An Extensible Meta-modelling Assistant

Ángel Mora Segura, Ana Pescador, Juan de Lara
Modelling and Software Engineering Research Group
(http://miso.es)
Universidad Autónoma de Madrid (Spain)
{Angel.MoraS, Ana.Pescador, Juan.deLara}@uam.es

Manuel Wimmer
Business Informatics Group
(https://www.big.tuwien.ac.at)
TU Wien (Austria)
wimmer@big.tuwien.ac.at

***Abstract*—Meta-models play a pivotal role in Model-Driven Engineering (MDE). They are used to create domain-specific models, and to type model management operations like model transformations or code generators. However, even though creating meta-models is a common activity, it is currently mostly a manual activity, which does not profit from existing knowledge.**

In order to facilitate the meta-modelling task, we propose an extensible meta-modelling assistant. While primarily focussed on helping in the creation of meta-models, it can also help in creating models. The assistant permits the provision of heterogeneous data description sources (like ontologies, RDF data, XML schemas, database schemas and meta-models), and enables their uniform querying. Different kinds of queries are supported, and improved through synonym search. Query results are prioritized through sense disambiguation, can be graphically visualized, and incorporated into the (meta-)model being built.

The assistant has been realized within Eclipse, and its architecture has been designed to be independent of the meta-modelling technology used. As a proof-of-concept, we show its integration within DSL-tao, a pattern-based meta-modelling tool built by our group, and two other tools developed by third-parties. The usefulness of the system is illustrated with a running example in the process modelling domain.

***Index Terms*—Modelling, Meta-modelling, (Meta-)modelling environments, Meta-modelling assistant, Domain-specific languages, Language engineering.**

I. Introduction

Model-Driven Engineering (MDE) promotes the active use of models to automate the different phases and activities of the systems development life-cycle. For example, models can be used to specify, analyse, test, simulate, execute, generate code and maintain the system to be built, among other activities [2], [32]. Many times, those models are not built using general-purpose modelling languages, like UML, but using Domain-Specific Languages (DSLs) [17]. These contain primitives and concepts specifically tailored to a particular domain, which may lead to simpler, more intensional models. The abstract syntax of DSLs is described through a meta-model (which is itself a model), and hence the construction of meta-models is a common activity in MDE.

Models with precisely defined syntax (i.e., which can be processed automatically) play a central role in modern approaches to Enterprise Modelling, especially in the context of Model-Driven Organizations (MDOs) [4]. In this approach, models are integrated and automate the systems that drive the organization, and they are the primary means by which stakeholders interact with the organization.

High quality meta-models, which appropriately capture the most important concepts of a domain, are pivotal for the success of MDE projects, as well as for the MDO vision. However, while meta-modelling is a frequent task in MDE, it remains a mostly manual duty. Hence, unlike modern programming IDEs, which offer different kinds of code recommenders [11], [29] (e.g., for using a given API or for program quick fixing) and reusable libraries, the meta-model developer normally has the burden to create the model from scratch. However, when building a meta-model, developers would greatly benefit from flexible access and reuse of existing knowledge in a domain. This knowledge might be stored in a variety of sources, such as already existing models and meta-models, or other kinds of data description artifacts, such as XML schemas, ontologies or RDF data.

Hence, in this paper, we propose an extensible meta-modelling assistant called Extremo, able to extract information from different sources (meta-models, ontologies, RDF schemas). The system enables the organization of (heterogeneous) sources in repositories, which then can be queried and visualized in a uniform way. The results of the queries are prioritized and aggregated for all sources in the repositories. Such results can then be incorporated into the model under construction. In order to offer more powerful query mechanisms, we make use of Wordnet [21], a database of the English language, to check for synonyms and word senses. While Extremo is particularly suited to help in the construction of meta-models, and was originally designed for this task, it can be used to help in the creation of models as well.

Extremo has been implemented as an Eclipse plugin and is freely available at http://miso.es/tools/extremo.html. The web site includes short videos, which illustrate the main concepts explained in this paper. Extremo's architecture is extensible and modular by profiting from Eclipse extension points, and permits adding support for new information sources and types of queries. The assistant system has been designed to be easily integrated with external modelling environments, also through extension points. As a proof-of-concept, we show its integration with DSL-tao [27], a pattern-based meta-modelling tool developed by our group. Moreover, to demonstrate the feasibility of extending tools developed by third-parties, we show its integration with the default meta-modelling editor of the Eclipse Modelling Framework (EMF) [33], and a UML modelling tool [13]. Along the paper,

we illustrate the usefulness of the approach with a running example in the area of process modelling for e-Government.

The rest of this paper is organized as follows. Section II provides an overview of the approach, its motivation, and introduces a running example. Section III explains the main ingredients of the assistant: the handling of heterogeneous sources (Section III-A), the ability to perform queries on them in a uniform way (Section III-B), and the possibility to visualize sources and query results (Section III-C). Section IV describes the extensible and modular architecture of the assistant. Section V presents its integration with DSL-tao and Section VI discusses the results of a preliminary evaluation. Section VII compares with related work and Section VIII concludes the paper and proposes lines for future research.

II. Overview and Running Example

Creating a high quality meta-model is a complex task, as it involves both deep knowledge in a particular domain, and experience in object-oriented design and class-based modelling. Hence, domain meta-modelling typically involves two roles: (i) a domain expert, and (ii), a meta-modelling expert. However, many times, the meta-modelling expert (the developer) is left alone in the construction of a meta-model, or needs to make decisions on tacit domain knowledge or under-specified language requirements. In this scenario, the developer would benefit from access to vocabularies and knowledge in a particular domain, available in sources like ontologies or as RDF data, which could then be easily included in the meta-model being built.

Typically, meta-models within a domain are not completely different from each other, but they have recurring patterns and idioms [27]. For example, when building a language to describe behaviour, designers may resort to accepted specification styles, including variants of languages like state machines, workflow languages, rule-based languages or data-flow languages, enriched with domain-specific elements. Therefore, not only for the purpose of access to domain concepts, but also to organize the meta-model infrastructure, it would be interesting to be able to inspect existing meta-models, ontologies, or any kind of data description artefact, in a uniform way.

For this purpose, we have built a (meta-)modelling assistance system, whose working scheme is shown in Fig. 1. The approach is based on the creation of a set of repositories (label 1 in the Figure), possibly made of heterogeneous data descriptions (OWL ontologies, Ecore meta-models, RDF files, XML schemas). Our system provides facilities for the uniform, flexible ("Google-like") query of such sources (label 2). Then, the results of the query for each source in the repository are aggregated and ranked according to the suitability for the user (label 3). At any point, information sources and query results can be visualized in a graph-based way (label 4). The assistance system is independent of any modelling tool, but has been designed to be easily integrated within them (label 5). This way, query results can be used easily by an external modelling tool by drag & drop.

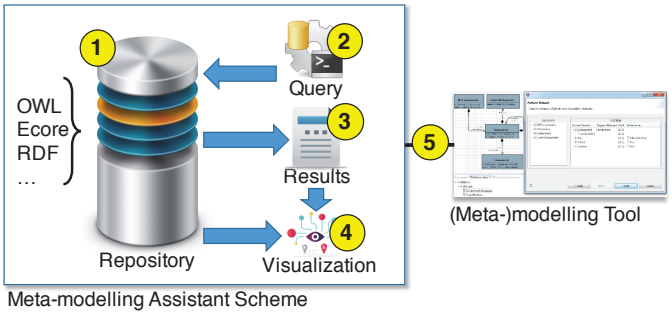

Fig. 1: Overview of the approach

This assistant architecture is useful in several scenarios. EXTREMO was conceived as a meta-modelling assistant, and hence its main application is to help in developing new meta-models for particular domains. In addition is is also useful when creating a "concept" meta-model [5]. A concept is a minimal meta-model that gathers the core primitives within a domain (e.g., for workflow languages). Concepts can be used as the source meta-model of a model transformation, which then becomes reusable, as the concept can be bound to a particular meta-model. Therefore, when creating a concept, one needs to query and understand many different meta-models for a particular domain, and hence the assistant becomes useful. Finally, the assistant can also be useful when creating models for a particular domain. In this paper, we will illustrate mainly the first scenario (creating meta-models) through a running example, while we show that that EXTREMO can also be integrated with modelling tools in Section VI.

A. Running example

As a running example, we will use EXTREMO to construct a DSL for describing administrative e-Government processes for immigration. As an illustration of the kind of concepts we would like to include in the DSL, Fig. 2 depicts a model describing the (simplified) process for issuing a visa.

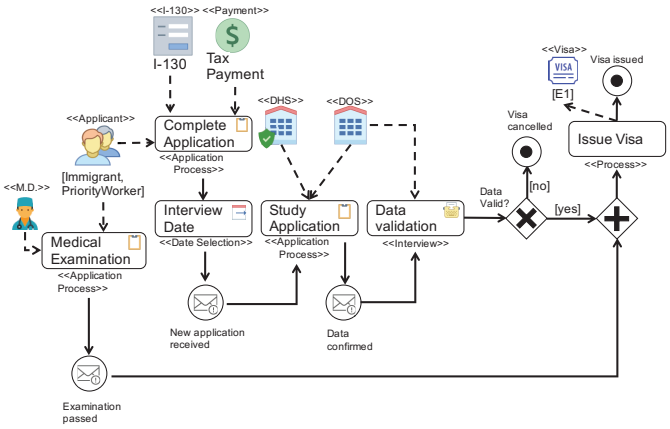

Fig. 2: Process for issuing a visa with the example DSL

In the process, the applicant is required to fill in a standard I-130 form and to perform the required tax payment. In the

visa application, it is necessary to set a date for an interview with the embassy personnel. Eventually, a new application is produced. Afterwards, two governmental organizations – the US Department of Home Security (DHS) and the Department of State (DOS) – perform a study of the submitted application. If positively resolved, the DOS interviews the candidate. In parallel, the candidate is expected to go through a medical examination, which is also required for issuing the visa. Once both the ordinary verifications and the medical report have been checked, the document is dispatched. In this case, the visa is approved and consequently issued; otherwise, the applicant is denied the document. This process describes the application for an E1 visa, while depending on the characteristics of the applicant (e.g., married to a citizen, business visitor), other types of visa should be issued, which would be described as different processes using this DSL.

It can be observed how this DSL contains domain-specific knowledge, related to the organizations (DOS, DHS), the actors (immigrant, medical doctor) and the artefacts (the I-130 form, different types of Visa, tax payment) involved. The DSL also includes specific activities common in the domain, like «Application Process», «Date Selection», or «Interview». In addition, the language has more generic concepts (e.g., tasks, processes, control flow directives, like split and join, etc.) typically found in process modelling languages.

Therefore, as a help in the construction of this DSL, we will gather information from well known standards from the Object Management Group (OMG)[1] (http://www.omg.org/spec/), or meta-models repositories, like the ATL zoo (http://web.emn.fr/x-info/atlanmod/). In these two repositories, we find descriptions of both standard process modelling languages, like BPMN [24] or BPEL [22], or process modelling languages tailored for specific domains, in Ecore/XMI format. This is the standard serialization format for meta-models in EMF [33], the "de-facto" meta-modelling standard in MDE. For the domain concepts, public repositories or open information resources would be required. In our case, we will take the US e-Government domain ontologies (http://oegov.org/), which are available in OWL and RDF formats.

However, it is not enough to have individual access to those data sources. As a help to the meta-model developer, we would need a system able to perform flexible queries uniformly on those heterogeneous sources, to aggregate their query results sorting them by relevance, to visualize the sources and results, and to facilitate the incorporation of the query results into the meta-model being developed. The next section describes how our meta-modelling assistant fulfills these requirements.

III. A META-MODELLING ASSISTANT

In this section, we detail the three main parts of our approach: the handling of heterogeneous sources (Section III-A), the uniform query support (Section III-B), and the uniform visualisation of heterogeneous sources and query results (Section III-C).

[1]The OMG is the standarization body behind many modelling standards like the UML, SysML, MOF or BPMN.

A. Handling heterogeneous sources

Our assistant requires a uniform way to access and query heterogeneous data sources, as well as mechanisms to organize and classify such resources. Fig. 3 shows a class diagram depicting how we have approached both objectives.

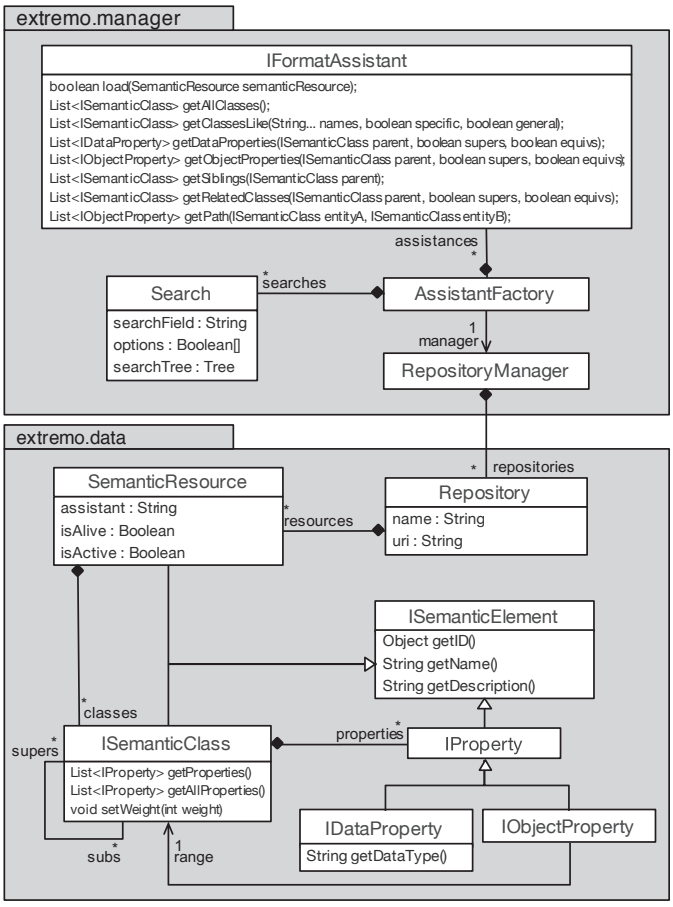

Fig. 3: Class diagram showing EXTREMO's common data model and the repository manager

In order to enable a uniform access to heterogeneous data, we use a common data model, shown in package extremo.data of Fig. 3. The idea is to provide mappings for different data sources to this common model. A SemanticResource represents a resource (e.g., a file) containing a collection of entities. We use a number of interfaces (ISemanticElement, IProperty, IDataProperty, IObjectProperty and ISemanticClass) to represent the main elements found in data description models like meta-models and ontologies, but abstracting from particular technological details.

This way, the data model assumes that every resource will consist of classes (ISemanticClass) made of properties (IProperty), which can have primitive data types (IDataProperty) or references to other classes (IObjectProperty). Classes may be related to super/subclasses via the supers/subs association roles. The model requires mechanisms to access the owned properties (method getProperties) and both to the owned and inherited properties (method getAllProperties) of a class.

	Ecore	OWL	RDF
SemanticClass	EClass	Class	Resource
ObjectProperty	EReference	ObjectProperty	Property
DataProperty	EAttribute	DatatypeProperty	Property
SemClass.supers	eSuperTypes	rdfs:subClassOf	
SemClass.properties	eAttributes eReferences	rdfs:domain	

TABLE I: Correspondence between different data modelling technologies and the EXTREMO data model

Table I shows how several technologies (Ecore, OWL, RDF) can be mapped to our common data model. Our design is extensible, as it facilitates the integration of new technologies. We will elaborate on the extensibility mechanisms of EXTREMO in Section IV.

Adding support for a new data format requires providing a parsing facility into the common data model, as well as some basic query mechanisms, as depicted in the interface IFormatAssistant in Fig. 3. The parsing mechanism (method load) is in charge of mapping the format-specific information to the common data model. The query mechanisms will be described in Section III-B.

The other ingredient of the assistant data handler includes a repository manager and an assistant manager (shown in package extremo.manager of Fig. 3). These elements take care of controlling the different sources and resources. Resources can be either remote or local files; either way, they get classified by the domain in the repositories. A repository is an abstract concept, and are not physically stored but defined only for a classification purpose. The AssistantFactory classifies the resources and controls which assistant will resolve the queries against which resource. This data, coming from arbitrary sources and representation formats, is accommodated into our common framework.

Running Example. In the example, we provide three assistants for the resolution of queries over meta-models (Ecore), ontologies (OWL) and Linked Data resources (RDF/XML). We have organized resources in three repositories, one for the ATL meta-model zoo, another for the OMG standards (both in Ecore format), and the last one for the e-Government ontologies. The ATL zoo repository contains Ecore resources representing different languages in the process modelling and workflow domains. Those include BPEL, BPMN, DoDAF, Gantt, ODP, PIF, SPEM and Workdefinitions. The OMG repository contains Ecore files with standard meta-models, like BMM, BPMN, CMMN, DMN, SBVR and SPEM. Finally, the e-Government repository contains ontologies modelling different facets of the structure of the United States government. These include the Government ontology, the U.S. Government ontology, the Department of Homeland Security ontology and Department of State ontology.

B. Querying

Often, a developer might lack a complete understanding of a certain domain, which may complicate the construction of a meta-model. Hence, we provide query facilities over the uniform data model, which are then realised over the heterogeneous sources. Our query facilities are intended to provide support for flexible search (i.e., "Google-like" queries) as opposed to the queries that one would formulate in languages like OCL or SQL. We selected this option as the purpose is to find any possible relevant information in the repositories in an exploratory way.

In order to be able to query heterogeneous sources in a uniform way, we rely on parsing the heterogenous data into a common model (see Fig. 3). Such common data model is also needed to provide support to non-well structured formats (e.g., plain text files) when naming and listing their entities. Finally, the common data model also facilitates connecting the assistant with linguistic and lexical databases that enable performing more flexible queries on data sources.

As the IFormatAssistant interface in Fig. 3 shows, our assistant supports queries to:

- Obtain all semantic classes in a resource.
- Obtain all classes, whose name is similar to a set of words that the user specifies. Additionally, the user may select whether she would like to obtain all more general or specific classes than the ones sought.
- Get all (data and object) properties of a semantic class, including inherited ones.
- Retrieve all classes that inherit from a common superclass (sibling classes).
- Obtain all classes related with a given one.
- Get a path of relations between two semantic classes.

Because of EXTREMO's common data model, these operations can be performed regardless of the format of each data resource, as every entity and its properties are presented in a uniform representation.

In our approach, when a semantic class is sought, the process encompasses two main stages. First, the user provides a list of *tentative* words as search terms. Then, this list is processed using Wordnet [21], a lexical database of the English language, with the objective of looking for relevant synonyms of the chosen search terms. Because a word may have different senses, some of them irrelevant for the domain of the meta-model being built, we need a mechanism to choose synonyms from relevant word senses. Hence, we perform a processing step, which constructs a word tree rooted in the input concepts and spreading to their closer synonyms and related concepts and expressions. Finally, the query algorithm looks for the semantic classes – extracted uniformly from the resources – matching the input terms and their synonyms and goes over the different semantic paths connecting all of them.

The way we explore the relations between search terms in a given context is inspired by the Lesk algorithm [18]. Our algorithm evaluates each word in a phrase assuming that every word tends to share a common semantic field with its siblings. We use Wordnet to select, among the available entities, the most suitable candidates to match the common semantic fields from the input term list.

For example, Fig. 4 shows an abstraction of the results found for the search "process, activity, task". For space reasons, we represent the senses of each word with an "s" and the list

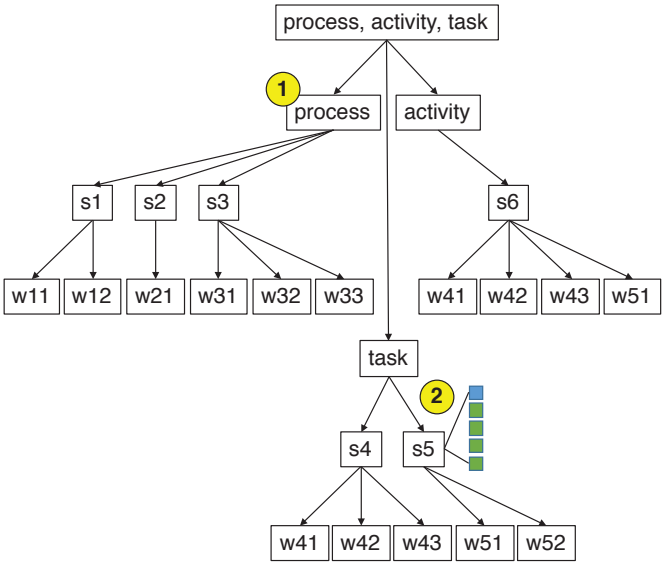

Fig. 4: Search terms tree definition expansion

of synonyms within each sense with a "w". For example, in the case of "process" (label 1), one of the senses given by Wordnet is *"a natural prolongation or projection from a part of an organism either animal or plant"*. For this sense, both "outgrowth" and "appendage" are considered synonyms of process. To determine if a sense has significance for the search we make use of the definition of the sense and the list of example phrases provided by Wordnet (depicted as colored rectangles in label 2). This evaluation continues incrementally until the final list of synonyms is found. The final list will be used to search in the resources within the repository.

To obtain the list of candidates, we use a rule system based on assigning points as the whole list of senses provided by Wordnet is evaluated. The points are not fixed, but can be configured in our tool. Some point strategies are predefined by the tool, but new strategies can be added as well. The rules for assigning those points are as follows.

- $\mathbf{S}_1$: If a synonym is present in the list of tentative words (in Fig. 4, say w11 = 'activity').
- $\mathbf{S}_2$: If a synonym of a sibling term is present in the senses of the current term (w41 = w11).
- $\mathbf{S}_3$: If a path between a synonym of a sibling term and one of the synonyms exists (w41 -> w11).
- $\mathbf{S}_4$: If a synonym of a sibling term is mentioned in the definition of one of the senses (w41 mentioned in the definition of s1).
- $\mathbf{S}_5$: If a synonym of a sibling node is mentioned in the usages of one of the senses (w41 mentioned in the usages of s1).

The final points for each synonym are the sum of points provided by the previous rules (S_1, ..., S_5). While we normally use a strategy that assigns more points to the words matched rules with lower numbers (i.e., $points(S_i) > points(S_{i+1})$), other strategies are possible as well, as we will discuss in Section IV.

It is worth mentioning that we detect and tackle compound names. It is quite common, specially in MDE, to find entities in meta-models or ontologies, whose name is the concatenation of two or more words, most often in camel case. For evaluating the search, we split these words. Moreover, plainly comparing word pairs (e.g., "process", "procedure") might throw unsatisfying, or too general, results. Hence, in order to make the search more complete, we used the Porter Stemmer Algorithm [28], which reduces the comparison of two words to their lexical roots.

The resulting words to be sought are ordered descendingly by their assigned points. As the search may match entities in more than one resource, those entities are grouped together and their group receives the maximum of all points received by any of its members.

Running example. For the case of "process, activity, task" search, we followed a strategy that gives more importance to the synonyms presence and the definitions, which assigned 1000 points to the $\mathbf{S}_1$ rule, 80 to $\mathbf{S}_2$, 20 to $\mathbf{S}_3$, 100 to $\mathbf{S}_4$ and 20 to $\mathbf{S}_5$. The resulting list of candidate words was:

1000 task
1000 process
1000 activity
320 job
320 chore
300 summons
260 body process
260 bodily process
260 bodily function
240 procedure
200 treat
200 physical process
200 outgrowth
200 appendage
180 natural process
180 natural action
100 work on
100 work
100 unconscious process
100 swear out
100 sue
100 serve
100 march
100 litigate
100 action
0 ~~undertaking~~
0 ~~project~~
0 ~~operation~~
0 ~~mental process~~
0 ~~labor~~
0 ~~cognitive process~~
0 ~~cognitive operation~~
0 ~~activeness~~

C. Visualization

The visual exploration of the content of the resources is another way to discover and comprehend the semantic concepts that pertain a domain. Based on the common data model, our assistant permits the homogeneous visualization of heterogeneous data sources. The visualization is graph-based, where semantic classes are represented as nodes, and inheritance and relations as two different types of edges. Fig. 5 shows how the different elements in the data model are represented in a graph-based way. We will discuss the capabilities of the supporting visualization tool in Section IV.

IV. Architecture and Tool Support

In order to facilitate its integration with (meta-)modelling tools, the assistant has been realized as a plugin for the Eclipse platform. EXTREMO is available at http://miso.es/

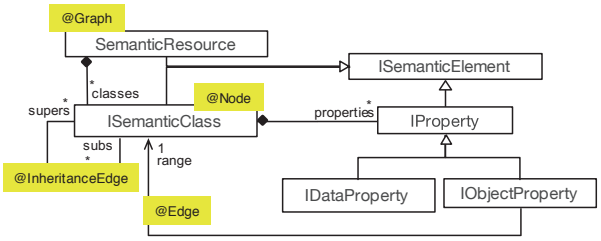

Fig. 5: Mapping the data model to a graph representation

tools/extremo.html. The web page includes videos, screenshots and installation details.

We take advantage of the Eclipse extension point infrastructure to make it easier for the developer to add new formats to the framework (by extending the classes shown in Fig. 3), and for its integration with modelling and meta-modelling tools. Moreover, we provide tool support for the classification of repositories independently of their format, and to help organizing the data sources as of their domain.

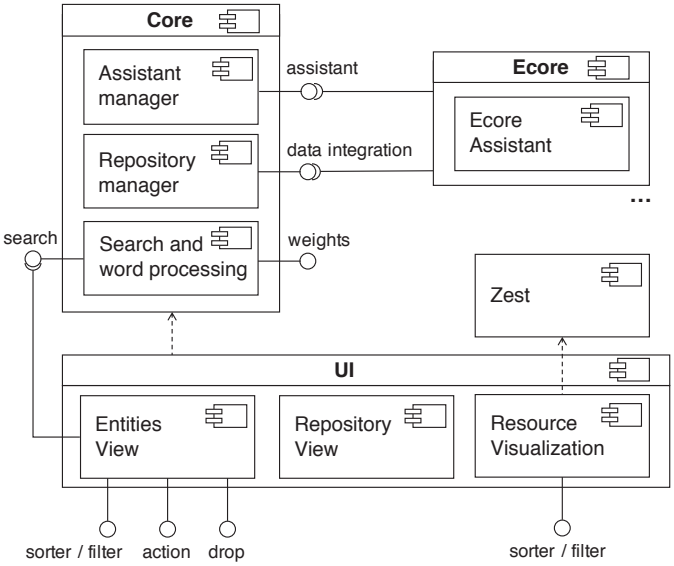

Fig. 6: Architecture of the proposal

A schema of the EXTREMO architecture is shown in Fig. 6. EXTREMO is made of a Core component, which provides support for the common data model and includes subcomponents for searching, repository management, and assistant management. The Core component provides two extension points to support additional data modelling technologies. The Figure shows an extension for the case of Ecore meta-models [33].

When a developer needs support for a new format (e.g., XML schemas), an implementation of the IFormatAssistant interface shown in Fig. 3 needs to be provided. This implementation class is in charge of parsing the data from the resources to the semantic classes. The assistant for the new format has to extend the core components of EXTREMO, both the assistant infrastructure and the data integration that provides the connection with the semantic classes of our data model.

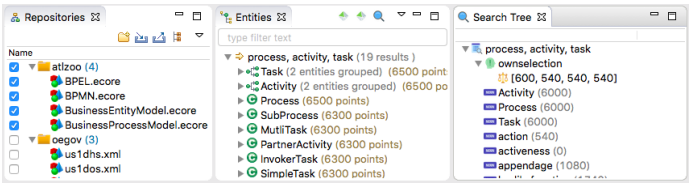

Fig. 7: Repositories (left), Entities (center) and Search Tree (right) views

Every extension needs to define its name, so that EXTREMO is able to distinguish them when more than one assistant per file extension is declared. EXTREMO automatically assigns the assistant to each resource while the repositories are imported.

The UI component in Fig. 6 provides Eclipse views for performing the queries and listing the results (entities view), organizing the repositories (repository view), managing the points given to the rules described in Section III-B and visually exploring a resource.

EXTREMO is useful in combination with data description tools, like meta-modelling environments, but is also useful as an assistant for modelling environments. For this integration, EXTREMO provides two extension points (named drop and action in Fig. 6), which allow the results of queries from the entities view to be inserted into the modelling tool via drag & drop and menu actions.

Fig. 7 shows a screenshot of the repository view (left), the entities view (middle) and the tree of words used in the search (right) that EXTREMO offers. Repositories and resources can be set to active or inactive. In the latter case, they will not be considered in searches. Whenever a search is triggered, the assistant of each available format resolves the query (as explained in Section III-B) against the resources selected in the repository view. Once the search has been resolved, the view shows the entities found by the list of term-synonyms and suggests an ordered list based on the points given during the evaluation of repositories. The tool features two extension points to change this order (an extension point for sorting and another one for filtering).

The middle of Fig. 7 shows the entities view, with the results of a query. The results include two grouped results for Task and Activity, and one result for Process. It can be seen how the view permits navigating through the elements of a semantic class, including its super-classes, sub-classes, references and attributes. As we will see in next section, when integrated with a meta-modelling tool, it will be possible to select this information and incorporate it into the meta-model being built.

The points given by the rules of Section III-B can be managed through a preference page, shown in Fig. 8. This allows changing the importance to the premises used by the rules. Points can be changed individually, or selecting strategies defined by the *weights* extension point in Fig. 6.

EXTREMO also offers visualization mechanisms for the resource contents. These mechanisms are based on Zest (see https://www.eclipse.org/gef/zest/), a component for graph-based visualization, which provides support for filtering and layouts. Visualization filtering can be customized by an ex-

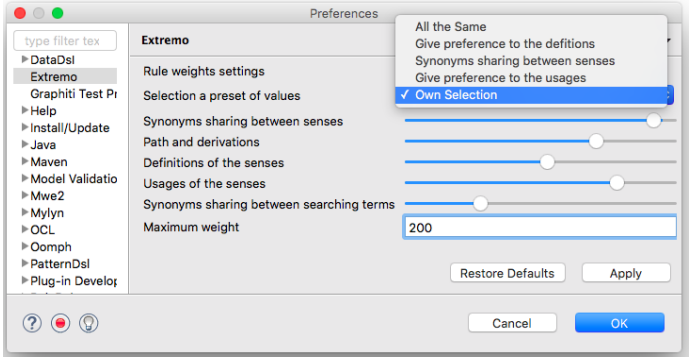

Fig. 8: Preference page of EXTREMO

tension point. The tool offers two views: one to explore the inheritance relationship between ISemanticClasses and the other to show the references IObjectProperty among them. Fig. 9 shows a screenshot of the entities contained in the BPMN resource and its references.

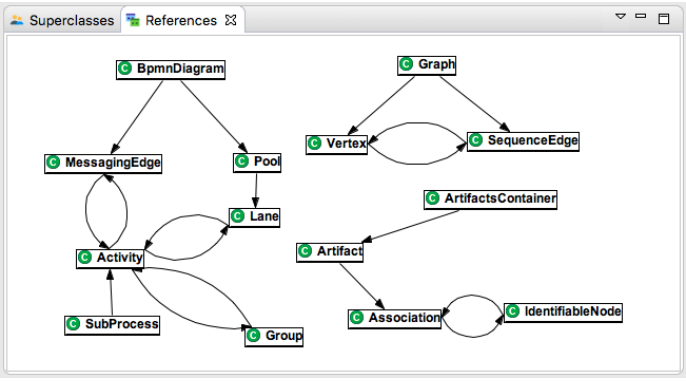

Fig. 9: Visualizing BPMN resource references with EXTREMO

V. EXAMPLE: INTEGRATION WITH DSL-TAO

As a proof of concept demonstrating the extensibility and usefulness of EXTREMO, in this section we show its integration with DSL-tao, a meta-modelling tool developed by our group. Hence, we present the main features of DSL-tao in Section V-A, and explain how it was integrated with EXTREMO in Section V-B.

A. Meta-modelling in DSL-tao

DSL-tao [27] is a tool (an Eclipse plugin) for the construction of DSLs using a pattern-based approach. It is freely available at http://miso.es/tools/DSLtao.html, and the web site includes videos and instructions for installation.

Designing a DSL involves defining its abstract syntax (the meta-model), concrete syntax (the visualization, graphical or textual) and semantics (the meaning of models, typically by an interpreter or a code generator). In addition, editing DSL models is usually performed using a dedicated environment providing services like model persistence, conformance checking, and others more advanced. DSL-tao proposes the use of patterns to address all these aspects in order to facilitate and speed up the construction of the meta-model and the DSL modelling environment.

To deal with the abstract syntax, DSL-tao introduces the notion of domain pattern [27]. A domain pattern gathers typical requirements of similar languages within a domain, documenting their variability. Here, there may be patterns for workflow languages, arithmetical or logical expressions, variants of state machines, query languages, and component-based architectural languages, among others. A DSL may use several domain patterns, customized for a given need, and extended with other domain-specific concepts. Hence, domain patterns help the developer to create a meta-model more quickly, reusing accepted idioms within a domain.

Patterns may include services, which typically contribute functionality to the generated environment via code generation. Such kind of patterns are called infrastructure patterns.

B. Integrating EXTREMO in DSL-tao

To integrate EXTREMO in a (meta-)modelling tool, the tool needs to implement one of the extension points provided by EXTREMO, as depicted in Fig. 6. One of the extension points enables the addition of an action to the semantic elements shown in the entity view of Fig. 7, both in the contextual menu and in the toolbar. The other extension point enables the drag operation from the entity view and dropping the selected information into any graphical editor based on Graphical Editing Framework [12]. This is the underlying framework of Eclipse graphical editors. For DSL-tao, we opted for an integration based on drag&drop, which enables a selection of the position in the canvas where the element is to be created.

In addition to implement the extension point, we created a pattern, called *Recommended*, to help documenting the information introduced from an external source. Hence, the first time a new class, attribute or reference is dropped from EXTREMO into DSL-tao, a pattern instance is created. Then, the rest of the elements will be added to this pattern instance. The pattern instance gathers the resource the element was obtained from, the kind of resource (meta-model, ontology, etc.), and the rationale. While the two first fields are automatically filled in the drop operation, the latter is used to document the reasons to introduce the element in the meta-model.

Fig. 10 shows DSL-tao being used in combination with EXTREMO. The canvas (label 5) holding the meta-model being built, and the applied patterns view (label 2) belong to DSL-tao, while labels (1, 3, 4) show views contributed by EXTREMO. The view labelled (1) in the figure shows the repositories being managed (OMG standard meta-models, ATL ecore meta-models, ontologies). The view labelled (2) is the applied patterns view which shows several applications of the *Recommended* pattern, produced by the incorporation of classes *Process*, *SequenceEdge*, *MessagingEdge* and *Activity* from the EXTREMO results view. In this applied pattern view, the user may add information about the provenance and rationale of the incorporated information. Moreover, it serves as a quick means to locate the introduced classes from external

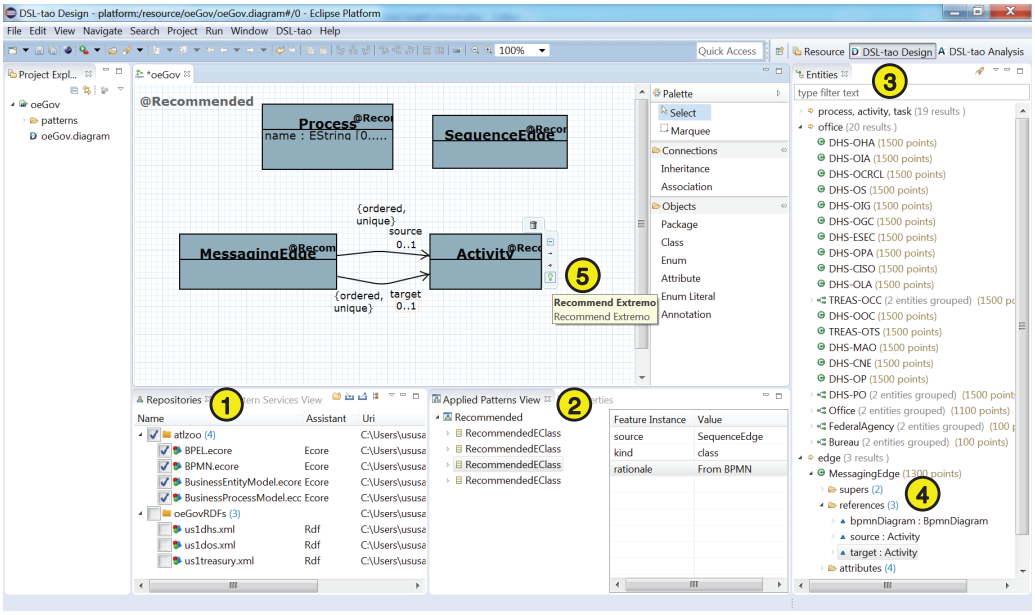

Fig. 10: Using DSL-tao in combination with EXTREMO

sources: clicking on one of the classes in the view selects the corresponding class of the canvas.

In the entities view (label 3), a search for "process, activity, task" is being performed and a list of entities prioritised by relevance is served (label 4). Dropping the entity from the view on the model being built inserts a new class in the diagram (label 5), and produces a new instance of the *Recommended* pattern. In addition, DSL-tao was extended with the possibility of showing contextual menus in the classes (label 5), so that the query facilities of EXTREMO can be invoked, taking the name of the class as search term.

VI. PRELIMINARY ASSESSMENT

In this section we provide some metrics and lessons learnt, which provide a preliminary assessment of the extensibility of EXTREMO, and how easy it is to integrate it with meta-modelling tools, and its usefulness for creating (meta-)models.

DSL-tao was developed by our group. Hence, to further assess how easy it is the integration with other tools, we performed two more experiments. Both integrate EXTREMO with tools developed by third-parties (and in which the code was not available). The first one integrated EXTREMO in a UML modelling tool[2] (as shown in Fig. 11), and the second in the standard Ecore editor. In both cases, the extension was performed using the *action* extension point. A drag&drop integration was not possible, because we would have needed to change the source code of the editors.

Table II shows the LOC necessary for each integration. In average, the integration amounted to 170 lines of Java code (LOC). All the code needed was related to the transformation from semantic entities to the classes of the modelling tool. In the DSL-tao case, we provided an option in the contextual

[2]UML2-MDT, www.eclipse.org/modeling/mdt

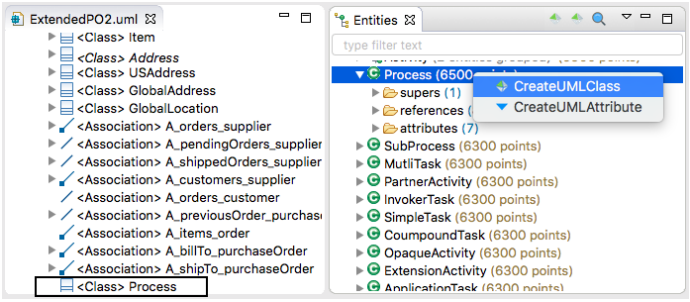

Fig. 11: Integration of EXTREMO in UML2

menu to directly invoke EXTREMO queries. This resulted in 17 LOC. To further take advantage of the information gathered with EXTREMO, we implemented the *Recommended* pattern, as a means to record the rationale for incorporating the information into the meta-model. While this functionality was not needed to integrate EXTREMO, it shows a typical usage of the external information. This amounted to 20 LOCs.

	DSL-tao	**EcoreEditor**	**UML2Editor**
Extension Point Used	drop	actions	actions
Extension Point Integration	35	5	5
New Class	24	64	68
New Attribute	27	66	62
New Reference	24	66	64
Total	110 LOC	201 LOC	199 LOC

TABLE II: LOCs for integrating EXTREMO with other tools

Hence, we can conclude that the cost of integrating EXTREMO with a (meta-)modelling tool is very light and can be performed with no access to the source code. However, the integration was made by the main EXTREMO developer, and a further validation with other developers would be needed. EXTREMO was integrated with a UML tool, which shows that

it can also be used as a modelling assistant. However, we realized that for its use with arbitrary modelling languages, it would be useful to query repositories of models. For that purpose, taking into account the meta-model (e.g., so that queries can provide filters by type) of the models in the respository would be highly beneficial. We leave these extensions for future work.

Next, we look at the extensibility of EXTREMO regarding information formats. Currently, we support OWL, Ecore models and RDF. Each of these cost less than 200 LOC. Hence, we can also conclude that the cost of integrating new sources is not high. However, all the format assistants were created by the main developer of EXTREMO. To further validate the extension facilities, we should perform more experiments, measuring the effort spent by a different developer.

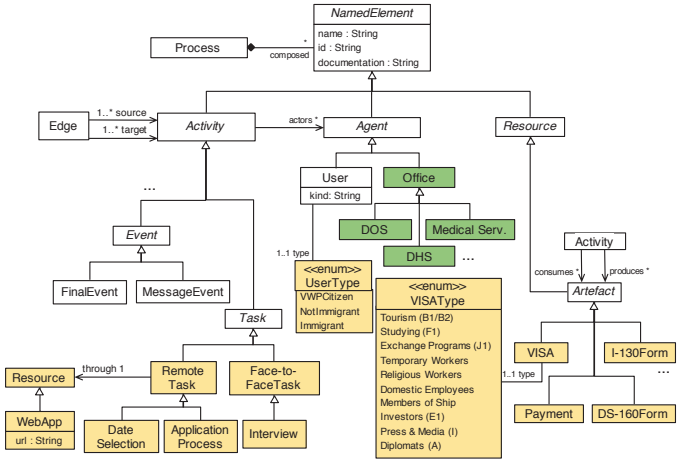

Fig. 12: Meta-model excerpt for the running example

Finally, we look at the amount of information gathered from external sources for the running example. Fig. 12 shows an excerpt of the resulting meta-model for the e-Government application. Roughly 40% of the classes in our solution have been obtained from different ontologies (shaded in yellow), 14% classes from open RDFs (in green) and the remaining classes have been obtained by combining different representations of process management meta-models. While this suggests that EXTREMO is useful as a help construct the meta-model, a further user study is needed to further validate this claim.

VII. RELATED WORK

The increasing complexity of software development has prompted the need for code recommenders for example, for API usage, or program quick fix. However, while code recommenders are increasingly used in programming IDEs [29], [11], there is lack of such tool for (meta-)modelling in MDE.

The closest work to our proposal is [8], [9], [10], where a generic architecture for model recommenders is proposed. The architecture is extensible, in the sense that different recommender strategies can be plugged-in. In contrast, the extensibility of our approach is in the supported data source, while we specifically focus on the extraction of knowledge from these sources. In addition, our approach supports out-of-the-box visualization and query aggregation facilities.

Other approaches to model recommendation focus on proposing suitable ways to complete a model with respect to a meta-model [31]. Hence, using constraint solving techniques, the system proposed ways to complete a model so that it becomes a valid instance of a meta-model. In [36] the authors use ontologies in combination with domain-specific modelling, and hence can use ontology reasoners to provide reasoning services like model validation, inconsistency explanation, and services to help in the model construction phase. These approaches are not applicable in our context, since our purpose is to create a meta-model.

Some researchers have exploited ontologies for creating DSLs [34]. For example, in [3] the authors advocate the use of (OWL) ontologies in the domain analysis phase of DSL construction. As they target textual DSLs, they propose a tool for the automated generation of a textual grammar for the DSL. In a similar vein, in [23], the authors generate meta-model design templates from OWL ontologies, which are later manually refined into domain meta-models. In our approach, we assume that not all the required information to create a meta-model is present in one ontology, but typically such information is scattered in informational resources of different kinds, like ontologies, RDF data, or meta-models.

Combining modeling approaches from MDE with ontologies has been studied in the last decade [15]. There are several approaches to transform Ecore-based models to OWL and back, e.g., cf. [35], [16]. In addition, there exist approaches that allow for the definition of ontologies in software modeling languages such as UML by using dedicated profiles [20]. Moreover, there are approaches which combine the benefits of models and ontologies such as done in [26], [25] for reasoning tasks. Not only the purely structural part of UML is considered, but some works also target the translations of constraints between these two technical spaces by using an intermediate format [7]. For the data import, we may build on these mentioned approaches, but we focus on recommendation services exploiting the imported data from different technical spaces to build domain-specific modeling languages.

Finally, there are some approaches directed to search relevant models within a repository. Their aim is slightly different from our goal, which is looking for relevant information within a repository. Moogle [19] is based on textual, “Google-like” queries, similar to ours. As they focus on EMF model-level queries, they use the meta-model information for filtering. We plan to add this functionality in the future. Their results are shown in textual format, while we parse and aggregate the results, and offer graphical visualization. EMF query is directed to search EMF models [14], using OCL queries or text-based search. The latter may include regular expressions, but does not look for relevant synonyms as we do. Moreover, our extensible approach supports technologies like Ecore, OWL and RDF. Furthermore, there are dedicated approaches offering search capabilities tailored for a specific modelling domain such as [1], [6]. While these approaches also allow

to reason on behavioral similarity aspects, we aim for general model search support irrespectively of the modelling domain and even of the technical space.

VIII. Conclusions and future work

In this paper, we have presented EXTREMO, an extensible assistant for meta-modelling. The system is able to gather information from different information sources (like ontologies, RDF or meta-models), by enabling uniform querying and visualization. EXTREMO is independent of the particular meta-modelling tool, but easily integrated with them due to its modular architecture based on extension points. We have shown its integration with DSL-tao and two other tools, and used it for the construction of a DSL in the e-Government domain. A preliminary assessment shows promising results in terms of simplicity of integration, extensibility, and usefulness.

In the future, we plan to connect EXTREMO with meta-model repositories, such as MDEForge [30]. We have integrated EXTREMO into DSL-tao using a re-active mode, where the assistant should be explicitly invoked. Similar to [9], we would also like to explore pro-active modes for assistance. To make EXTREMO useful as a modelling assistant, we plan to enable queries at the model level. This would require taking into account the meta-model the queried models are conformant to (e.g., to filter the query results, and to be able to create objects using the query results). Finally, we plan to perform a detailed study on the effects of different point strategies for search, and a user study to better assess EXTREMO's value for meta-modelling, and gather suggestions for improvement.

Acknowledgements

This work is supported by the Ministry of Education of Spain (FPU grant FPU13/02698), the Spanish MINECO (TIN2014-52129-R), the R&D programme of the Madrid Region (S2013/ICE-3006), the EU commission (FP7-ICT-2013-10, #611125), and by the Christian Doppler Forschungsgesellschaft and the BMWFW, Austria.

References

[1] B. Bislimovska, A. Bozzon, M. Brambilla, and P. Fraternali. Textual and content-based search in repositories of web application models. *TWEB*, 8(2):1–11, 2014.

[2] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[3] I. Ceh, M. Crepinsek, T. Kosar, and M. Mernik. Ontology driven development of domain-specific languages. *Comput. Sci. Inf. Syst.*, 8(2):317–342, 2011.

[4] T. Clark, V. Kulkarni, B. Barn, R. B. France, U. Frank, and D. Turk. Towards the model driven organization. In *HICSS*, pages 4817–4826, 2014.

[5] J. S. Cuadrado, E. Guerra, and J. de Lara. A component model for model transformations. *IEEE TSE*, 40(11):1042–1060, 2014.

[6] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, 2011.

[7] D. Djuric, D. Gasevic, V. Devedzic, and V. Damjanovic. A UML Profile for OWL Ontologies. In *MDAFA*, pages 204–219, 2004.

[8] A. Dyck, A. Ganser, and H. Lichter. Enabling model recommenders for command-enabled editors. In *MDEBE*, pages 12–21, 2013.

[9] A. Dyck, A. Ganser, and H. Lichter. A framework for model recommenders - requirements, architecture and tool support. In *MODELSWARD*, pages 282–290, 2014.

[10] A. Dyck, A. Ganser, and H. Lichter. On designing recommenders for graphical domain modeling environments. In *MODELSWARD*, pages 291–299, 2014.

[11] Eclipse Code Recommenders. http://www.eclipse.org/recommenders.

[12] Eclipse Graphical Editing Framework. https://eclipse.org/gef/.

[13] Eclipse UML 2 support. http://wiki.eclipse.org/MDT/UML2.

[14] EMF Query. https://projects.eclipse.org/projects/modeling.emf.query.

[15] D. Gasevic, D. Djuric, and V. Devedzic. *Model Driven Engineering and Ontology Development*. Springer, 2009.

[16] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *MoDELS*, pages 528–542, 2006.

[17] S. Kelly and J. Tolvanen. *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley, 2008.

[18] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *SIGDOC*, pages 24–26. ACM, 1986.

[19] D. Lucrédio, R. P. de Mattos Fortes, and J. Whittle. MOOGLE: a metamodel-based model search engine. *Software and System Modeling*, 11(2):183–208, 2012.

[20] M. Milanovic, D. Gasevic, A. Giurca, G. Wagner, and V. Devedzic. Towards Sharing Rules Between OWL/SWRL and UML/OCL. *ECEASST*, 5, 2006.

[21] G. A. Miller. Wordnet: A lexical database for english. *Comm. ACM*, 38(11):39–41, 1995.

[22] OASIS Web Services Business Process Execution Language (WSBPEL). https://www.oasis-open.org/committees/wsbpel/.

[23] A. Ojamaa, H. Haav, and J. Penjam. Semi-automated generation of DSL meta models from formal domain ontologies. In *MEDI*, pages 3–15, 2015.

[24] OMG Business Process Model and Notation. http://www.bpmn.org/.

[25] F. S. Parreiras and S. Staab. Using ontologies with UML class-based modeling: The TwoUse approach. *DKE*, 69(11):1194–1207, 2010.

[26] F. S. Parreiras, S. Staab, and A. Winter. On marrying ontological and metamodeling technical spaces. In *FSE*, pages 439–448, 2007.

[27] A. Pescador, A. Garmendia, E. Guerra, J. S. Cuadrado, and J. de Lara. Pattern-based development of domain-specific modelling languages. In *MoDELS*, pages 166–175, 2015.

[28] M. F. Porter. An algorithm for suffix stripping. *Program*, 40(3):211–218, 2006.

[29] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[30] J. D. Rocco, D. D. Ruscio, L. Iovino, and A. Pierantonio. Collaborative repositories in model-driven engineering. *IEEE Software*, 32(3):28–34, 2015.

[31] S. Sen, B. Baudry, and H. Vangheluwe. Towards domain-specific model editors with automatic model completion. *Simulation*, 86(2):109–126, 2010.

[32] T. Stahl and M. Völter. *Model-driven software development - technology, engineering, management*. Pitman, 2006.

[33] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2008.

[34] R. Tairas, M. Mernik, and J. Gray. Using ontologies in the domain analysis of domain-specific languages. In *TWOMDE*, pages 20–31, 2008.

[35] T. Walter, F. S. Parreiras, G. Gröner, and C. Wende. OWLizing: Transforming Software Models to Ontologies. In *ODiSE*, pages 7:1–7:6, 2010.

[36] T. Walter, F. S. Parreiras, and S. Staab. An ontology-based framework for domain-specific modeling. *SoSyM*, 13(1):83–108, 2014.