- ORIGINAL ARTICLE -

# A Reference Architecture for Ontology Engineering Web Environments

## Una Arquitectura de Referencia para Ambientes Web de Ingeniería Ontológica

Germán Braun[1,3], Elsa Estevez[2,3], and Pablo Fillottrani[2,4]

[1]*Universidad Nacional del Comahue,*
{german.braun}@fi.uncoma.edu.ar
[2]*Universidad Nacional del Sur,*
{ece, prf}@cs.uns.edu.ar
[3]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina.*
[4]*Comisión de Investigaciones Científicas (CIC), Provincia de Buenos Aires, Argentina.*

## Abstract

Ontology authoring, maintenance and use are never easy tasks, mostly due to the complexity of real domains and how they dynamically change as well as different background possessed by modellers about methodologies and formal languages. However, although the needs for ontologies are well-understood, not less important is to provide editing tools to manipulate and understand them. In this context, this work proposes and documents a reference architecture for such tools running in web environments. Moreover, it provides the rationale for boosting the collaborative development of a novel tool based on this architecture, named *crowd*. Previous surveys reveal that few Web-based ontology engineering environments have been developed and in addition, almost all of them are mere visualisers, with limited graphical features and lacking inference services.

**Keywords:** Ontology Engineering, Software Architectures.

## Resumen

La definición, mantenimiento y use de ontologías son tareas difíciles debido, en mayor medida, a la complejidad inherente al mundo real y a cómo éste cambia dinámicamente. Asismismo, también se debe a las diferencias en conocimiento sobre metodologías y lenguajes formales por parte de los modeladores. Sin embargo, aunque la necesidad de crear y obtener ontologías es clave, es también importante contar con herramientas para manipularlas y entenderlas. Este trabajo propone y documenta una arquitectura de referencia para ambientes Web y ofrece los fundamentos para impulsar el desarrollo colaborativo de la herramienta *crowd*, la cual está basada sobre dicha architectura. Revisiones previas de la literatura indican la existencia de un número reducido ambientes para la Ingeniería Ontológica basados en tecnologías Web, sin embargo, casi en su totalidad son sólo visualizadores de modelos con soporte gráfico limitado y ausencia de razonamiento lógico integrado.

**Palabras claves:** Ingeniería Ontológica, Arquitecturas de Software.

## 1 Introduction

Ontology authoring, maintenance and use are never easy tasks, mostly due to the complexity of real domains and how they dynamically change as well as different background possessed by modellers about methodologies and formal languages [1]. However, although the needs for ontologies are well-understood, not less important is to provide editing tools to manipulate and understand them [2].

Ontology Engineering is a set of activities, methodologies, languages and tools related to the authoring, maintenance, refinement, evaluation and use of ontologies. Authoring is key for the ontology engineering and it requires to be backed by proper methodologies for guiding users through the building process. In particular, its complexity increases when such models should be created from scratch or when they should be merged to other ones in order to compose them. All of these activities appear fragmented across several tools adding a more important ingredient to the complexity of the ontologies life cycle. The lack of an adequate and seamless environment affects not only the uptake of ontologies by new users, but also the quality of artifacts created by modellers. Previous surveys [3, 4] reveal that a lot of ontology engineering environments have been developed, but too few of them are web-based systems and in almost all the cases they are mere visualisers, with limited graphical features and lacking of inference services. In fact, some of them claim no well-accepted framework for common authoring tasks exist because of a poor understanding of the effectiveness of tools [2].

In this direction, new architectures for ontology engineering environments must be prepared for envisioned future changes while being easily extendable and providing basic functionality. Thus, these architectures are for a family of systems and particular members of this family can be constructed based on a reference architecture containing explicit places where extensions can take place. This work proposes and documents a reference architecture for ontology engineering web environments. Moreover, it provides the rationale for boosting the collaborative development of a novel tool based on this architecture, named *crowd*. Documenting software architecture is essential to work cooperatively in solving larger problems than single groups could solve individually. At the same time, it helps to define and evaluate quality attributes, identify parts where systems could be extended and communicate baselines and design decisions made to let others use it properly [5, 6].

The rest of this article is structured as follows. Section 2 explains the motivation of this research work and previous results. Section 3 presents a survey on related architectures. Section 4 describes our proposed reference web architecture and section 5 gives details about the first version of *crowd*. Finally, in section 6, we summarise conclusions and discuss limitations and future works.

## 2 Motivation and Background

Misunderstandings about some property-based distinctions between ontologies and models have been undertaken by a lot of different proposals [7]. These distinctions were mainly based on the needs of formalisation of conceptual models (p.e. UML, ER, ORM) for expressing ontologies and conversely the logic-based reasoning capabilities provided by ontology languages (p.e. OWL [8]) not captured by models. In particular, despite obvious differences between the expressiveness of conceptual modelling languages (CMLs) and ontology languages, many tools have partially validated this claim [9, 10, 11, 12]. Thus, the effectiveness of using graphical syntax based on these conceptual languages for expressing ontologies, even with complex languages, is still been considered for ontology engineering environments [3, 2]. Evidence for these claims indicate that the common core of UML, (E)ER and ORM 2 can be formalised in the $\mathcal{ALNI}$ Description Logics (DL) [13] guaranteeing tractable reasoning over them [14]. Even more, each language can express full $\mathcal{ALCQI}$ while keeping reasoning capabilities.

At the same time, OWL notations have been also proposed, which are mostly based on graphs [15, 16]. Aiming at the needs for an unified visual notations for ontologies, these approaches claim that switching between tool can be confusing given the diversity of visual representations. In addition, several tools implement just a few ontology engineering activities

providing very different user interfaces, graphical and formal languages and reasoning capabilities. These last quotes give a real insight into going towards an unified tool integrating not only the visual notations and their common features, but also supporting other activities of the ontology engineering altogether in a single graphical and logical framework with a set of core functionalities.

In this context of lacking of a holistic view, a web tool called *crowd* [17, 18, 19] has been developed. *crowd* is a multi-view web environment for ontology development being supported by both Universidad Nacional del Comahue[1] and Universidad Nacional del Sur[2] in Argentina. The first intention behinds the tool is to assist users to author and edit ontologies and conceptual models adopting standard CMLs and employing complete logical reasoning to verify the satisfiability of specifications, infer implicit constraints and suggest new ones. The leverage of automated reasoning is enabled by a precise semantic definition of all the elements of the diagrams. Hence, diagrams constraints are internally translated into a logic-based formalism capturing typical features of models. To this end, the tool is fully integrated with a powerful logic-based reasoning server acting as a background inference engine. Moreover, since *crowd* is based on a deduction-complete notion of reasoning support relative to the diagram graphical syntax, users will see the original model graphically completed with all the deductions and expressed in the graphical language itself. This includes checking class and relationship consistency, discovering implied class and cardinality constraints. Empowered by web technologies, *crowd* has been designed as a scalable and maintainable architecture for adapting new reasoning, query and documenting engines, graphical languages and design methodologies. It has been conceived from scratch as a graphical-centric tool for ontology modelling, adopting standard languages and considering the possibility to expand its graphical primitives for more expressiveness. The first version is available at `http://crowd.fi.uncoma.edu.ar` together with its source code.

The road map followed to develop the first version of *crowd* considered the features and related work explained below:

**(F1)** Strong Theoretical Background of Graphical Tools. [19, 20] *crowd* is based on a well-founded structure, a heterogeneous algebra [21], describing a family of sets such as a metamodel $\mathcal{M}$ for a non-empty set of CMLs, a non-empty set $\mathcal{L}$ of the languages supported by $\mathcal{M}$, a non-empty set of formal languages $\mathcal{FL}$ for ontologies and a non-empty set $E$ of finitary operations for mapping primitives from $\mathcal{L}$ to an ontology language from $\mathcal{FL}$ (whenever possible).

---

[1] `http://faiweb.uncoma.edu.ar/`
[2] `https://cs.uns.edu.ar/home/`

**(F2)** Logic-Based Reasoning on Graphical Ontologies. [22, 9, 20] A full-flagged DL encoding of the CMLs enables *crowd* to offer standard logic reasoning on graphical ontologies as a way to help users along some of the ontology engineering activities. This tool allows to graphically support powerful OWL 2 features such as non-qualified and qualified cardinality restrictions for any *n* natural number.

**(F3)** Ontology-Based Metamodelling. Graphical views in *crowd* require a full specification of the similarities and mismatches of the CMLs. In this context, *crowd* is backed by a needed metamodel, named KF [23], for operations of conversion, transformation and approximation among models. Each ontology in *crowd* is an instance of the KF-metamodel.

**(F4)** Ontology Multi-View. [18] Based on the baselines of the KF-metamodel, *crowd* implements a multi-view GUI for switching between them and thus offering different ways to interact with the current model.

**(F5)** Export OWL 2. *crowd* enables exporting full descriptions of graphical ontologies in OWL syntax. It also provides OWLlink [24] specifications in order to get the inputs offered to off-the-shelf reasoners.

**(F6)** Import OWL 2. Importing in *crowd* is a two-step process looking for extracting as many graphical axioms as possible from ontologies in OWL documents. This novel process is supported by logic-based reasoning together with a SPARQL-DL [25] engine.

**(F7)** Ontology Documentation. Human-readable documentation of ontologies facilitates their understanding, reuse and adoption by third-parts. In this sense, *crowd* helps users documenting their ontologies by invoking Widoco [26] and producing a full, enriched version of their models.

**(F8)** Namespaces Definitions. *crowd* handles namespace definitions through a novel treatment of URIs and thus enabling the reuse of other vocabularies in the same graphical model. It provides a modal widget for prefixes and their values. Moreover, each graphical primitive presents a dedicated widget to enter its specific definition.

## 3 Related Architectures

Through literature and web search we have identified the following ontology editing tools that are, at least to some degree, actively maintained: WebProtégé [28], Protégé [29] -OWLViz[3], OntoGraf[4], SOVA[5],

[3]`http://protegewiki.stanford.edu/wiki/OWLViz` accessed July 2018

[4]`http://protegewiki.stanford.edu/wiki/OntoGraf` accessed July 2018

[5]`http://protegewiki.stanford.edu/wiki/SOVA` accessed July 2018

NORMA [10, 30], ICOM [9], TopBraid Composer [31], Graphol [16], OWLGrEd [11], Menthor [32], NeOn Toolkit [33], VOWL [34], GrOWL [35], OntoTrack [36], SWOOP [37], Hozo [38] and Graffo [39]. Five of them (VOWL, Graphol, Hozo, GrOWL and Graffo) are mere ontology visualisers, while the remaining ones present (at least in the literature) some degree of interactively and integration of logical support with graphical models. Moreover, OntoTrack, GrOWL and SWOOP seem to be not publicly available to download or deprecated in order to run a demo of the software.

However, none of them presents both interactive and web support in the very same ontology engineering environment as proposed by our novel architecture. In the remaining, we conduct a comprehensive analysis over these tools (leaving OntoTrack, GrOWL and SWOOP out of the scope of this work) by addressing the following aspects: (1) operating mode - meaning standalone or web (or both); (2) graphical language type - possible categories include ad-hoc node-link notations, UML/EER/ORM 1/ORM 2 or own languages; (3) graphical expressiveness - OWL 1, OWL 2, RDF or subsets of them, (4) non-graphical expressiveness - underlying ontology languages supported OWL 1, OWL 2, RDF or subsets of them; (5) integration of visual models and reasoning aiming at identify tool that graphically complete original models with all the deductions and expressed in the graphical language itself (whenever possible) (✓/✗); and (6) multi-view support, where diagrams could be shown in more than one graphical language (✓/✗). In particular, the aim of assessing such an integration is to pragmatically justify the needs for the knowledge visualisation process defined in [19]. Specifically, the graphical and reasoning integration refers to a "back and forth" transformation between a graphical model and a logic formalisation capturing the semantics of this model.

As shown by the comparison, only OWLGrEd, VOWL and WebProtégé offer some kind of web support. Nevertheless, the first two are mere visualisers of UML-like and VOWL language, respectively, and no editing possibilities. Conversely, WebProtégé does provide editing support, laying a strong emphasis on collaborative development although visual notations are absent. Other weak aspect of the surveyed tools is the lack of multi-view. Multiple views of a model allow to interoperate them as well as share and communicate them among different stakeholders [23]. Multi-view in the context of a tool fully integrated with logic reasoning also require multiple ways to encode graphical models. This is also provided by the proposed environment and is absent in the related tools. Integration of visual models and reasoning is slightly provided by ICOM and NORMA (subsumptions, $m..n, n = 1$ cardinalities, disjunctions and equivalences) while in remaining surveyed tools no integration exist or only for subsumptions in few cases. With reference to onto-

Table 1: Overview of ontology editors considering the criteria proposed.

| | Web Standalone | Graphical Language | Graphical Expressiveness | Non-Graphical Expressiveness | Graphical and Logical Integration | Multi View |
|---|---|---|---|---|---|---|
| WebProtégé | web | ✗ | ✗ | OWL 1/OWL 2 | ✗ | ✗ |
| Protégé OWLViz | standalone | ad-hoc Node-link | subsumption | OWL 1/OWL 2 | ✓ *only subsumption | ✗ |
| Protégé OntoGraf | standalone | ad-hoc Node-link | subsumption | OWL 1/OWL 2 | ✗ | ✗ |
| Protégé SOVA | standalone | ad-hoc Node-link | OWL 1 | OWL 2 | ✓ *only subsumption | ✗ |
| NORMA | standalone | ORM 2 | $\mathcal{ALCQI}$ | $\mathcal{ALCQI}$ | ✓ | ✗ |
| ICOM | standalone | EER | $\mathcal{ALCI}$ | $\mathcal{ALCQHI}$ | ✓ | ✗ |
| TopBraid Composer | standalone | ad-hoc Node-link/ UML | OWL 2 subset | OWL 2 | ✓ *only subsumption | ✓ |
| OWLGrEd | both | Extended-UML | OWL 2 subset | OWL 2 | ✗ | ✗ |
| eddy - Graphol | standalone | Graphol *own language | $\mathcal{SROIQ(D)}$ | $\mathcal{SROIQ(D)}$ | ✗ | ✗ |
| Menthor | standalone | OntoUML *own language [27] | OntoUML | UFO [27] | ✗ | ✗ |
| NeOn Toolkit | standalone | ad-hoc Node-link | OWL 2 | OWL 2 | ✓ *only subsumption | ✗ |
| VOWL | both | VOWL *own language | OWL 2 | OWL 2 | ✗ | ✗ |
| Hozo | standalone | ad-hoc Node-link | role-concepts & wholeness/relation concepts | role-concepts & wholeness/relation concepts | ✗ | ✗ |
| Graffo | standalone | ad-hoc Node-link | OWL 2 | OWL 2 | ✗ | ✗ |

logy engineering activities, only the classic standalone Protégé is strong in the use of ontologies because is highly extensible thanks to its very sophisticated plug-in architecture. Table 1 shows the comparison.

## 4 *crowd* Architecture Views

In this work, we document and detail the architecture of *crowd* offering three views: modules, component and connectors and allocation. Each of them highlights different aspects of implementation and the underlying decisions for future and collaborative support. In each view, we explain its motivation and its intended meaning, and describe its principal parts based on the chosen styles. For all the cases, we use UML notation. Before introducing the architecture, we briefly explain some assumptions made. First, we consider modules as implementation units of software together with a set of responsibilities. Likewise, components represent elements such as objects, clients and servers. Finally, artifacts represent files and software packages to be installed and deployed in a running environment.

### 4.1 Module View

**Motivation.** The system is decomposed in implementation units or modules, which can be related by *is-part-of*, *depend-on* and *is-a* relations. Each module provides a responsibility set aiming at defining the role of such module in achieving certain functionality. This view details how these units ensemble to form one longer structure. Its intended use is to provide a blueprint for the source code.

**Architecture Styles.** Decomposition, Uses and Layered.

**Description.** The module view of *crowd* is a coherent combination of architecture styles. First of all, *crowd* is divided into layers, each one providing a set of services to the other layer.

As depicted in Fig. 1, the upper layer groups a set of modules related to the user interface of the tool. At the same time, it is decomposed in implementation units describing a containment relation and the dependencies among them. The main module is the `gui`, which depends on the packages `views` and `model`, all of them for UML, EER and ORM support. The first one provides the definition of templates and their respective event handlers. The latter one manages the underlying graphical models, their primitives and the interaction with the graphical library by means of respective UML, ORM and EER factories. Two specific modules support common features: `model.uris` and `views.common` implements orthogonal functionalities for the treatment of namespaces and common templates and events of the GUI, respectively.

The bottom layer contains the modules required for the processing of ontologies: translators. builders, users management and interfaces with off-the-shelf tools. The main module of this layer is `translator`, which has a dual role orchestrating both the encoding of graphical model into logic and processing ontologies extracted from OWL documents using specific `strategies` and `builders`. `strategies` depend on CMLs and how they are encoded in logic, while `builders` give the structure of `documents` for different syntaxes. Lastly, `translator` contains `qapackages`, which provides queries and parses their
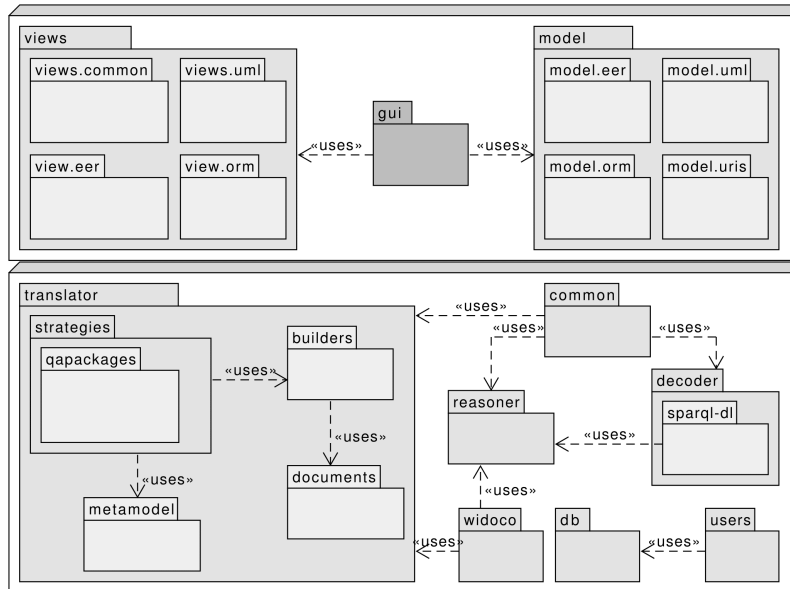
Figure 1: UML module view of *crowd*

answers when ontologies are processed by reasoners. To this end, `reasoner` connects *crowd* to reasoning systems and allows to create as many interfaces for other tools as desired. In this sense, `reasoner` implements connectors for the documentation tool `widoco` and a query engine `sparql-dl`. The `decoder` package is in charge of extracting ontologies from OWL documents by means of SPARQL-DL. It invokes the respective engine set with pattern queries that return structural aspects of the current ontology. This module implements key functionalities for managing URIs and also depend on a strategy and a builder. Altogether, `translator`, `decoder` and `reasoner` are used by the `common` module, which is responsible for properly combining them to built even more complex functionalities. Fig. 2 illustrates the interaction among objects in the OWL 2 importing scenario.

Finally, *crowd* includes simple modules for user and model management.

## 4.2 Component-and-Connector View

**Motivation.** This kind of views complements the previous one by showing elements (components) that have some run-time presence such as objects, and how these elements communicate each other by means of connectors (communication links, protocols). Connectors define pathways of interaction between components, which have a set of ports to connect them. The resultant view is a graph showing how the system works.
**Architecture Styles.** Call-Return.
**Description.** *crowd* has been conceived from scratch as a client-server implementation. In this sense, `Client` provides a user friendly interface to graphical conceptual modelling running on a web browser
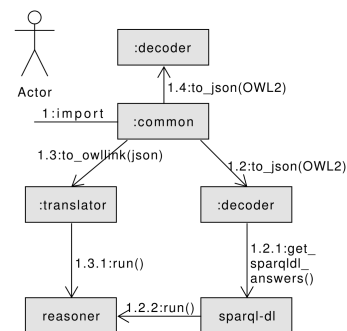


Figure 2: Communication diagram for importing OWL 2 ontologies. Arrows indicate name and direction of message transmission between objects. Numbers represent the sequence of messages passed between objects.

and requires reasoning services of the `Server` running on a web server. This tool follows the very same principles of the web, where clients access information from one or more servers distributed using the HTTP protocol, which is a form of request/reply invocation.

The Fig. 3 depicts a Component-and-Connector View (C&C) for the client side, its component and the communication scheme between them and can be described as followed. *crowd* supports widgets and events for multiples diagram objects for standard languages. They are accessible through the `GUI` component that handles different instances of `GUIIMPL`. Each one of such instances is the representation of a specific interface, its events, widgets and diagram. An `Adapter` component implements the Adapter structural pattern [40] for providing a bridge between an
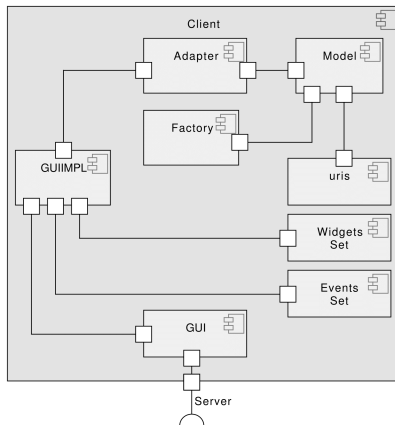
Figure 3: UML C&C view of the *crowd* client.

Table 2: Mapping between C&C and Module Views for both client and server. (*) represent modules as part of more than one run-time component.

| C&C View | Module View |
|---|---|
| Model, Factory, Uris | model |
| GUI, GUIIMPL, Adapter | gui |
| Widgets Set, Events Set | views |
| *:Translator | common, translator |
| *:Decoder | common, decoder |
| OntoExtractor | decoder |
| Strategy | strategies |
| *:Builder | builder |
| QAPackage | qapackages |
| Off-the-shelf Docs Generator | widoco |
| Off-the-shelf Reasoner, SPARQL-DL engine | reasoner |
| User Management | users |

interface and a diagram (or `Model`), for instance an UML diagram and its GUI. The `Widget Set` manages all the widgets needed for a specific diagram and also provides common ones to all the languages. Lastly, a `Factory` component implements an Abstract Factory pattern [40]. This factory defines an interface for creating the graphical elements of the standard languages of *crowd*. It delegates this task to the underlying library JointJS [6].

Moreover, the supported CMLs share some common components such as `uris`, for the full treatment of namespaces. An URI is an unique identifier for each primitive of the ontologies. *crowd* keeps the track of each primitive by specifying its (graphical) name, an abbreviation (prefix) and an URL. An comprehensive treatment of URIs is essential for the tool because ontologies must be available through internet [1].

The Fig. 4 schematises the server, its sub-components and connectors of the environment. For simplicity of the diagram, only `Reasoning` elements are fully expanded while details of `Import` ones are hidden because they present similar C&C views. The highlighted part of this scheme (in green) indicates the importance of the logic-based reasoning systems

in backing the services of importing OWL ontologies and reasoning over graphical models. In all the cases, off-the-shelf tools provide communicating interfaces or protocols such as OWL API [41] and OWLlink. Both `Traslator` and `Decoder` components connects to external engines. In particular, the `OntoExtractor` component represents run-time objects implementing the responsibilities defined for the `decoder` and `sparql-dl` modules. It is in charge of processing the outputs of SPARQL-DL queries sent to the respective engine, which aims at extracting both intentional (TBox) and assertional (ABox) axioms from OWL ontologies. Such queries are defined in the specification of SPARQL-DL language[7] (see Fig. 5) and are processed by an engine[8] built on top of existing reasoners and that can be fed with ontology documents in any of its linearisations (OWL/RDF, OWL/XML, between others).

Similarly, the components `Strategy`, `Builder` and `QAPackage` communicate each other and implement responsibilities associated to the module `translator`. `Translator` also connects to external reasoning systems though the OWLlink protocol so that they are fed with OWLlink files containing the OWL encoding of an graphical ontology together with a set of OWLlink queries for sanity checking of the ontology. The output of this reasoning task is post-processed by the `QAPackage`. Finally, the new OWL ontology is sent to a `Decoder` instance to be after compared against the original graphical one. In this sense, the decision to integrate two different ways to query ontologies (OWLlink and SPARQL-DL) is in fact pragmatic and is well-grounded in the well-known requirements of interoperability: the use of standard protocols as OWLlink to interconnect to other reasoning systems and the support for the importing of OWL ontologies, which requires of more expressive query languages to extract as much knowledge as possible from OWL documents. Fig 6 presents a sequence diagram describing a complete functionality of system and complements the C&C view highlighting the right interactions in run-time.

To summarise, Table 2 illustrates the mapping between C&C and Module views showing which modules in the Module view contribute to the implementation of components shown in the C&C view.

### 4.3 Allocation View

**Motivation.** Allocation views present a mapping between software and non-software elements. The deployment view depicts the physical configuration of the software architecture, in particular, how the components of the architecture are allocated to physical nodes. The install style maps the components of the architecture to the structures in the file system of the

---

[6] https://www.jointjs.com/opensource

[7] http://derivo.de/en/resources/sparql-dl-api/sparql-dl-syntax/
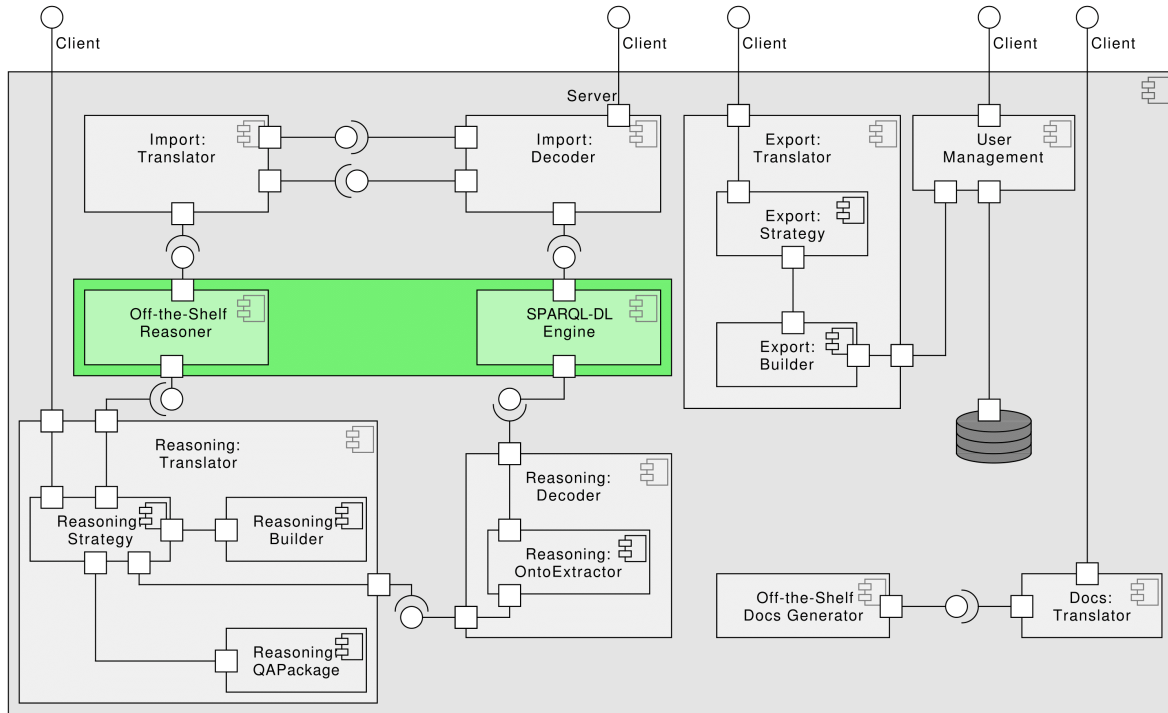
[8] http://derivo.de/en/resources/sparql-dl-api/

Figure 4: UML C&C view of *crowd* server. Circles on top indicate services to be provided to clients.
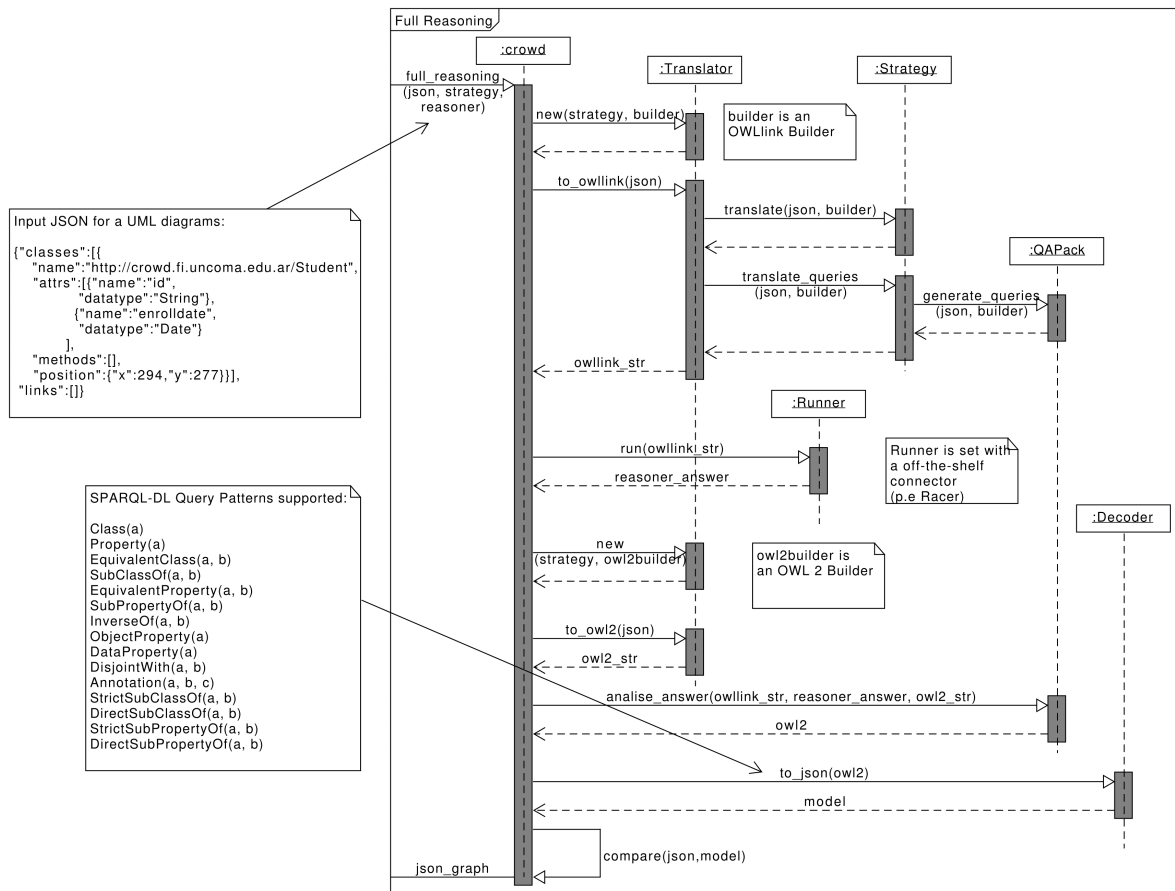


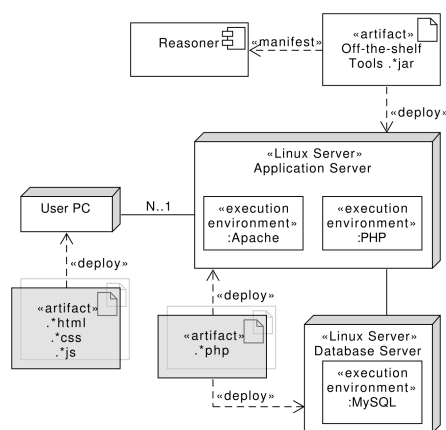Figure 5: UML sequence diagram for full reasoning service, input json example and SPARQL-DL queries.

Figure 6: UML deployment and install scheme for the centralised architecture of *crowd*.

running environment; while the work assignment style shows the teams responsible for the development of each of the software modules.

**Architecture Styles.** Deployment and Install.

**Description.** *crowd* is a centralised architecture so that all the PHP artifacts, i.e. ".*php" files, are deployed in a single HTTP server running Apache[9] and MySQL[10] services. Off-the-shelf artifacts, ".*jar" files for reasoning systems and SPARQL-DL engines, are also installed in the same server and correspond to the `Reasoner` component. Currently, server is x86 Intel 3GHz, running Apache v2.4.10, MySQL v5.6.4 on a Debian GNU/Linux 8 jessie. In addition, PHP v7.0 and Java RE v1.8.0 are also deployed in such server. Finally, multiplicity N..1 indicates the number of instances of a client at each end of the communication path. Such clients represent web browsers, which interpret the HTML, CSS and JS artifacts, ".*html", ".*css" and ".*js" files of the user interface.

## 5 Evaluation: *crowd* v0.9

First version of *crowd* has been deployed at `http://crowd.fi.uncoma.edu.ar`. Its implementation is the first one so that there are not yet performance assessments of the overall system. The initial steps of the development have been reported in [17, 18].

Currently, the *crowd* editor presents a workspace for editing visual models in UML: classes, attributes, generalisation and binary associations with $n..m$ cardinalities, for $n, m \geq 1$. Visual models can be encoded in DL (and serialised in OWL 2) to be sent them to reasoning systems for checking satisfiability and properties refinement. Released version of *crowd* manages URIs and enables definition of namespaces for re-using of other ontologies available across the web. Users can log-in into *crowd* to host their diagrams

although modelling tasks can be done but no state is saved in our server. The interface is zoomable, that is, the level of detail and size of the icons that represent the model can be smoothly changed by pressing dedicated options, which define a scale from 25% to 125%. This allows the user to visualise big ontologies into the same windows.

Complementary evaluations and applications of our reference architecture beyond *crowd* have been recently published in [42, 43]. The first one implements a tool for validating visual Orthogonal Variability Models (OVM) in the context of Software Product Lines (SPL). The second one implements a Visual Query Language (VQL) based in UML for the SPARQL-DL language.

## 6 Conclusions and Future Works

In this article, we have proposed and documented a reference architecture for a web ontology engineering environment. We described in details three complementary views merging different styles for each one: modules (decomposition, layered, uses), component-and-connectors (call-return) and allocation (deployment and install). The architecture described in this paper addresses specific needs. In particular, decreasing the cognitive effort of users when understanding, integrating and using of ontologies. Moreover, it aims at providing the rationale for boosting the collaborative development of a novel tool based on this architecture, named *crowd*.

Some limitations have been identified. CMLs languages are not enough expressive considering the underlying expressiveness of reasoning systems. Users could not visually represent disjoint and equivalence constraints, among others logical axioms. Due to this, we have considered two possible solutions: allowing users to define logical expressions, which requires knowledge about formal systems, or providing new visual primitives with a precise semantic definition. In this sense, *crowd* implements a text area for introducing OWL 2 statements as a temporary solution.

Future work involves releasing a new version of *crowd* implementing the remaining primitives of UML and (E)ER and full reasoning over these diagrams. Moreover, we expect to extend *crowd* to other engineering activities such as alignment and merging of ontologies as well as managing of instances and increasing compliance with W3C recommendations.

## Competing interests

The authors have declared that no competing interests exist.

---

[9] `https://httpd.apache.org/`
[10] `https://www.mysql.com/`

## References

[1] R. Mizoguchi, *Ontology Engineering Environments*, pp. 275–295. Springer, 2004.

[2] M. Vigo, S. Bail, C. Jay, and R. Stevens, "Overcoming the Pitfalls of Ontology Authoring: Strategies and Implications for Tool Design," *International Journal of Human Computer Studies*, 2014.

[3] N. Achich, B. Bouazis, A. Algergawy, and F. Gargouri, "Ontology visualization: An overview," in *Intelligent Systems Design and Applications - 17th ISDA*.

[4] N. M. Meenachi and M. S. Baba, "Article: Web Ontology Language Editors for Semantic Web- A Survey," *International Journal of Computer Applications*, 2012.

[5] D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson, *Documenting Software Architectures: Views and Beyond*. 2nd ed., 2010.

[6] H. Gomaa, *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press, 1st ed., 2011.

[7] C. Atkinson, M. Gutheil, and K. Kiko, "On the Relationship of Ontologies and Models," in *Proceedings of the 2nd WoMM*, 2006.

[8] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, eds., *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[9] P. Fillottrani, E. Franconi, and S. Tessaris, "The ICOM 3.0 intelligent conceptual modelling tool and methodology," *Semantic Web*, 2012.

[10] M. Curland and T. A. Halpin, "The NORMA Software Tool for ORM 2," in *CAiSE Forum*, Lecture Notes in Business Information Processing, Springer, 2010.

[11] K. Cerans, J. Ovcinnikova, R. Liepins, and A. Sprogis, "Advanced OWL 2.0 Ontology Visualization in OWLGrEd," in *DB&IS*, 2012.

[12] R. Hodrob and M. Jarrar, "On Using a Graphical Notation in Ontology Engineering," 2012.

[13] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[14] P. R. Fillottrani and C. M. Keet, "Evidence-based languages for conceptual data modelling profiles," in *Proceeding of the 19th ADBIS*, 2015.

[15] S. Negru, F. Haag, and S. Lohmann, "Towards a unified visual notation for owl ontologies: Insights from a comparative user study," in *Proceedings of the 9th ICSS*, 2013.

[16] M. Console, D. Lembo, V. Santarelli, and D. F. Savo, "Graphol: Ontology representation through diagrams," in *Informal Proceedings of the 27th International DL*, 2014.

[17] C. Gimenez, G. Braun, L. Cecchi, and L. Fillottrani, "crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning - Preliminary Report," in *Proc. of the 2nd SAOA@JAIIO*, 2016.

[18] G. Braun, C. Gimenez, P. Fillottrani, and L. Cecchi, "Towards conceptual modelling interoperability in a web tool for ontology engineering," in *3rd SAOA@JAIIO*, 2017.

[19] G. Braun, C. Gimenez, L. Cecchi, and P. Fillottrani, "Towards a visualisation process for ontology-based conceptual modelling," in *Proc. of the IX ONTOBRAS*, 2016.

[20] G. Braun, L. Cecchi, and P. Fillottrani, "Integrating graphical support with reasoning in a methodology for ontology evolution," in *Proc. of the 1st JOWO@IJCAI*, 2015.

[21] G. Birkhoff and J. D. Lipson, "Heterogeneous algebras," *Journal of Combinatorial Theory*, 1970.

[22] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artif. Intell.*, 2005.

[23] C. M. Keet and P. R. Fillottrani, "An ontology-driven unifying metamodel of UML class diagrams, eer, and ORM2," *Data Knowl. Eng.*, 2015.

[24] T. Liebig, M. Luther, O. Noppens, and M. Wessel, "Owllink," *Semantic Web*, 2011.

[25] E. Sirin and B. Parsia, "Sparql-dl: Sparql query for owl-dl," in *In 3rd OWL Experiences and Directions Workshop (OWLED-2007*, 2007.

[26] D. Garijo, "WIDOCO: A Wizard for Documenting Ontologies," in *ISWC 2017*, 2017.

[27] G. Guizzardi, *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, 2005.

[28] T. Tudorache, C. Nyulas, N. F. Noy, and M. A. Musen, "Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web," *Semantic Web*, 2013.

[29] H. Knublauch, R. Fergerson, N. Noy, and M. Musen, "The Protégé OWL plugin: An open development environment for semantic web applications," 2004.

[30] F. Sportelli, "NORMA: A software for intelligent conceptual modeling," in *Proceedings of FOIS*, 2016.

[31] TopQuadrant, "TopQuadrant — Products — TopBraid Composer," 2011.

[32] J. L. R. Moreira, T. P. Sales, J. Guerson, B. F. B. Braga, F. Brasileiro, and V. Sobral, "Menthor editor: An ontology-driven conceptual modeling platform," in *JOWO@FOIS*, 2016.

[33] P. Hasse, H. Lewen, R. Studer, and M. Erdmann, "The NeOn Ontology Engineering Toolkit," 2008.

[34] S. Lohmann, S. Negru, and D. Bold, "The protégévowl plugin: Ontology visualization for everyone," in *The Semantic Web: ESWC 2014*, 2014.

[35] S. Krivov, R. Williams, and F. Villa, "GrOWL: A tool for visualization and editing of OWL ontologies," *J. Web Sem.*, 2007.

[36] T. Liebig, "Ontotrack: Fast browsing and easy editing of large ontologies," in *In Proceedings of 2nd EON@ISWC*, 2003.

[37] A. Kalyanput, B. Parsia, E. Sirin, B. Grau, and J. Hendler, "Swoop: A 'web' ontology editing browser," *Journal of Web Semantics*, 2005.

[38] K. Kozaki, Y. Kitamura, M. Ikeda, and R. Mizoguchi, "Hozo: An Environment for Building/Using Ontologies Based on a Fundamental Consideration of "Role" and "Relationship"," in *13th EKAW Conference*, 2002.

[39] R. Falco, A. Gangemi, S. Peroni, D. M. Shotton, and F. Vitali, "Modelling OWL ontologies with graffoo," in *The Semantic Web: ESWC 2014*, 2014.

[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[41] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semant. web*, 2011.

[42] n. Oyarzún, G. Braun, L. Cecchi, and P. Fillottrani, "A Graphical Web Tool with DL-based Reasoning," in *Proc. of the XXVI CACIC*, 2018.

[43] C. Gimenez, G. Braun, L. Cecchi, and P. Fillottrani, "Towards a Visual SPARQL-DL Query Builder," in *Proc. of the XXVI CACIC*, 2018.