCC DL*. Currently, the integrated reasoner in this tool is Racer[11]. Table 2 shows some variability dependencies, its associated graphics and its *ALCC DL* encoding. For more details refer to [7].

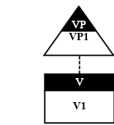
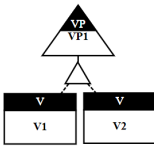
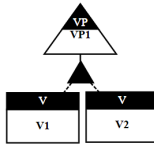
Expression	crowd-variability graphic	<i>ALCC DL</i> encoding
Optional Variability Dependency		$\exists vp1v1^{=1} \sqsubseteq VP1$ $V1 \sqsubseteq \leq 1vp1v1^{-} \sqcap \geq 1vp1v1^{-}$
Alternative Variability Dependency		$VP1 \sqsubseteq \exists vp1v1^{=1} \sqcap \exists vp1v2^{=1}$ $\perp \sqsubseteq \exists vp1v1 \sqcap \exists vp1v2$ $V1 \sqsubseteq \leq 1vp1v1^{-} \sqcap \geq 1vp1v1^{-}$ $V2 \sqsubseteq \leq 1vp1v2^{-} \sqcap \geq 1vp1v2^{-}$
Variant Variability Dependency		$VP1 \sqsubseteq \exists vp1v1^{=1} \sqcup \exists vp1v2^{=1}$ $V1 \sqsubseteq \leq 1vp1v1^{-} \sqcap \geq 1vp1v1^{-}$ $V2 \sqsubseteq \leq 1vp1v2^{-} \sqcap \geq 1vp1v2^{-}$

Table 2. OVM Variability Dependencies

Encoding has suffered some changes from the original one because of pragmatic reasons. In [7], each variant and variation point are represented by singleton concepts and dependencies are represented by roles. However, it does not present any axiom to ensure the singleton property of the classes. Therefore, in order to accomplish that, we have modified the codification by adding cardinalities.

The JSON object to be sent to the reasoner contains information about each element in the diagram. JSON object also includes data about the diagram itself together with options to manage the visualisation of the primitives. For variants and variation points, it records its name and position. For constraint and variability dependencies, it keeps information about its name, origin, targets and type of dependency. In particular, for alternative and variant variability dependencies, it registers its position. Fig. 4 presents the JSON representation and Fig. 5 the respective DL encoding in OWLlink format for the model in Fig. 3.

The *crowd-variability* Editor works on diagrams, which may contain only one OVM diagram. The diagrams are referred as models. Only one diagram is visible at a time and the editing of each of them is independent. The tool does not implement special visual techniques for handling very large diagrams yet.

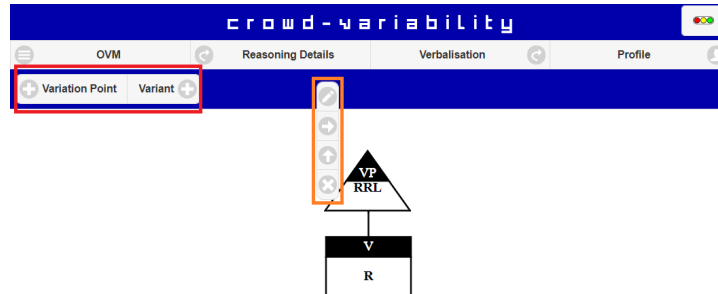


Figure 3. Look and feel of a mandatory dependency example in crowd-variability.

```
{ "metadata": [{" tool": "crowd - variability",
  "owner": "GILIA - Universidad Nacional del Comahue" }],
  "graphops": [{" language": "ovm", "version": "", "displayRequires": "true",
    "displayExcludes": "true", "displayMandatory": "true",
    "displayAlternative": "true", "displayOptional": "true", "displayVariant": "true" }],
  "variation_points": [{" name": "RRL", "position": {" x": 462, "y": 74 } }],
  "variants": [{" name": "R", "position": {" x": 365, "y": 109 } }],
  "constraint_dependencies": [],
  "variability_dependencies": [{" name": "m1", "origin": "RRL", "target": ["R"], "type": "mandatory" }]}
```

Figure 4. crowd-variability JSON representation of model in Figure 3

```
//Mandatory Dependency Domain
//∃rrlr ⊆ RRL
<owl:SubClassOf>
  <owl:ObjectSomeValuesFrom>
    <owl:ObjectProperty IRI="rrlr"/>
    <owl:Class abbreviatedIRI="owl:Thing"/>
  </owl:ObjectSomeValuesFrom>
  <owl:Class IRI="rrl"/>
</owl:SubClassOf>

//RRL ⊆ ∃rrlr=1
<owl:SubClassOf>
  <owl:Class IRI="rrl"/>
  <owl:ObjectIntersectionOf>
    <owl:ObjectSomeValuesFrom>
      <owl:ObjectProperty IRI="rrlr"/>
      <owl:Class IRI="owl:Thing"/>
    </owl:ObjectSomeValuesFrom>
    <owl:ObjectMaxCardinality cardinality="1">
      <owl:ObjectProperty IRI="rrlr"/>
    </owl:ObjectMaxCardinality>
  </owl:ObjectIntersectionOf>
</owl:SubClassOf>

//Mandatory Dependency Range
//∃rrlr- ⊆ R
<owl:SubClassOf>
  <owl:ObjectSomeValuesFrom>
    <owl:ObjectInverseOf>
      <owl:ObjectProperty IRI="rrlr"/>
    </owl:ObjectInverseOf>
    <owl:Class abbreviatedIRI="owl:Thing"/>
  </owl:ObjectSomeValuesFrom>
  <owl:Class IRI="r"/>
</owl:SubClassOf>

//R ⊆ ∃1rrlr- ⊆ 1rrlr-
<owl:Class IRI="r"/>
  <owl:ObjectIntersectionOf>
    <owl:ObjectMaxCardinality cardinality="1">
      <owl:ObjectInverseOf>
        <owl:ObjectProperty IRI="rrlr"/>
      </owl:ObjectInverseOf>
    </owl:ObjectMaxCardinality>
    <owl:ObjectMinCardinality cardinality="1">
      <owl:ObjectInverseOf>
        <owl:ObjectProperty IRI="rrlr"/>
      </owl:ObjectInverseOf>
    </owl:ObjectMinCardinality>
  </owl:ObjectIntersectionOf>
</owl:SubClassOf>
```

Figure 5. crowd-variability OWLink representation of mandatory dependency in Figure 3

For depicting the current *look and feel* of *crowd-variability* editor, Fig. 3 models a mandatory variability dependency between a variation point RRL and a variant R. Both elements were created by pressing the buttons highlighted in red. The GUI includes tool-bars for variation points and variants that presents users' options to change their name(pencil button), add the respective constraint(right arrow button) or variability(up arrow button) dependencies and delete them(cross button). In particular, the dependency in Fig. 3 has been created by pressing the button with an up arrow of the RRL's tool-bar colored in orange, as variants do not have the option of creating variability dependencies.

5 Preliminary Evaluation

crowd-variability has been designed as an extensible and sustainable architecture. It has been developed using expansible graphical libraries and Web technologies. Therefore, it can be extended to other variability modelling proposals such as Feature Models (FM), Common Variability Language (CVL) and SeVaTax. These proposals have a similar expressiveness, however, there are great differences between them. Feature Models model variability through a hierarchical set of features and its relationships [20]. Common Variability Language is a domain-independent language for specifying and resolving variability [21]. SeVaTax is an extension of the orthogonal variability models [9].

From a graphical point of view, JointJS graphics library offers the possibility of expanding its functionalities by creating or/and adding new plug-ins, that specify each primitive of new variability modelling languages. Furthermore, new formalisation methods, like the one for OVM diagrams presented in [7], can be incorporated.

Currently, this tool is integrated with Racer reasoning system. Nonetheless, new back-end reasoners can be connected to the tool.

Hence, a relevant feature of *crowd variability* is to consider users' preferences and usages, allowing the selection of different approaches to model variability and distinct reasoners for the reasoning service.

6 Comparison with other tools

Currently, the increment of the complexity of information systems determine more complex OVM to be modeled. Therefore, manual management of such models becomes an impossible labor without an automated tool support. In this regard, we have surveyed existing tools in certain aspects like model specification and its integration with automatic reasoning systems.

FaMa-OVM[6] is an extensible tool for automated analysis of OVM diagrams, integrated with multiple off-the-shelf reasoners such as SAT4j, JavaBDD and Choco-Solver. However, its input model is specified in a textual format, which describes the model relationships, constraints, attributes and global attributes. These last two features correspond to OVM diagrams with attributes, also supported by FaMa-OVM, although the use of a textual format makes the model specification a difficult and error-prone task.

In [5], OVM models are formalised in VFD+ (based in FFD), a parametric construct designed to define the syntax and semantics of FODA-inspired Feature Diagrams(FD) languages. Nonetheless, in order to automate the analysis, the models in VFD+ are translated into CNF to be later sent to the SAT4j solver that determines the formula satisfiability. Furthermore, this tool does not offer any graphical front-end yet.

As it has been mentioned before, both SAT-approaches present certain limitations related to more restrictive logics and thus failing to reflect the finer logical structure of variability models. On these proposals, the relationships among services are missed when embedding variability models in CNF(Conjunctive Normal Form). As a consequence, the domain validation process involves several product-by-product SAT checks. For this reason, identifying inconsistency sources is hard in such approaches as has also been evaluated in [9].

7 Conclusions and Future Work

In this work, we have presented *crowd-variability*, a Web tool that integrates graphical support for orthogonal variability models design and automatic reasoning so as to validate such models. It pursues to solve those limitation presented in surveyed proposals based on SAT-solvers through the use of description logics. Moreover, we aim to avoid error occurrences in models design by focusing on a graphical user interface with a set of primitives of the OVM language and the necessary functions for diagrams construction and edition. Taking this into consideration, we have reached the first prototype of *crowd variability*, which allows simple OVM designs and its consistency analysis by logic reasoning systems. Finally, we address extensions tool capability in a preliminary evaluation.

In future works, we plan to continue evaluating *crowd-variability*, enhance and extend its functionalities. Currently, it determines when a model is valid, however, we also aim to be able to determine dead services, i.e. services that will never be derived for a product, valid product instantiations and to explain the inconsistencies found in the OVM.

References

1. Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
2. Fabricia Roos-Frantz, José A Galindo, David Benavides, Antonio Ruiz Cortés, and J Garcia-Galán. Automated analysis of diverse variability models with tool support. *Jornadas de Ingeniería del Software y de Bases de Datos (JISBD 2014), Cádiz, Spain*, page 160, 2014.
3. David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, September 2010.
4. Matthias Kowal, Sofia Ananieva, and Thomas Thüm. Explaining anomalies in feature models. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2016*, pages 132–143, New York, NY, USA, 2016. ACM.

5. A. Metzger, K. Pohl, P. Heymans, P. Y. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 243–253, Oct 2007.
6. Fabricia Roos Frantz, José Ángel Galindo Duarte, David Felipe Benavides Cuevas, and Antonio Ruiz Cortés. FaMa-OVM: A tool for the automated analysis of ovms. In *Proceedings of the 16th International Software Product Line Conference - Volume 2. SPLC '12, ACM, New York, NY, USA, 2012*.
7. Germán Braun, Matias Pol'la, Laura Cecchi, Agustina Buccella, Pablo Fillottrani, and Alejandra Cechich. A DL Semantics for Reasoning over OVM-based Variability Models. 2017.
8. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In *Logics for Databases and Information Systems*, pages 229–263. Kluwer, 1998.
9. M. Pol'la, A. Buccella, M. Arias, and A. Cechich. Sevatax: service taxonomy selection validation process for spl development. In *2015 34th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–6, Nov 2015.
10. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
11. Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The racerpro knowledge representation and reasoning system. *Semantic Web Journal*, 2012.
12. Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. OwlLink. *Semantic Web*, 2(1):23–32, 2011.
13. Christian Gimenez, Germán Braun, Laura Cecchi, and Pablo Fillottrani. Una arquitectura cliente-servidor para modelado conceptual asistido por razonamiento automático. In *XVIII Workshop de Investigadores en Ciencias de la Computación*, 2016.
14. Christian Gimenez, Germán Braun, Laura Cecchi, and Pablo Fillottrani. crowd: A tool for conceptual modelling assisted by automated reasoning - preliminary report. In *the 2nd Simposio Argentino de Ontologías y sus Aplicaciones SAOA '16 JAIIO '16*, 2016.
15. Eduardo Grosclaude, Rafael Zurita, Rodolfo del Castillo, Miriam Lechner, and José Riquelme. Designing a myro-compatible robot for education as copyleft hardware. In *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*, 2014.
16. Rafael Zurita, Juan de la Fuente, Martín Bucarey, Daiana Bonet, Rodolfo del Castillo, Guillermo Grosso, Laura Cecchi, Jorge Rodríguez, et al. Mejorando las posibilidades de aprender a programar, ampliación del robot educativo multiplo n6 max a franksteto. In *XII Congreso de Tecnología en Educación y Educación en Tecnología (TE&ET, La Matanza 2017)*, 2017.
17. Multiplo —Open Source Robotics Building System. Último acceso Julio 2018, website <http://www.robotgroup.com.ar/en/kits-de-rob%C3%B3tica/robots/n6-max-detail>.
18. Ahmet Soylu, Felix Mödritscher, Fridolin Wild, Patrick De Causmaecker, and Piet Desmet. Mashups by orchestration and widget-based personal environments: Key challenges, solution strategies, and an application. *Program*, 2012.
19. V. Haarslev and R. Möller. Racer system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.
20. K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA., 1990.
21. O. Haugen, B. Moller-Pedersen, J. Oldevik, G. K. Olsen, and A. Svendsen. Adding standardized variability to domain specific languages. In *2008 12th International Software Product Line Conference*, pages 139–148, 2008.