

Automated modelling assistance by integrating heterogeneous information sources

Ángel Mora Segura^a, Juan de Lara^a, Patrick Neubauer^b, Manuel Wimmer^b

^a*Modelling & Software Engineering Research Group*
<http://miso.es>

Universidad Autónoma de Madrid (Spain)

^b*CDL-MINT, Business Informatics Group*

<https://www.big.tuwien.ac.at>
TU Wien (Austria)

Abstract

Model-Driven Engineering (MDE) uses models as its main assets in the software development process. The structure of a model is described through a meta-model. Even though modelling and meta-modelling are recurrent activities in MDE and a vast amount of MDE tools exist nowadays, they are tasks typically performed in an unassisted way. Usually, these tools cannot extract useful knowledge available in heterogeneous information sources like XML, RDF, CSV or other models and meta-models.

We propose an approach to provide modelling and meta-modelling assistance. The approach gathers heterogeneous information sources in various technological spaces, and represents them uniformly in a common data model. This enables their uniform querying, by means of an extensible mechanism, which can make use of services, e.g., for synonym search and word sense analysis. The query results can then be easily incorporated into the (meta-)model being built. The approach has been realized in the EXTREMO tool, developed as an Eclipse plugin.

EXTREMO has been validated in the context of two domains – production systems and process modelling – taking into account a large and complex industrial standard for classification and product description. Further validation results indicate that the integration of EXTREMO in various modelling environments can be achieved with low effort, and that the tool is able to handle information from most existing technological spaces.

Keywords: Modelling, (Meta-)modelling, Modelling assistance, Domain-specific languages, Language engineering

Email addresses: Angel.MoraS@uam.es (Ángel Mora Segura), Juan.deLara@uam.es (Juan de Lara), neubauer@big.tuwien.ac.at (Patrick Neubauer), wimmer@big.tuwien.ac.at (Manuel Wimmer)

1. Introduction

Model-Driven Engineering (MDE) advocates an active use of models throughout the software development life-cycle. Thus, models can be used to specify, analyse, test, simulate, execute, generate code and maintain the software to be built, among other activities [1, 2, 3].

Models are sometimes built with general-purpose modelling languages, such as the Unified Modelling Language (UML) [4]. In other cases, modelling is performed using Domain-Specific Languages (DSLs) [5]. DSLs contain tailored domain-specific primitives and concepts accurately representing the abstractions within a domain, which may lead to simpler, more intensional models. The abstract syntax of a DSL is described by a meta-model, which is itself a model. Meta-models are typically built using class diagrams, describing the set of models considered valid. Thus, the construction of models and meta-models is a recurrent and central activity in MDE projects [6].

High quality models and meta-models are pivotal for the success of MDE projects. They capture the most important concepts of a domain or describe the features of a system. Nevertheless, they are mostly built in an unassisted way, with no mechanisms for reusing existing knowledge. This situation contrasts with modern *programming* IDEs, which support code completion or provide help for using a given API [7, 8]. However, in the MDE field, the modeller normally has the burden of creating the model from scratch. For this reason, modellers would greatly benefit from flexible access and reuse of existing knowledge in a domain. This knowledge might be stored on various technological spaces [9, 10], including the modelling technical space, but also the XML, ontologies, and RDF technical spaces.

In order to improve this situation, we propose an extensible approach that provides assistance during the modelling process. In our proposal, we extract the information from an extensible set of different technical spaces. For example, in the XML technical space, DTDs or XML schemas as well as specific XML documents are covered by the assistant; while in the modelling technical space, meta-models and models can be queried. This heterogeneous information is stored in a common data model, so that it can be queried and visualized in a uniform way. The query mechanism is extensible and can make use of services, e.g. for synonym search or word sense analysis. The results of the queries are prioritized and aggregated for all information sources in the repositories and can then be incorporated into the (meta-)model under construction.

We have realized this concept in EXTREMO and provide an open source Eclipse plugin, which is freely available at the EXTREMO project website¹. The web site includes short videos illustrating the main concepts explained in this paper as well as a set of resources, which have been used during the evaluation. EXTREMO's architecture is extensible and modular by the use of Eclipse extension points, and enables the addition of new information sources and types of

¹<http://miso.es/tools/extremo.html>

43 queries. The assistant has been designed to be easily integrated with external
44 modelling environments, also through extension points.

45 We have evaluated our approach under several perspectives. First, we show
46 EXTREMO’s usefulness to create DSLs in two case studies. The first one is in
47 the area of process modelling for immigration procedures and the second is in
48 the area of standard-conforming industrial production systems. We have eval-
49 uated its extensibility by describing its integration with a variety of modelling
50 tools, ranging from graphical to tree-based editors. In order to evaluate format
51 extensibility (i.e., the ability to import from new technical spaces), we perform
52 an analytical evaluation of the degree of coverage of the data model. The query
53 mechanism is tested by describing a catalogue of common queries for object-
54 oriented notations. Finally, we address a discussion and the lessons learned
55 from the results of the evaluation.

56 In comparison with our previous work [11], we provide extensions for a set
57 of different technical spaces that include constraint interpreters and an exten-
58 sible query mechanism. Moreover, EXTREMO’s internal data model has been
59 extended to handle level-agnostic information, i.e., for an arbitrary number
60 of meta-levels. For example, we have integrated XML schemas and multi-level
61 models [12] as information sources, and integrated EXTREMO with further mod-
62 elling and meta-modelling environments. Finally, we report on an evaluation
63 based on process modelling and production systems domain case study, using
64 the eCl@ss standard [13] and provide an analytical evaluation on the generality
65 of the data model we propose.

66 The rest of this paper is organized as follows. Section 2 provides an overview
67 of the approach and its motivation. Section 3 explains the main parts of the
68 assistant: the handling of heterogeneous sources (Section 3.1), the ability to
69 perform queries on them in a uniform and extensible way (Section 3.2), and
70 the handling of constraints (Section 3.3). Section 4 describes the extensible and
71 modular architecture of the assistant, and how it can be integrated with mod-
72 elling and meta-modelling tools. Section 5 evaluates the approach under three
73 different perspectives, which include the usefulness for (i) language engineer-
74 ing, (ii) data extensibility, and (iii) integrability with external tools. Section 6
75 compares with related work, and Section 7 presents the conclusions and lines
76 for future research.

77 2. Motivation and overview

78 Many technical tasks in software engineering require from access to knowl-
79 edge found in a variety of formats, ranging from documents in natural lan-
80 guage, to semi-structured and structured data. There is a current trend to
81 make such information readily available and easy to embed in different types of
82 artefacts generated during the software construction process [14]. For example,
83 in the programming community, there are efforts to profit from code reposi-
84 tories, and Q&A sites like StackOverflow to automate coding and documentation
85 tasks [15, 16, 17]. Some of these approaches are based on a phase of artefact
86 collection, followed by their preprocessing and storage into a uniform database,

87 which then can be queried using appropriate languages [16]. Following this
88 trend, our objective is to make available to the model engineer a plethora of
89 (possibly heterogeneous) resources that can be queried in a uniform way, and
90 embedded into the model being built.

91 In general, the task of creating a high quality meta-model is complex because
92 it involves two roles: (*i*) a domain expert, who has in-depth knowledge of a
93 particular domain and (*ii*), a meta-modelling expert, who is experienced in
94 object-oriented design and class-based modelling. Nevertheless, many times, the
95 meta-modelling expert is left alone in the construction of a meta-model, or needs
96 to make a decision based on tacit domain knowledge or under-specified language
97 requirements. In this scenario, the meta-modelling expert takes the role of the
98 domain expert too, which may lead to mistakes or omissions, compromising the
99 quality of the meta-model.

100 Meta-models within a domain are not completely different from each other,
101 but they sometimes have recurring patterns and use common idioms to represent
102 concepts [18, 19]. For example, while building a language to describe behaviour,
103 designers normally resort to accepted specification styles, including variants of
104 languages such as state machines, workflow, rule-based or data-flow languages,
105 enriched with domain-specific elements. The language designer can obtain this
106 information from sources like meta-models, class diagrams, ontologies, XML
107 schema definitions (XSD) or RDF documents. Moreover, having access to a va-
108 riety of information sources helps in obtaining the necessary domain knowledge,
109 vocabulary and technical terms required to build the meta-model. This situa-
110 tion also applies when building models, instances of a given meta-model. In this
111 case, it may be helpful to have a way to query information sources and knowl-
112 edge bases. These queries may use the data types present in the meta-model to
113 help filtering the relevant information.

114 For this purpose, we have devised a modelling assistance approach, whose
115 working scheme is shown in Figure 1. The approach is useful for creating models
116 at any meta-level. Our proposal is based on the creation of a set of repositories
117 (label 1 in the Figure), in which heterogeneous data descriptions (OWL ontolo-
118 gies, Ecore meta-models, RDF Schemas, XML schema definitions), and data
119 sources (OWL, RDF data, EMF models, XML documents) are injected.

120 Our system represents this heterogeneous data using a common data model,
121 so that information sources can be stored in the repository in a uniform way.
122 The system provides extensible facilities for the uniform and flexible query of
123 the repository (label 2). We provide basic services for synonym search and word
124 sense analysis, and a predefined catalogue of queries, which can be externally
125 extended. The repository can also store heterogeneous constraints, and we
126 support their evaluation using an extensible facility (label 3). The results of the
127 queries for each source in the repository are aggregated and ranked according
128 to their suitability and relevance. These query results and the information
129 sources themselves can be visualized (label 4). Although the assistance system
130 is independent of any modelling tool, it has been designed to be easy to integrate
131 with a wide range of tools (label 5).

132 We have identified several scenarios where the assistant is useful. They can

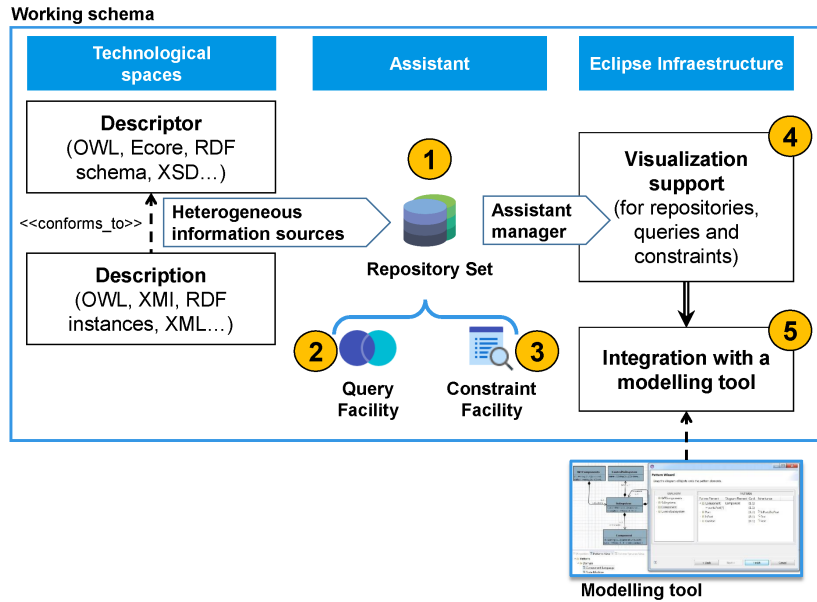


Figure 1: Overview of our approach

133 be generally classified in three areas. Firstly, for creating models and meta-
 134 models. Second, to create artefacts describing a set of other artefacts, like in
 135 model-based product lines [20, 21] or model transformation reuse [22]. Finally,
 136 to evaluate quality aspects of a set of (perhaps heterogeneous) resources. More
 137 in detail, our approach is useful:

- 138 • As support for the development of new domain meta-models. This way,
 139 domain concepts and vocabulary can be sought in external sources such
 140 as XML documents or ontologies.
- 141 • To create models for a particular domain. In this case, model elements
 142 conforming to the same or similar meta-model can be incorporated into the
 143 model being built, and heterogeneous information sources can be queried
 144 in order to extract concrete data values.
- 145 • To design a “concept” meta-model [22]. A concept is a minimal meta-
 146 model that gathers the core primitives within a domain, e.g., for workflow
 147 languages. Furthermore, concepts can be used as the source meta-model
 148 of a model transformation, becoming reusable, so they can be bound to a
 149 particular meta-model. This task implies the querying and understanding
 150 of a variety of meta-models for a particular domain and therefore the
 151 assistant becomes useful.
- 152 • To aggregate multiple existing models into a model-based product line [20,
 153 21]. In this approach, a description of the space of possible features of a

154 software system is created, typically through a feature model. The choice
155 of features implies the selection of a software model variant. This way,
156 there is a need for understanding an existing family of models and to de-
157 scribe their variability. One possible approach is to merge or superimpose
158 all model variants (leading to a so called 150% model) and use “negative
159 variability”, which selectively removes deselected artefacts [20, 21].

- 160 • To detect “bad smells” [23] or signs of bad modelling practices [24] in a set
161 of resources. This is possible, as we can perform queries over a repository
162 to e.g., detect isolated nodes, or find abstract classes with no children.
163 Moreover, our query mechanism is extensible, so that technology-specific
164 or domain-specific queries can be created.

165 While our approach is useful in all these scenarios, to focus the paper, we
166 concentrate on the modelling and meta-modelling scenarios.

167 3. The ingredients of a (meta-)modelling assistant

168 In this section, we detail the three main parts of our approach: (i) the
169 handling of heterogeneous sources through a common data model (Section 3.1),
170 (ii) the uniform support for queries (Section 3.2), and (iii) the managing of
171 constraints (Section 3.3).

172 3.1. Handling heterogeneous sources

173 The first part of our approach addresses to the need for integrating several
174 data sources stored in a variety of formats, providing a mechanism to organize
175 and classify such resources. For this purpose, we rely on a common data model
176 for storing the information of an arbitrary number of meta-levels. Heteroge-
177 neous sources, like XML or Ontologies can then be integrated by establishing
178 transformations into our common data model (cf. Figure 2). As we will see in
179 Section 4, we have designed an extensible, component-based architecture which
180 permits adding support for new sources – with so called format assistants –
181 externally.

182 In detail, each file or information source is represented by a `Resource`, which
183 can be aggregated into `Repository` objects. Each resource contains a collection
184 of `SemanticNode`, i.e., entities that are added to account for different technical
185 spaces [25]. In other words, semantic nodes are elements that gather knowledge
186 from (original) source elements and hence, serve as an abstraction for managing
187 heterogeneous information. `Resources` can be nested to account for hierarchical
188 organization. For example, in `METADEPTH` and `EMF`, models and packages are
189 nested, respectively.

190 `Resources`, nodes and properties are `NamedElements` that are identified by their
191 name and can both act as descriptor, i.e., be a type or class, and be described
192 by other elements, i.e., be instances of other elements. Further, our common
193 data model can accommodate instance-relations found in heterogeneous techni-
194 cal spaces, which include (i) descriptor-only elements, such as, meta-classes in

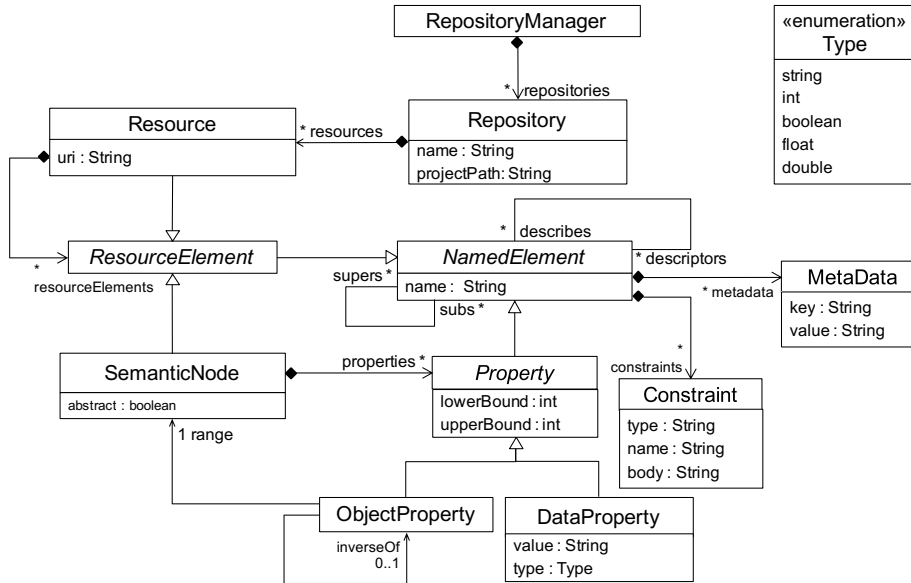


Figure 2: The common data model (package dataModel)

195 meta-models, (ii) described-only elements, such as objects in models, and (iii)
 196 elements that are descriptors of other elements and are described by others sim-
 197 simultaneously, such as clabjects² as applied in multi-level modelling [26]. Hence,
 198 our data model is meta-level agnostic, as we represent with the same concepts
 199 both models and meta-models, classes and objects, and attributes and slots,
 200 leading to simplicity and generality [27]. Moreover, our data model can accom-
 201 modate elements that are described by several elements. Thus, we can accom-
 202 modate non-exclusive class membership of objects, such as found in Ontologies,
 203 and modelling approaches supporting multiple typing, like MetaDepth [28] or
 204 the SMOF OMG standard [29].

205 **NamedElements** can be associated with **MetaData** to account for technology-
 206 specific details that do not fit into our common data model. For example, when
 207 reading an EMF meta-model or a METADEPTH model, it may be necessary to
 208 store whether an object property represents a composition or the potency³ of
 209 an element, respectively. Additionally, a **Resource**, which is also a **NamedElement**,
 210 can manifest conformance relations between artifacts, such as models and meta-
 211 models, or XML documents and XML schema descriptions (XSDs), and thus
 212 permits representing simple mega-models [30].

213 **SemanticNodes** can take part in generalization hierarchies (multiple inheritance

²A clabject is a model element that has both type and instance facets and hence holds both attributes, i.e., field types or classes, and slots, i.e., field values or instances.

³The potency of an element is represented by zero or a positive number that accounts for the instantiation-depth of an element at subsequent meta-levels [26].

214 is supported), and be tagged as *abstract*. Generalization hierarchies can be sup-
 215 ported at any meta-level (i.e., not only at the class level), to account for ap-
 216 proaches where inheritance can occur at the object level [31]. A node is made of
 217 a set of properties (`DataProperty`) and a set of links to other nodes (`ObjectProperty`),
 218 both defining cardinality intervals. Similar to nodes, properties unify the con-
 219 cept of *attribute*, i.e. a specification of required properties in instances, and *slot*,
 220 i.e. a holder for values. The common data model supports a range of neutral
 221 basic data types (`Type` enumeration), such as `string`, `int`, `boolean` and `double`. For
 222 generality, the value of the property is stored as a `String`. Finally, any element
 223 can have `Constraints` attached. The handling of heterogeneous constraints will be
 224 explained in Section 3.3.

225 Table 1 shows how several technologies can be mapped to our data model.
 226 We consider the modelling space (in particular the Eclipse Modelling Framework
 227 (EMF) [32], a widely used implementation of the Meta Object Facility (MOF)
 228 OMG standard [33]), ontologies and XSDs. In all three cases, we show how to
 229 map elements at different meta-levels into our common data model. Section 5.3
 230 will assess the generality of our data model by means of an analytical evaluation.

Common Data Model	EMF	OWL	XSD
Meta-level (types)			
<i>Resource</i>	Ecore file/EPackage	OWL file	XSD file
<i>SemanticNode</i>	EClass	OWL Class	xs:element
<i>Property (abstract)</i>	EStructuralFeature	rdfs:domain	xs:complexType xs:element
<i>ObjectProperty</i>	EReference	owl:ObjectProperty	Nested xs:element IDREF attribute
<i>DataProperty</i>	EAttribute	owl:DatatypeProperty	xs:attribute
<i>Property.supers</i>	EClass.eSuperTypes	Inverse of rdfs:subClassOf	xs:element type attribute
<i>Constraint</i>	OCL EAnnotation	N/A	xs:restriction
Model/Data level (instances)			
<i>Resource</i>	XMI file	OWL file	XML file
<i>SemanticNode</i>	EObject	Individual	XML element
<i>ObjectProperty</i>	Java reference	owl:ObjectProperty	Nested xs:element IDREF attribute
<i>DataProperty</i>	Java attribute	owl:DatatypeProperty	XML attribute

Table 1: Mapping different representation technologies to the common data model

231 EMF supports two meta-levels, and the mapping is direct. Figure 3 shows
 232 a schema of how the translation from EMF into our data model is performed.
 233 The figure shows on the top that both meta-models (called *ecore* models) and
 234 models (typically serialized in XMI format) are transformed into `Resources`. In
 235 this case, the `Resource` object from the model is described by the `Resource` of the
 236 meta-model. The elements within both meta-models and models follow this
 237 translation scheme as well. Both `EClasses` and `EObjects` are transformed into
 238 `SemanticNodes` with a suitable descriptor relation, and similar for references and

239 attributes. At the model level (in the compiled mode of EMF) links and slots
 240 are represented as Java fields, whose value is obtained via getter methods.

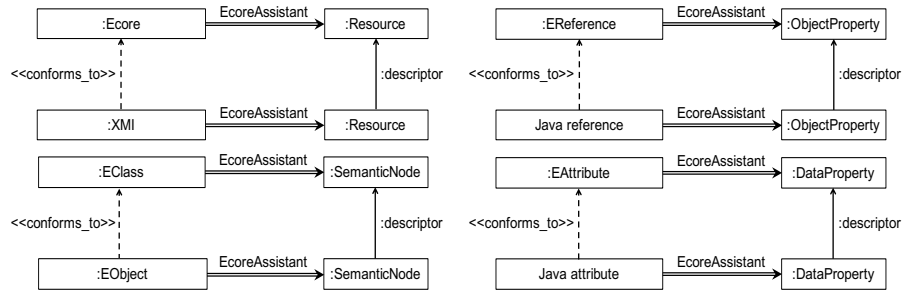


Figure 3: Injecting EMF (meta-)models into the common data model

241 Figure 4 depicts the translation of XSDs, i.e., acting as language definitions
 242 and XML documents. This case is conceptually similar to EMF. The figure
 243 shows on the top that a schema, typically serialized in XSD format, is trans-
 244 formed into a Resource of our common data model. Then, as a result of XSDs
 245 being described in the XML format, the Resource object from the document is
 246 described by the Resource of the schema.

247 The elements within the schema and the document follow this translation as
 248 well. For example, an XML element is transformed into a SemanticNode with a
 249 descriptor relation to an xs:element. Moreover, an XML element or XML attribute
 250 is transformed either into an ObjectProperty or a DataProperty depending on the
 251 type it specifies. In particular, in case of xs:IDREF an ObjectProperty is created,
 252 while a DataProperty is created for any other type.

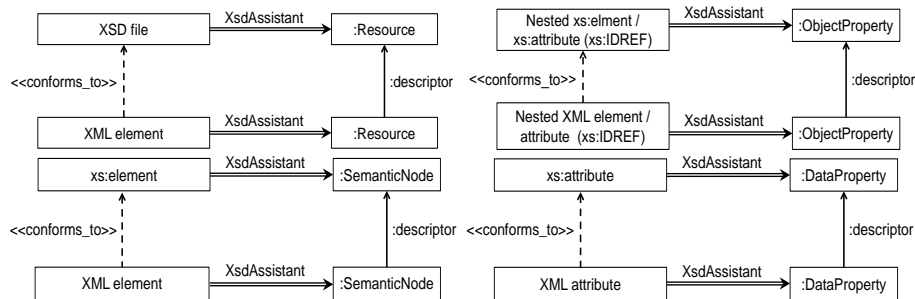


Figure 4: Injecting XML schema descriptions into the common data model

253 Finally, in the case of Ontologies there are no explicit meta-levels (Figure 5).
 254 Then, classes may have several descriptors, and individuals (instances of a class)
 255 can have several classifiers, covering the different levels in the representation
 256 of concepts. Thus, SemanticNodes and Properties can take part in generalization
 257 hierarchies, all of them represented by the taxonomy.

258 The technical realization of new format assistants will be explained in Sec-

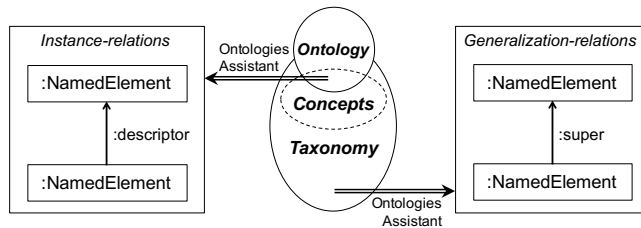


Figure 5: Injecting ontologies into the common data model

tion 4, while Section 5.3 will evaluate the generality of the data model.

3.2. Querying

Once the heterogeneous information is transformed into a common representation, it can be queried in a uniform way. As information conforming to many different schemas may be available in the data model, the query mechanism provided needs to support the flexible exploration of the information gathered. Moreover, oftentimes, the developer may only have a vague idea on what to search, hence a mechanism for inexact matching [34] is needed. This mechanism should be able to provide the user with not only exact matches, but with other related elements as well.

Thus, we refrained from using directly standard model query languages, like the Object Constraint Language (OCL) [35], because they would assume a precise, unique domain meta-model (while may need to query several models conformant to different domain meta-models), and rely on exact matches of queries. Moreover, queries would need to be re-expressed using the meta-model in Figure 2, instead of in terms of the domain meta-models, which may lead to cumbersome expressions. Finally, we need our query mechanism to work at any meta-level.

Figure 6 shows the meta-model describing our extensible query approach. The meta-model provides a mechanism for configuring queries (by specifying the necessary query inputs, class `SearchParam`), while the actual query results are reified using objects of type `SearchResult`. The results can be aggregated in groups (through class `GroupedSearchResult`), or returned atomically (class `AtomicSearchResult`).

Our approach supports two types of queries: atomic (`SimpleSearchConfiguration`) and composite (`CompositeSearchConfiguration`). Atomic queries are configured by declaring a set of `SearchParams` (representing the expected input parameters from the user), specifying the type of element it searches (`filterBy` attribute). Composite queries are formed by combining other (atomic or composite) queries, through the `and`, or `and not` logical connectives. Composite queries are issued by combining the result sets of previous (simple or composite) queries (reference `CompositeSearchConfiguration.inputSearchResults`), and may be nested (composition reference `CompositeSearchConfiguration.children`).

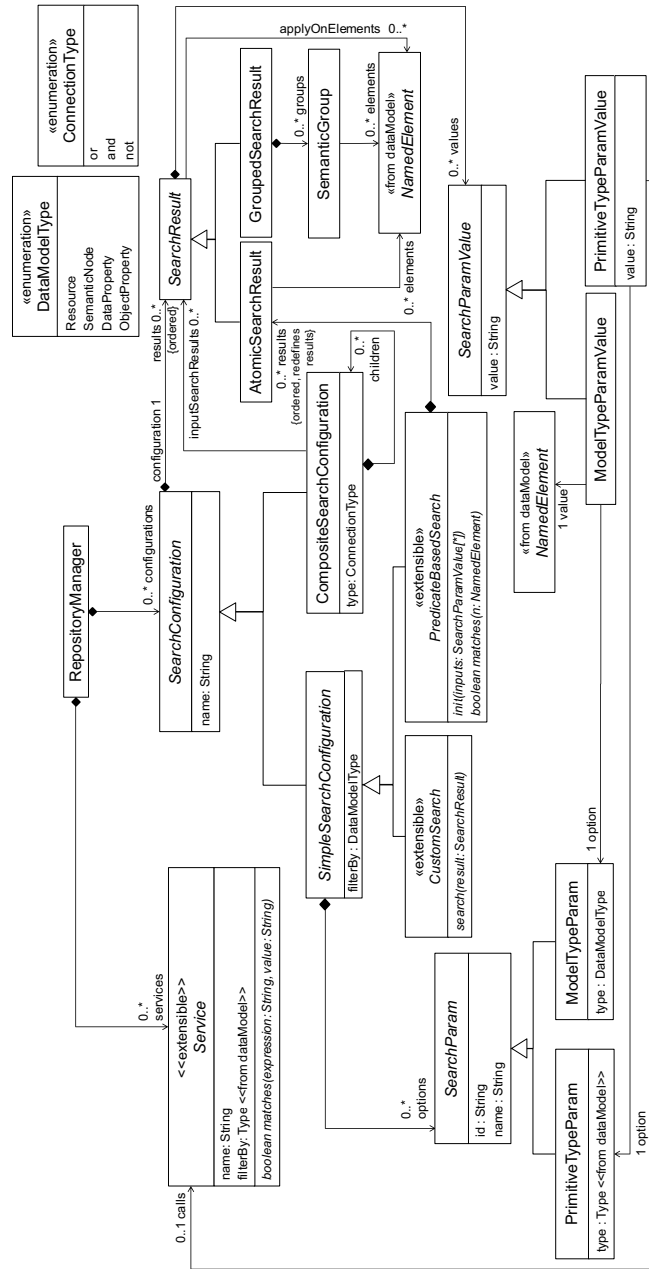


Figure 6: Meta-model of our extensible query mechanism (package queries).

292 Atomic queries can follow two styles: predicate-based (class `PredicateBased`
 293 `Search`) or custom (class `CustomSearch`). In both cases, user-defined queries are
 294 expected to extend these two abstract classes. In practice, as we will show in
 295 Section 4, this is realized by profiting from Eclipse extension points. Their

296 difference relies on how the iteration is performed: internally (driven by the
297 engine), or externally (driven by the user code); and on the possibility of group-
298 ing the results. In practice, predicate-based queries are easier to specify, while
299 custom queries permit more control on how outputs are presented.

300 Predicate-based queries select their result by providing a boolean predicate,
301 evaluated on `NamedElements`. Their result is atomic (i.e., not divided in groups),
302 made of all `NamedElement` objects satisfying the predicate. This kind of queries
303 feature internal iteration. This way, our engine is responsible to traverse the
304 repository and iteratively pass each object (of the type expected by `filterBy`) to
305 the `matches` method. Alternatively, `CustomSearch` queries are responsible to both
306 traverse the elements in the repository (external iteration) and select those
307 elements fulfilling the query. This is a more flexible way of querying, which may
308 construct results aggregated in groups.

309 When a custom query is to be executed, the `search` method receives a `SearchRe-`
310 `sult` object, which initially contains a set of `SearchParamValue` objects with the input
311 parameter values (as entered by the user), and optionally a set of elements over
312 which the query is to be executed (`applyOnElements` collection). If `applyOnElements`
313 is empty, then the query is assumed to search through the whole repository. The
314 input parameters of a search can either be of primitive data type (class `Primi-`
315 `tiveTypeParamValue`), or model elements of a given kind like `Resource`, `SemanticNode`,
316 `DataProperty` or `ObjectProperty` (class `ModelTypeParamValue`). After the execution of
317 the `search` method the `SearchResult` must point to the selected element. In the
318 grouped elements output case (class `GroupedSearchResult`), the `search` method im-
319 plementation must decide how to split the members from the `elements` collection
320 by the definition of `SemanticGroups`.

321 Please note that different invocations to the `search` method results in different
322 `SearchResult` objects, placed in the `results` ordered collection. This enables having a
323 history of searches performed, which can be useful in explaining the provenance
324 of the different elements in a model being built.

325 In a predicate-based query, we rely on a `matches` method that must evaluate
326 if the list of elements from the input collection belongs to the output (`elements`
327 collection) or not. In this case, results are saved as `AtomicSearchResults`. Before
328 the iteration starts, the `init` method is invoked, receiving a collection of input
329 values (objects of type `SearchParamValue`), corresponding to the query parameter
330 values input by the user.

331 3.2.1. Query services

332 As seen in the meta-model of Figure 6, queries can make use of `Services`
333 through an extensible mechanism (extending meta-class `Service`). Services are
334 functions for certain data types from our data model, such as `strings` or `integers`.
335 Therefore, services can be used by a particular query on `SearchParamValues` of a
336 given type. Next we describe some of the most prominent services we provide.

337 **Inexact Matching** Searches that rely on exact string matching are likely to
338 provide poor results, as entities can be named using compound names

339 (“ProcessModel”), or using different derivations (e.g., “processing”, “pro-
340 cessed”). This way, we provide a means for *inexact matching* [34] to in-
341 crease the possibilities of finding useful information. The service is based
342 on two techniques: (i) detection of compound names, and (ii) comparing
343 words using their roots. This service is available on search inputs of type
344 String (i.e., when the corresponding `PrimitiveTypeParam` is typed as `String`).

345 Regarding the first technique, it is quite common to find entities in meta-
346 models or ontologies whose name is the concatenation of two or more
347 words, most often in camel case. Our service considers the complete word,
348 and also its different parts in the comparison. Regarding the second tech-
349 nique, comparing word pairs (e.g., “process”, “procedure”) might throw
350 unsatisfying, or too general, results, even if they belong to the same word
351 sense. For this reason, the service uses the Porter stemmer algorithm [36],
352 a method to reduce the comparison of two words to their lexical roots.

353 **Word synonym expansion and ranking** The exploration of a domain is a
354 difficult task for two reasons: (i) in a new domain, with no experience
355 background, a developer may lack the knowledge base about the vocabu-
356 lary that defines the domain and (ii) in a known domain, a developer
357 can lose sight of the main parts of the domain and, as a consequence,
358 build a poor meta-model. Thus, it might be useful to increase the name-
359 based searches to consider synonyms relevant for the domain. However,
360 words have synonyms with respect to different senses and therefore better
361 search results are obtained by ruling out unrelated senses. For example,
362 the word “glass” has different senses, e.g., to refer to “tumbler or drinking
363 container” and to “lens to aid in vision”.

364 Thus, we offer a service that, given a set of terms, expand them creating a
365 list of synonyms, ranking them according to the relevance for the domain
366 of the input terms. The service is inspired by the Lesk algorithm [37]
367 and evaluates each term in a sentence assuming that every term tends to
368 share a common semantic field with its siblings. For that purpose, we
369 use Wordnet (a lexical database for the English language) [38] to select,
370 among the available entities, the most suitable candidates to match the
371 common semantic fields from an input term list. The list of candidates is
372 obtained by the use of a rule-based system. In the system, we assign points
373 to the whole list of senses provided by Wordnet while they are evaluated
374 to discover whether they fit to the target domain or not.

375 The points are given by an strategy depending on which rule is more
376 important during the evaluation. With the service we provide a set of
377 strategies but new strategies can be added as well. Next, we present the
378 rules for assigning those points and the strategies defined:

379 We consider the following entities to calculate the ranking:

- 380 • U is the universal set of all possible terms, or words.

- 381 • $T = \{t_1, \dots, t_n\} \subseteq U$ is a set of input terms, or words, e.g. “process”,
 382 “activity”, “task”.
 383 • $S = \{s_1, \dots, s_m\}$ is a set of senses, for example the term “process”
 384 can refer to “set of actions” (e.g., s_1) or “a series of changes in the
 385 body” (e.g., s_2).
 386 • We consider phrases (denoted by p_i), made of sequences of terms.
 387 We write $t \in p_i$ to denote that term t occurs in the phrase p_i .

388 We assume the following linguistic functions and predicates, which can be
 389 provided by systems like Wordnet:

- 390 • Given a term $t \in U$, function $sense(t)$ returns the different senses of
 391 t . For example $sense(process) = \{s_1, s_2\}$.
 392 • Given a sense $s \in S$, function $syn(s)$ returns a set of terms, which
 393 are synonyms with respect to the sense s .
 394 • Given a term $t \in U$, function $defs(t)$ returns a set of definitions (a
 395 set of sentences) of term t .
 396 • Given a term $t \in U$, function $exs(t) = \{p_1, \dots, p_m \mid t \in p_i\}$ returns a
 397 set of example sentences, where each p_i contains the term t .
 398 • Given two terms, predicate $syn(t, t')$ holds if both terms are syn-
 399 onyms, independently of the sense.
 400 • Given two terms $deriv(t, t')$ holds if the terms have a linguistic deriva-
 401 tive path.

402 Given a set of terms T , our system returns a set of terms $O = T \cup_{t \in T} \{u \in$
 403 $U \mid syn(t, u)\}$, made of the input terms plus their synonyms where the
 404 words in O are ranked according to a number of points, given by the
 405 following rules:

R₁: Synonyms of input terms

$$\forall t, t' \in T \bullet syn(t, t') \implies points(t') := points(t) + p_1$$

R₂: Synonyms of sense synonyms

$$\forall t \in T, \forall s \in sense(t), \forall g \in syn(s) \bullet \\ syn(t, g) \implies points(g) := points(s) + p_2$$

R₃: Linguistic derivatives of sense synonyms

$$\forall t \in T, \forall s \in sense(t), \forall g \in syn(s) \bullet \\ deriv(t, g) \implies points(g) := points(s) + p_3$$

R₄: Synonyms in definitions

$$\forall t \in T, \forall s \in sense(t), \forall g \in syn(s), \forall p \in defs(t) \bullet \\ g \in p \implies points(g) := points(p) + p_4$$

R₅: Synonyms in examples

$$\forall t \in T, \forall s \in \text{sense}(t), \forall g \in \text{syn}(s), \forall p \in \text{exs}(t) \bullet \\ g \in p \implies \text{points}(g) := \text{points}(g) + p_5$$

406 Several strategies can be used to assign points to every rule:

- 407 • All the same: Each **R_i** receives the same quantity of points (p_i).
- 408 • Synonyms first: **R₁** and **R₂** receive more points than the rest of rules.
- 409 • Definitions first: **R₄** receives more points than the rest of rules.
- 410 • Examples first: **R₅** receives more points than the rest of rules.
- 411 • Custom: A distribution based on a custom criteria.

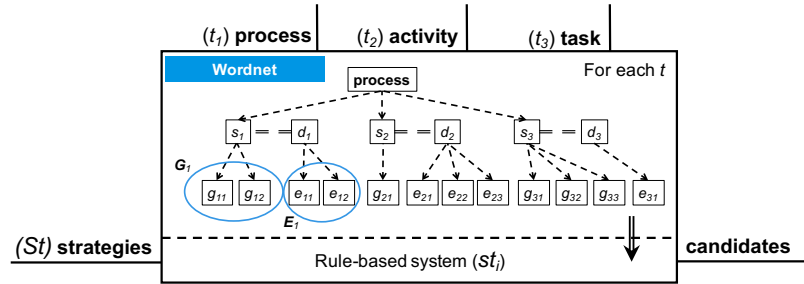


Figure 7: Example execution of the synonym expansion and ranking service

412 *Example.* Figure 7 shows an execution schema for the service. Firstly, a
 413 set T of terms is received by the service. For each term (t_i) , a tree of
 414 senses (s_{ij}), definitions (d_i), examples (e_{ij}) and synonyms (g_{ij}) is formed,
 415 using information from Wordnet. The set of rules are applied for each
 416 term according to a selected strategy (st_i). For example, for the list of
 417 terms $T = \text{process}, \text{activity}, \text{task}$ for a custom strategy that assigns 1000
 418 points to the value p_1 , 80 to p_2 , 20 to p_3 , 100 to p_4 and 20 to p_5 , the
 419 following output list of ranked candidates is obtained:

420	1000 task	429	240 procedure
421	1000 process	430	200 treat
422	1000 activity	431	200 physical process
423	320 job	432	200 outgrowth
424	320 chore	433	200 appendage
425	300 summons	434	180 natural process
426	260 body process	435	180 natural action
427	260 bodily process	436	100 work on
428	260 bodily function	437	100 work

438	100 unconscious process	446	0 project
439	100 swear out	447	0 operation
440	100 sue	448	0 mental process
441	100 serve	449	0 labor
442	100 march	450	0 cognitive process
443	100 litigate	451	0 cognitive operation
444	100 action	452	0 activeness
445	0 undertaking		

453 where the service discards the words with no points.

454 **Numeric intervals** For numeric properties, instead of writing concrete values,
 455 it is possible to enter intervals (e.g., [1..5] or [2..*]) to achieve more flexible
 456 queries.

457 *3.2.2. Examples*

458 In this section we present two examples, illustrating the query extensibility
 459 mechanisms and the services provided. The list of all available queries is shown
 460 in Table 9 in the appendix.

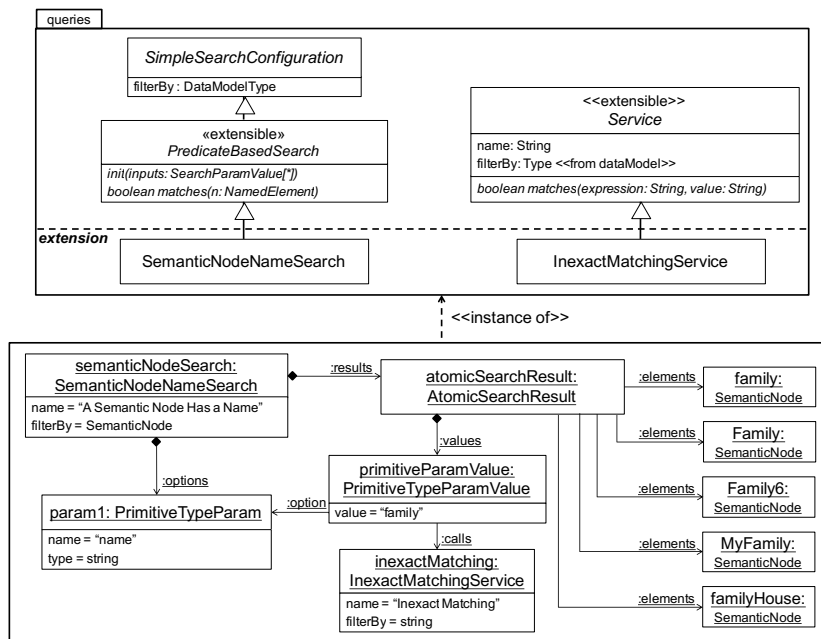


Figure 8: An instance of a predicate-based search (PredicateBasedSearch) with an atomic result (AtomicSearchResult)

461 Figure 8 depicts a predicate-based search that gathers the results atomically.
 462 The upper part of the figure represents an excerpt of the `queries` package shown
 463 in Figure 6, which is extended with a query (`SemanticNodeNameSearch`) and the
 464 “Inexact Matching” service explained in Section 3.2.1.

465 The lower part of the figure shows an instance model containing the pa-
 466 rameter of the search (object `PrimitiveTypeParam`), its input value (object `Primi-`
 467 `tiveTypeParamValue`) and the search results (semantic nodes referenced from object
 468 `searchResult`). Overall, the input value “family” is received by the search, which
 469 is then passed to the service. The service checks whether the attribute value
 470 of the semantic node matches with the expression or not. Finally, the set of
 471 selected semantic nodes are attached to the `AtomicSearchResult` object.

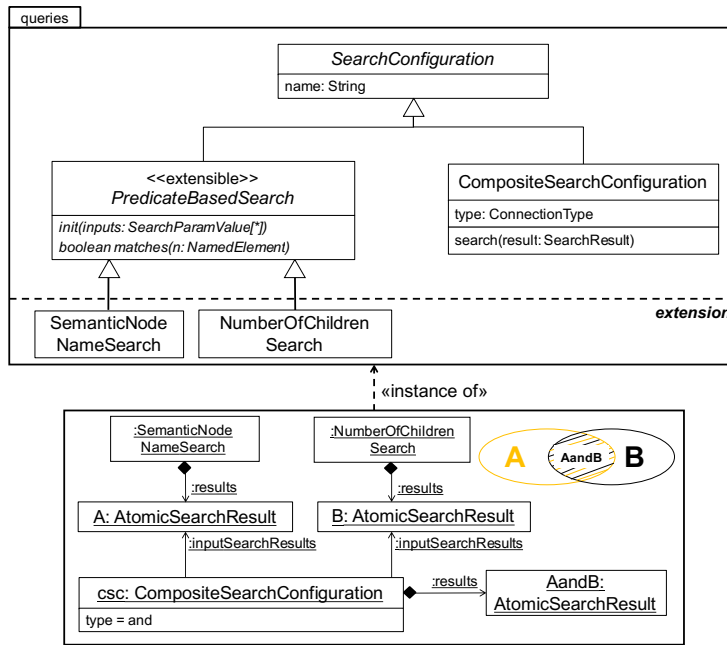


Figure 9: An and-type composition (`CompositeSearchConfiguration`) of two searches

472 Next, Figure 9 shows a composite search made of the conjunction of the
 473 search result of two queries: the `SemanticNodeNameSearch` query and `NumberofChildrenSearch`
 474 (`NumberOfChildrenSearch` (which searches for nodes with more than a number of children through
 475 a generalization hierarchy, cf. Table 9)). The composite query returns `Semantic-`
 476 `Nodes` belonging to both search results, as schematically depicted by the Venn
 477 diagram in the top-right corner of the lower model.

478 3.3. Handling constraints

479 In our data model, `SemanticNodes` are used to store the knowledge found in an
 480 element, including properties and links to other nodes. Additionally, they may
 481 include restrictions to be satisfied by their instances.

482 For example, in meta-models, a class may contain OCL invariants that all ob-
 483 jects of such class should obey. Other modelling technologies, like METADEPTH
 484 allow attaching constraints to whole models [12]. Similarly, elements in XML
 485 schemas may contain restrictions on properties of a given data type. These may
 486 be used to define patterns on strings (regular expressions), restrictions on the
 487 string length (min/max/exact length), and handling of white spaces, among
 488 others. While OWL does not directly support integrity constraints, extensions
 489 have been proposed for this purpose [39, 40].

490 Thus, our data model includes a facility to store and interpret heterogeneous
 491 constraints. Constraints can be attached to any NamedElement, which includes
 492 Resources, SemanticNodes and Properties, covering the main scenarios in the different
 493 technical spaces. Constraints have a type identifying the kind of constraint (e.g.,
 494 OCL), a name, and a body.

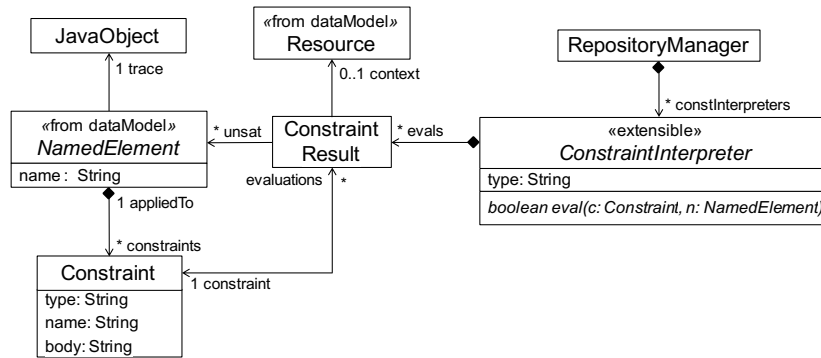


Figure 10: Meta-model of the extensible constraint handling mechanism.

495 In order to support evaluation of heterogeneous constraints (OCL, XML
 496 schema restrictions, etc) our approach is extensible. This way, constraint inter-
 497 preters can be incorporated by extending class ConstraintInterpreter, declaring the
 498 constraint types they can handle, and implementing the evaluate method. The
 499 method receives a constraint and an instance of the element the constraint is
 500 attached to, and returns a boolean indicating if the element satisfies the con-
 501 straint. As we will see in Section 4, the addition of constraint interpreters is
 502 done through an Eclipse extension point. Similar to query results, constraint
 503 results are reified using ConstraintResult objects, which hold elements that do not
 504 satisfy the constraint, and in addition organizes results in the context of the
 505 enclosing Resource.

506 4. Architecture and tool support

507 We have realized the previously presented concepts in a tool called EX-
 508 TREMO. It has been implemented as an Eclipse plugin, is open source, and is
 509 freely available at <http://miso.es/tools/extremo.html>. The web page in-

510 cludes videos, screenshots and installation details. A schema of EXTREMO's
 511 architecture is shown in Figure 11.

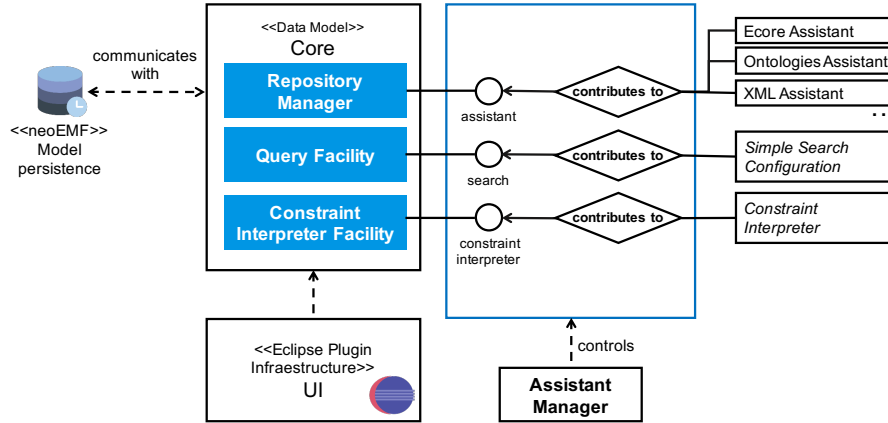


Figure 11: Architecture of EXTREMO

512 EXTREMO is made of a **Core** component, which provides support for the
 513 common data model and includes subcomponents for the query and constraint
 514 handling facilities. The **Core** component can be extended in different ways, e.g.,
 515 to provide access to heterogeneous information sources, as shown in Figure 11.
 516 This extensibility is realized through Eclipse extension points. These are inter-
 517 faces that can be implemented by external components to provide customized
 518 functionality.

519 To allow for scalability, the repository is persisted using NeoEMF [41],
 520 a model persistence solution designed to store models in NoSQL datastores.
 521 NeoEMF is based on a lazy-loading mechanism that transparently brings into
 522 memory model elements only when they are accessed, and removes them when
 523 they are no longer needed.

524 The **UI** component permits visualization and interaction with the resources
 525 and query results. A set of views, commands, wizards and actions has been
 526 defined to control the access to the repository, the list of query results and the
 527 constraint validation facilities. By extending this component, it is possible to
 528 integrate EXTREMO with external modelling tools.

529 Next, we will detail the structure of the **Core** subsystem in Section 4.1, while
 530 the **UI** subsystem will be described in Section 4.2.

531 4.1. The Core subsystem

532 Figure 12 shows the main components of the **Core** subsystem: (i) a **Repository**
 533 **Manager**, which controls the access to the common data model and assists in the
 534 process of incorporating a new technological space; (ii) a **Query Facility**, which
 535 supports our query mechanisms in an extensible way; and (iii) a **Constraint Inter-**
 536 **preter Facility**, which provides flexible support for different constraint formats and
 537 interpreters.

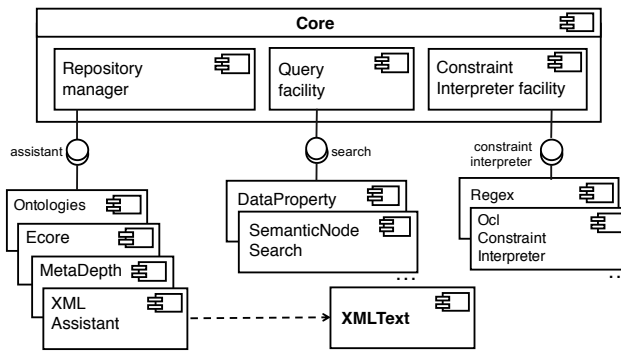


Figure 12: Architecture of the Core component

538 For each part, a set of extension points have been defined, and the figure
 539 shows some example implementations of these. The first extension point (*assis-*
 540 *stant*) permits adding support for new data formats. It requires implementing a
 541 mapping from the format-specific structure, such as XML, to the common data
 542 model, as described in Section 3.1.

543 We predefined a set of assistants as well as a framework for their creation,
 544 which permits their conceptual organization as model-to-model transformations.
 545 Hence, we provide an abstract class with a number of methods, which need to
 546 be overridden for the particular assistant, and act as rules in a transformation.
 547 In these methods, it is possible to create one or more elements in the common
 548 data model, hence supporting one-to-many and many-to-one mappings.

549 To facilitate the construction of new assistants, it is possible to define class
 550 hierarchies, and hence reuse import functionality. In addition, our scheme can
 551 profit from existing transformations between two specific technical spaces. For
 552 example, if one had a transformation from XSD to Ecore, and an assistant for
 553 Ecore (translating Ecore to EXTREMO's common data model), then an assistant
 554 for XSD can be built by first translating into Ecore and then invoking the Ecore
 555 assistant. This is the way we implemented the *XsdAssistant* (see Figure 4) [42, 43,
 556 44].

557 The second and third extension points provide extensibility for queries. Con-
 558 ceptually, user defined queries extend the meta-model of Figure 6. The exten-
 559 sions allow defining custom, predicate-based, and composite queries by subclass-
 560 ing from the corresponding classes in Figure 6. Finally, the last extension point
 561 permits contributing support for evaluating new types of constraints, extending
 562 the meta-model in Figure 10.

563 4.2. The UI subsystem

564 Figure 13 shows the architecture of the UI subsystem. It is made of contribu-
 565 tions to the Eclipse infrastructure to visualize and interact with the repository,
 566 to visualize the search results and the list of elements that satisfy a constraint.

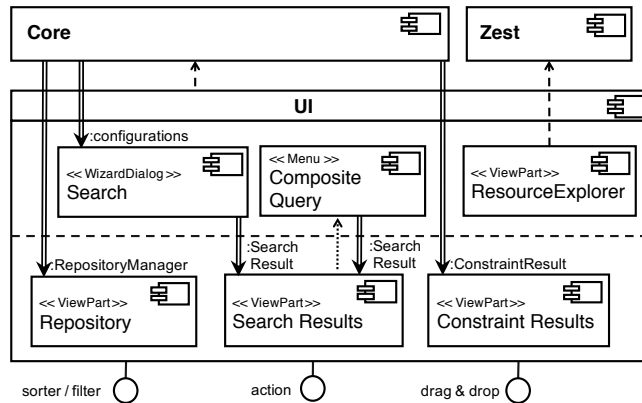


Figure 13: Architecture of the UI component

567 The UI subsystem is composed of: (i) a Search wizard dialog, that receives
 568 the list of search configurations; (ii) a Resource Explorer, a mechanism based
 569 on Zest (see <https://www.eclipse.org/gef/zest/>), a component for graph-
 570 based visualization, which provides support for filtering and layouts; and (iii) a
 571 set of view parts, that reflect the current state of the repository set model, the
 572 query results (as instances of the class `SearchResult`) and the constraints validation.
 573 All of them can be sorted or filtered by means of an extension point.

574 As an example, Figure 14 shows EXTREMO in action. In particular, it shows
 575 the query dialog by which any of the defined queries can be selected, input
 576 values can be given to each of its input parameters, and services can be selected
 577 depending on the types of the parameters. In the lower part, the figure shows
 578 the repository view, with some resources and their content; and the search result
 579 view, which permits browsing through the query results. It must be noted that
 580 semantic nodes in the repository view have distinct icons, depending on whether
 581 they contain data, object properties or constraints, and on whether they are
 582 types, instances or both.

583 Figure 15 shows the resource explorer. In particular, it shows on the right an
 584 instance of our common data model and the relationships between nodes. Since
 585 the resource explorer is based on a component for graph-based visualization, the
 586 `SemanticNode` instances are represented as nodes and the `ObjectProperty` instances
 587 are represented as edges of the graph. The left part shows how the resource
 588 explorer can be invoked from every resource with a action contribution to the
 589 pop-up contextual menu of the repository view.

590 The UI subsystem has been designed so that it can be flexibly integrated with
 591 external modelling tools. For that purpose, two extension points have been de-
 592 fined. The first one enables the addition of an action to the view parts, as a
 593 reference in the contextual menu and the toolbar. The other extension point
 594 enables the drag operation from the views and the dropping of the selected in-
 595 formation into any graphical editor based on Graphical Editing Framework [45].

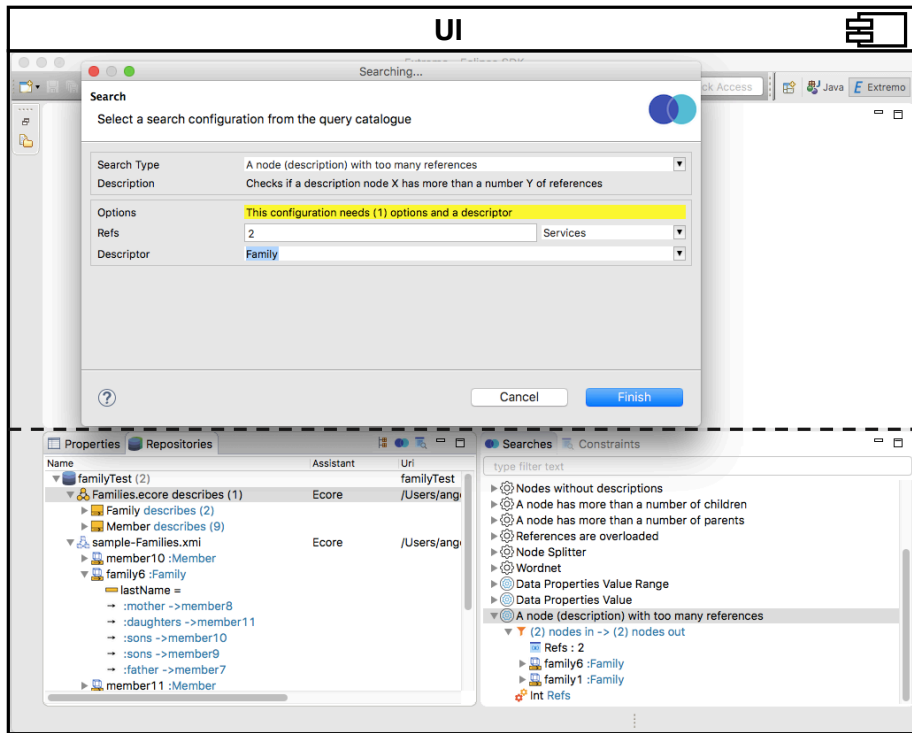


Figure 14: EXTREMO in Action: Search Wizard Dialog and views

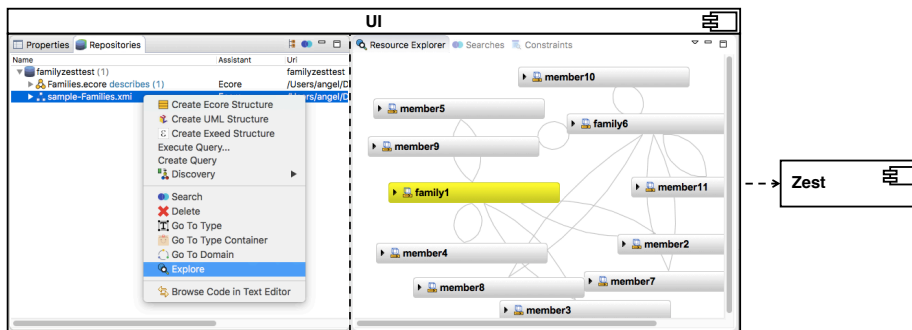


Figure 15: EXTREMO in Action: Resource Explorer

596 This is the underlying framework of Eclipse graphical editors. We will explain
 597 these two integration styles in the next two sections.

598 4.2.1. Integration with a modelling tool by means of actions

599 In this section we present the integration of the core components of EX-
 600 TREMO with a modelling tool using an action contribution to the view parts of

601 the UI subsystem. We will illustrate this section with the UML model editor⁴,
 602 a solution based on a `TreeEditor`. This kind of editor is an extensible part of the
 603 Eclipse infraestructure.

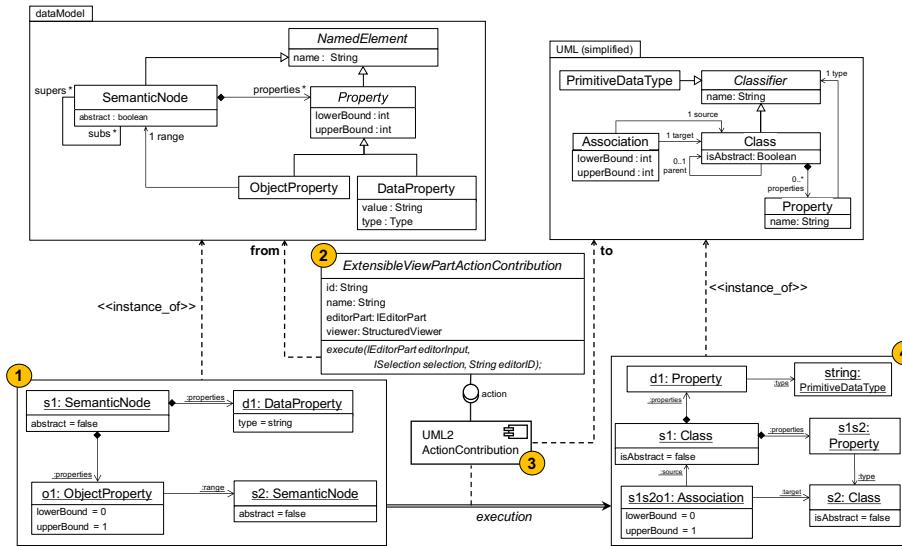


Figure 16: Example: action-based integration

604 Figure 16 illustrates the working scheme of this integration style, where an
 605 instance of our data model (Figure 2) selected by the user (e.g., via a query) is
 606 incorporated into a UML model [4]. The elements selected by the user (label 1)
 607 are two non-abstract `SemanticNode` objects. The first one contains a `DataProperty`
 608 (`d1`, typed as `string`) and an `ObjectProperty` that refers to the second `SemanticNode`.
 609 The addition of a new action contribution is enabled by an extension point.
 610 The action contribution provides the mapping between our common data model
 611 and the intrinsic model of the `TreeEditor`. The action contribution must extend
 612 the `ExtensableViewPartActionContribution` class (label 2) and implement an abstract
 613 method that creates the instances of the target model, conceptually working as
 614 a model-to-model transformation (label 3). Finally, a UML model is created
 615 (label 4). The initial two `SemanticNode` objects are mapped into two non-abstract
 616 `Class` objects, the initial `DataProperty` object is mapped into a `Property` object
 617 and the initial `ObjectProperty` object is mapped into an `Association` object.

618 Figure 17 shows how this is achieved in practice. The repository view of
 619 EXTREMO is shown in label 1. In this view, the user can select a number of
 620 elements that are to be incorporated into the UML model (editor shown in label
 621 3). For this purpose, a contextual menu offers the different action contributions
 622 (label 2). The UML editor in the figure shows already the Ecore elements

⁴UML2-MDT, www.eclipse.org/modeling/mdt

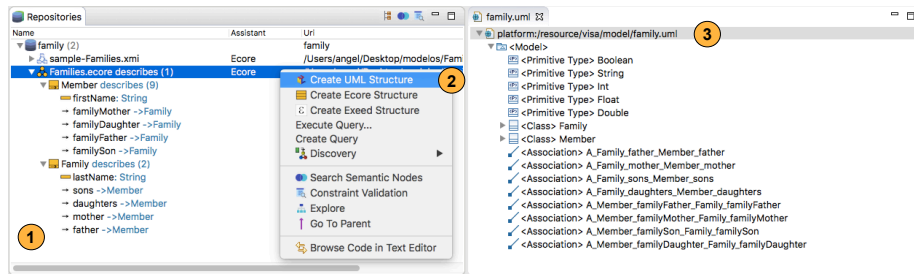


Figure 17: Integrating EXTREMO with the UML2 modelling tool using an action contribution

623 incorporated into the UML model.

624 *4.2.2. Integration with a modelling tool by means of drag and drop*

625 The second integration style enables the addition of a drag operation from
 626 the views and the dropping of a set of NamedElements into a graphical editor
 627 based on Graphical Editing Framework [45], the underlying framework of Eclipse
 628 graphical editors.

629 We will illustrate this section with DSL-tao [19], an Eclipse plugin for the
 630 construction of DSLs using a pattern-based approach. The underlying representation
 631 of DSL-tao is Ecore. Thus, in this case, Figure 18 shows an example that
 632 transfers an instance of our data model (Figure 2) into an instance of Ecore.

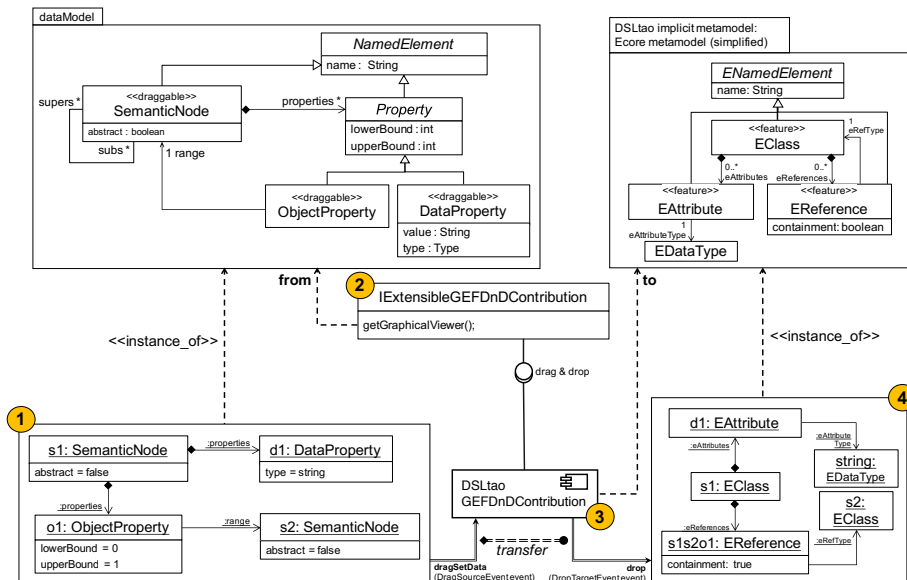


Figure 18: Example: drag and drop based integration

633 As in the previous section, the portion of the model selected by the user

634 (label 1) contains two non-abstract `SemanticNode` objects. The first one contains
 635 a `DataProperty` (`d1`, typed as `string`) and an `ObjectProperty` that refers to the second
 636 `SemanticNode`. The addition of a new dropping contribution is enabled by an
 637 extension point. The dropping contribution provides the transfer of a set of
 638 `NamedElements` selected from the views and the catching of the drag and drop
 639 event. The dropping contribution must extend the `IExtensibleGEFDnDContribution`
 640 interface (label 2) and implement a method resolving the graphical viewer editor
 641 that receives the selected elements (label 3). Finally, an `Ecore` model is created
 642 (label 4). The initial two `SemanticNode` objects are mapped into two `EClass` objects,
 643 the initial `DataProperty` object is mapped into an `EAttribute` object and the initial
 644 `ObjectProperty` object is mapped into an `EReference` object.

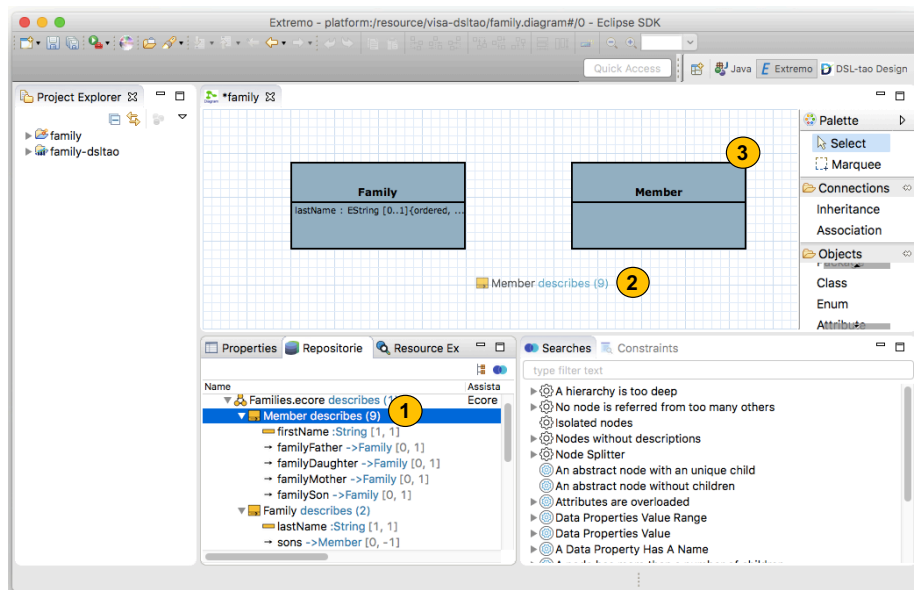


Figure 19: Integrating EXTREMO with DSL-tao using a drag and drop contribution

645 Figure 19 shows how this is achieved in practice. The Figure shows the main
 646 canvas of DSL-tao, which permits building graphically a meta-model using the
 647 palette to the right. The bottom of the figure shows the views contributed by
 648 EXTREMO. In this case, the user may initiate a drag from some model element
 649 in the view and drop it into the canvas (labels 2 and 3). In the Figure, the user
 650 has incorporated class `Member` into the meta-model, while it is also possible to
 651 incorporate attributes and references.

652 5. Evaluation

653 We present an evaluation of our approach under different perspectives. We
 654 start by describing the research questions we address in the evaluation in Sec-

655 tion 5.1, followed by three different evaluations in Sections 5.2, 5.3 and 5.4, and
656 finishing with a discussion of threats to validity in Section 5.5.

657 5.1. Research Questions

658 By conducting this evaluation, we aim at solving the following research ques-
659 tions:

660 *RQ1: How useful is EXTREMO to solve practical problems in Language Engi-
661 neering?*

662 In order to assess the usefulness of EXTREMO, we will use two demonstration
663 cases [46]. In both of them, EXTREMO was used to support the development of
664 complete DSL implementations for process modelling in the immigration domain
665 (Section 5.2.1) and modelling of production systems (Section 5.2.2).

666 *RQ2: How capable is EXTREMO to represent information coming from different
667 technological spaces?*

668 As previously described, the range of formats currently supported by EXTREMO
669 can be extended. This way, Section 5.3 presents an analytical evaluation to
670 assess how well EXTREMO’s data model covers the space of possible information
671 description approaches.

672 *RQ3: How integrable is EXTREMO?*

673 One of the salient features in EXTREMO is the possibility of integrating it with
674 third-party tools. Having already successfully integrated a number of tools
675 with EXTREMO, we intend to assess the reach and limitations of the integration
676 mechanism. This question is approached in Section 5.4.

677 5.2. Evaluating usefulness

678 In this section, we evaluate usefulness by presenting two demonstration cases.
679 The first one is on process modelling in the immigration domain (Section 5.2.1),
680 while the second one is on modelling of production systems (Section 5.2.2).
681 In the cases, EXTREMO was used in combination with two different modelling
682 tools (DSL-tao and the Ecore tree editor), while assistants for four different
683 technologies were used (Ecore, Ontology, CSV and XML). The purpose of the
684 cases is evaluating the power of combining heterogeneous information sources
685 for language engineering.

686 Characteristics of the two selected cases are (i) they cover both structural
687 and behavioral modelling languages, (ii) they combine information from stan-
688 dardized modelling languages and from very focused domain-specific languages,
689 and (iii) modelling languages are integrated with small to large domain descrip-
690 tions coming from different technological spaces. They differ in the variety of
691 technical spaces and the size of resources being considered.

692 For both cases we will highlight the use of our framework over a sequence of
693 phases, which include (i) the scope of the language and an example of model,
694 (ii) the collection of resources required, (iii) the resource import and querying
695 of the common data model, (iv) the construction of a language meta-model and
696 creation of instances, and (v) the results obtained.

697 *5.2.1. Immigration process modelling*

698 In the first demonstration case, we present the construction of a DSL for
699 describing administrative e-Government processes for immigration.

700 **Scope of the Language.** We will construct a language for the modelling of
701 processes to issue visas according to the American regulation⁵, called
702 IWML. The language will include elements showing the variety of offices,
703 concepts from the domain of immigration and agents involved in the pro-
704 cess and generic elements of workflow languages like BPMN [47].

705 **Example Model.** Figure 20 shows an example IWML model, which contains
706 (i) domain-specific knowledge related to the agents (DOS, DHS, Medical Ser-
707 vice, User), and the artefacts (the I-130 form, the variety of Visas and the
708 Payment method) involved in the process, (ii) specific activities needed in
709 immigration processes, such as «Application Process», «Date Selection», or
710 «Interview», (iii) more generic elements – such as tasks, events, and gate-
711 ways – typically found in process modelling languages. In the figure, a
712 User is required to fill in a standard I-130 form and to set a date for an
713 interview with the embassy personnel. Then, the user is interviewed by
714 the US Department of Home Security (DHS) and they perform a study of
715 the application submitted. Afterwards, the Department of State (DOS)
716 validate the data. In a parallel process, the candidate is expected to go
717 through a Medical Examination. Once both the ordinary verifications and the
718 medical report have been checked, the document is dispatched. Depend-
719 ing on the characteristics of the applicant various types of visa should be
720 issued. In this case, a E1 type Visa is approved and consequently issued;
721 otherwise, the document is denied.

722 **Resource Collection.** In order to gather all the required elements we defined
723 a set of repositories and resources according to the data model shown in
724 Figure 2. A list of resources were taken from: (i) the Object Management
725 Group (OMG)⁶ (<http://www.omg.org/spec/>); (ii) Ecore repositories,
726 such as the ATL zoo (<http://web.emn.fr/x-info/atlanmod/>), which
727 includes meta-models for BPEL, DoDAF, Gantt, ODP, SPEM and some
728 others; and (iii) Ecore files available on the OMG repository with standard
729 meta-models, such as BMM, BPMN, CMMN, DMN, SBVR and SPEM.
730 For the domain-specific concepts, open resources were required. In our
731 case, we took CSVs from the U.S. Open Data Portal (<https://www.data.gov/>)
732 and US e-Government domain ontologies (<http://oegov.org/>),
733 which are available in OWL and RDF formats. This second repository
734 contains ontologies that model the set of United States government facets.
735 These include the Government ontology, the U.S. Government ontology,

⁵<https://www.usa.gov/visas>

⁶The OMG is the standardization body behind many modelling standards such as UML, SysML, MOF or BPMN.

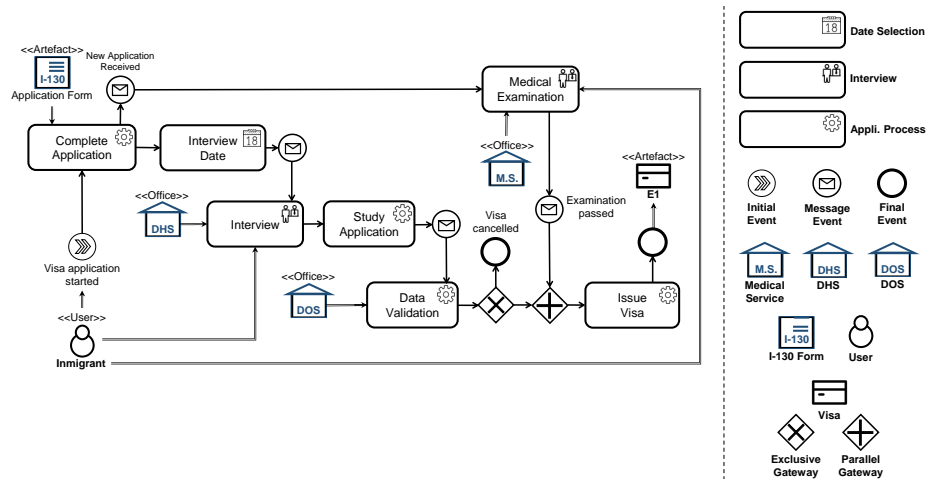


Figure 20: Example process for issuing a visa with IWML (left) and legend (right)

736 the Department of Homeland Security ontology and Department of State
737 ontology.

738 Table 2 summarizes the number of instances of the resource collection
739 by focusing on the meta-level and, in the case of the ontologies, taking
740 into account also the individuals. Even though in this case the resources
741 imported were not large (this will be evaluated in the second case study)
742 we can appreciate that EXTREMO was able to gather information coming
743 from different technological spaces. The total size of the reused artefacts
744 amounted to around 2,4 MB of data.

Ontology concept	Number of instances	EMF concept	Number of instances
OWL file	22	Ecore file/EPackage	30
OWL Class	141	EClass/EDataType	2484
Individuals	1118	EReference	892
owl:ObjectProperty	35	EAttribute	326
owl:DatatypeProperty	91	OCL EAnnotation	0

Table 2: Number of collected instances of different Ontology and EMF-concepts, respectively.

745 **Resource Import.** We imported the Ecore meta-models taken from the OMG
746 and the ATL zoo by the application of our EcoreAssistant (cf. Section 3.1)
747 and the domain-specific concepts by applying our OntologyAssistant and the
748 CsvAssistant.

749 **Meta-Model Construction.** The meta-model was developed using DSL-
750 tao, integrated with EXTREMO following the approach described in Section 4.2.2

751 by means of a drag and drop extension point. Figure 21 shows a moment
 752 in the construction of the meta-model. In particular, it shows the Eclipse
 753 IDE with the EXTREMO perspective open (label 1), which includes the
 754 *Resource Collection* previously mentioned in the repository view (label 2)
 755 and the BPMN.core resource visualized in the resource explorer (label 3).
 756 The meta-model construction phase involved some of the queries listed in
 757 Table 9, such as finding types of forms, organizations and gateways. Once
 758 a query has been issued, the resulting elements can be highlighted on the
 759 original resource (label 4). Finally, a set of semantic nodes can be dropped
 760 over the DSL-tao design diagram (label 5).

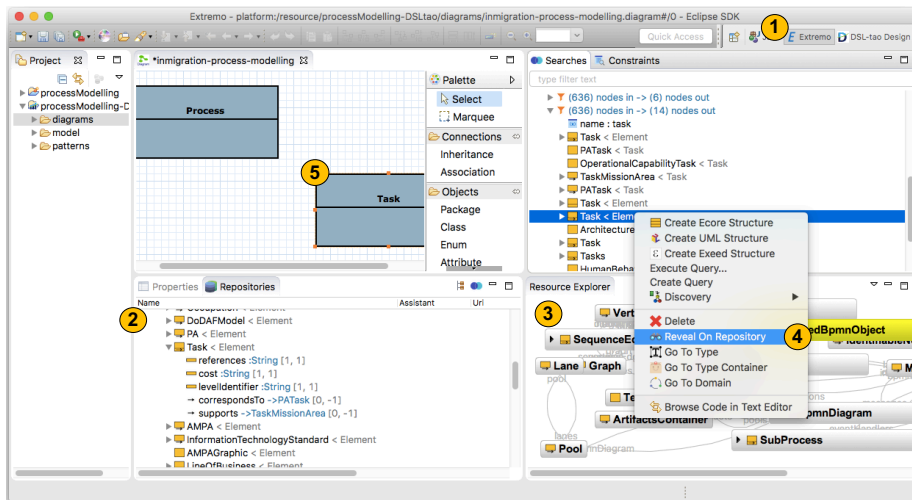


Figure 21: Integration of EXTREMO with DSL-tao (from the perspective of the case study)

761 **Result.** Figure 22 shows the final result obtained. In detail, IWML has elements
 762 originating from the set of meta-models that describe workflows, such as Gateways,
 763 Events or Task. Moreover, IWML is composed of some domain-specific concepts taken
 764 from the set of domain ontologies, like DOS, DHS or MedicalService; CSVs, such as
 765 the type of Visas or the set of Users. Roughly 22% of the classes in our solution
 766 have been obtained from different ontologies and CSVs, 48% of the classes have
 767 been obtained by combining different representations of process modelling meta-
 768 models and the rest (30%) have been added by hand taking information from the
 769 government websites. This suggests that EXTREMO is useful as a help in the
 770 construction of a complex meta-model, as we were able to build the meta-model
 771 by reusing elements from different heterogeneous information sources.
 772
 773

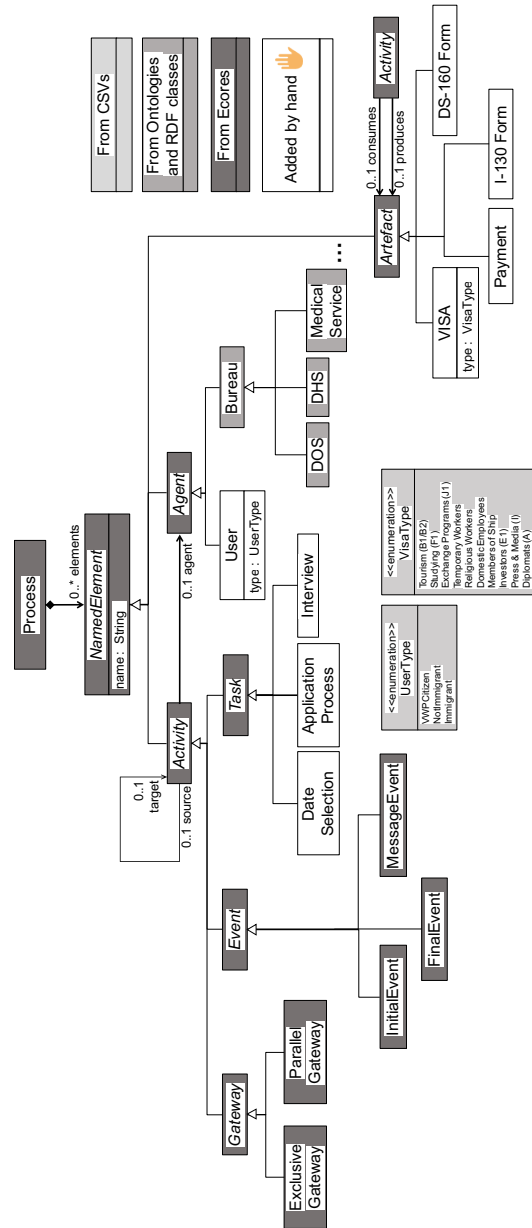


Figure 22: Excerpt of the processes for immigration meta-model.

774 *5.2.2. Industrial production system modelling*

775 In the second case, we present the development of a DSL for industrial
 776 production systems to enable the construction of models that conform to an

777 interoperable cross-industry standard for products and services called eCl@ss⁷.
778 The goal of this case study is to reduce the number of potential candidates for
779 conveyor-belt system-components that are available in the eCl@ss-standard, and
780 thus to conveyor-belt system-modellers, by applying EXTREMO for constructing
781 the desired language.

782 **Scope of the Language.** We construct a language for the modelling of pro-
783 duction systems conforming to the eCl@ss-standard called EPML. The
784 language includes elements from conveyor-belt systems that must fulfill
785 the constraints imposed by the eCl@ss-standard and the GEMOC initia-
786 tive⁸, such as the Signal Process Modelling Language (SigPML)—a DSL
787 dedicated to data flow processing.

788 **Example Model.** Figure 23 shows an example EPML model, which contains
789 (i) electrical drives (DC Engine), (ii) communication cables or ready-made
790 data cables that represent SigPML connectors, (iii) PC-based controls
791 or field buses, i.e., decentralized peripherals, which represent controls,
792 (iv) controls in terms of inductive proximity switches, and (v) sets of
793 rectangular industrial connectors, which represent connector systems. In
794 the figure, a DC Engine is connected to a Fieldbus through a communication
795 cable and a data cable with a set of industrial connectors. The Fieldbus also
796 has a connection with a Proximity Switch through a communication cable and
797 a data cable using the industrial connectors that the Proximity Switch has.

798 **Resource Collection.** In order to gather all the required elements we define
799 a set of repositories and resources according to the data model shown
800 in Figure 2. The resources were taken from: (i) the GEMOC initiative,
801 such as SigPML defined in form of an Ecore meta-model; (ii) the eCl@ss-
802 standard, defined in form of several XML schema definitions (XSDs) and
803 XML instances.

804 Table 3 summarizes the number of instances of the eCl@ss-standard as
805 well as the SigPML by focusing on the meta-level. However, domain-
806 specific concepts in the eCl@ss-standard measure a substantial size, i.e.,
807 only the basic and advanced specifications in the English language consist
808 of 41,000 product classes and 17,000 properties, which amount to 15.5
809 Gb of data. In this situation, extracting desired concepts requires the
810 manual examination of a vast amount of resources as well as their re-
811 implementation by a target DSL. Moreover, any update that is performed
812 on eCl@ss-standard resources, may also impact the implementation in
813 the target system and involve complex and time-consuming maintenance
814 tasks. In an effort to counteract such limitations we apply the EXTREMO
815 framework as follows.

⁷An ISO/IEC-compliant international standard for the unified and consistent definition and classification of products, materials, and services alongside typical supply chains by the use of commodity codes.

⁸<http://www.gemoc.org>

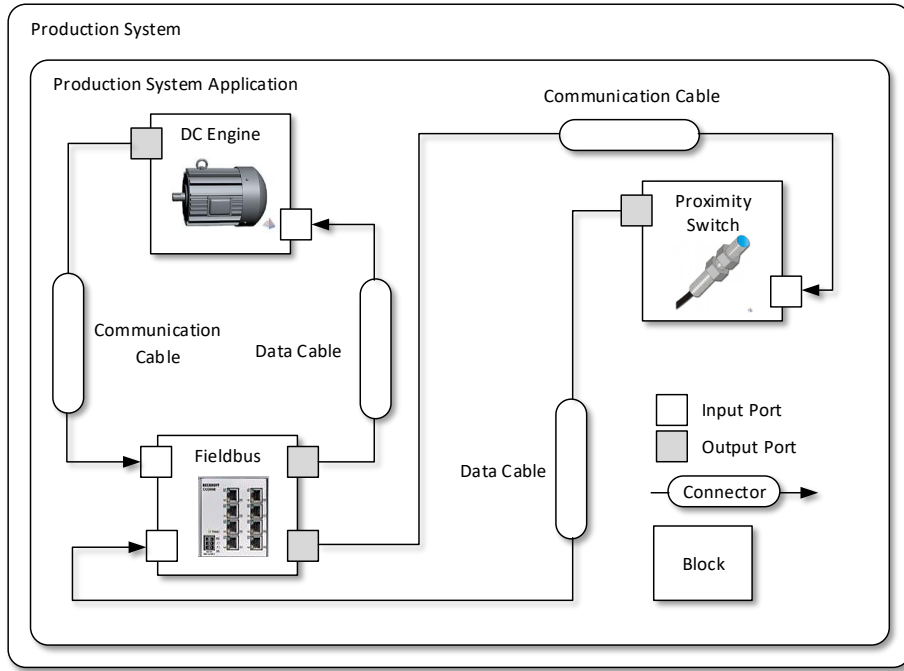


Figure 23: Abstract graphical representation of the conveyor-belt production system model.

XSD concept in eCl@ss-standard	Number of instances	EMF concept in SigPML	Number of instances
XSD file	30	Ecore file/EPackage	1
xs:element	960	EClass	14
xs:element IDREF attribute	110	EReference	18
xs:attribute	104	EAttribute	10
xs:restriction	84	OCL EAnnotation	0

Table 3: Number of collected instances of different XSD and EMF-concepts, respectively.

816 **Resource Import.** First, we import the resource collection by the applica-
 817 tion of the EcoreAssistant and the XsdAssistant. The XsdAssistant reuses func-
 818 tionality of the XMLINTELLEDIT framework [44]—composed of XML-
 819 TEXT [42] and INTELLEDIT [43]. The XMLTEXT framework transforms
 820 XML-artifacts, i.e., XSDs and XML instances, to corresponding MDE-
 821 artifacts, i.e., Ecore meta-models and conforming models. Then, the Eco-
 822 rAssistant is used to map the MDE-artifacts into the common data model.

823 **Meta-Model Construction.** Next, we employ the EXTREMO Eclipse perspec-
 824 tive as well as the Sample Reflective Ecore Model Editor, integrated with

825 EXTREMO following the approach described in Section 4.2.1. Figure 24
 826 shows a moment in the construction of the EPML meta-model. In partic-
 827 ular, it shows the set of resources in the repository view (label 1) and the
 828 EXTREMO functionalities involved in the meta-model construction phase
 829 for querying (label 2), traversing (label 3), and applying desired concepts
 830 from the imported repositories (label 4). For example, available concepts
 831 that represent an electrical drive in the eCl@ss-standard are gathered by
 832 issuing an EXTREMO query for retrieving semantic nodes that are named
 833 “engine” and then used for creating corresponding concepts in the EPML
 834 meta-model. EXTREMO traversal-functionalities, such as Reveal On Repos-
 835 itory, Go To Type, and Go To Domain, are employed for gathering respective
 836 classes, which are referenced as super-types and thus enforce conformance
 837 to the eCl@ss-standard.

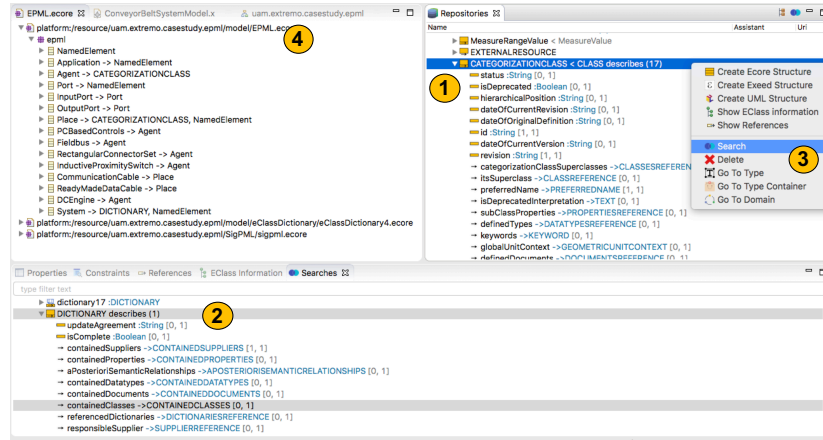


Figure 24: Integration of EXTREMO with the Sample Reflective Ecore Model Editor (from the perspective of the case study)

838 The final result of the EPML meta-model construction process is depicted
 839 in Fig. 25. In detail, the EPML data flow process elements originated
 840 from the SigPML (in dark-grey) such as System, Application, Block, Connec-
 841 tor, and Port. Moreover, EPML is composed of several eCl@ss-standard
 842 concepts (in light-grey), which include (i) electrical drives, (ii) cables,
 843 (iii) controls, (iv) binary sensors, i.e., safety-related sensors, and (v) con-
 844 nector systems. For example, the eCl@ss-standard CATEGORIZATIONCLASS
 845 represents the super-type of Block and Connector in EPML. Additionally,
 846 subtypes of Block and Connector are also instances of CATEGORIZATIONCLASS
 847 in the eCl@ss-standard. As a result of distinguishing specific instances
 848 of categorization-classes adds additional EPML-specific semantics that
 849 gather concepts found in SigPML and the eCl@ss-standard.

850 **Result.** Finally, we evaluate the capability of handling large models as they
 851 occur in the eCl@ss-standard in form of XML files, which are transformed

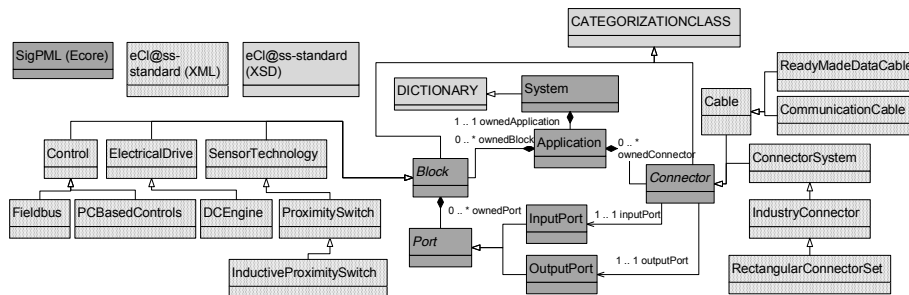


Figure 25: Excerpt of industrial production system meta-model based on SigPML and the eCl@ss-standard.

852 to XMI files by the XMLINTELLEDIT framework to enable their use by
 853 EXTREMO. Further, SigPML (only) contains 13 semantic nodes at meta-
 854 model level and none at model-level and is thus neglected in Table 4.
 855 To summarize, the meta-level contains one *Resource*, i.e., “EPML.ecore”,
 856 which references 18 different XSD files, that is instantiated by a single
 857 XML file, i.e., “eClass9.1_BASIC_EN_SG.27.xml” (55.6 MB). Moreover,
 858 at the model-level there are 487,746 instances of semantic nodes (525 differ-
 859 ent kinds), 805,097 instances of object properties (500 different kinds),
 860 487,745 instances of data properties (26 different kinds), and 820,356 in-
 861 stances of constraints⁹ (88 different kinds).

862 Consequently, our results indicate that the EXTREMO-constructed EPML
 863 reduces the number of potential candidates for conveyor belt system com-
 864 ponents, which are available in the eCl@ss standard, by approximately
 865 99.17% (97.05%), i.e., from 487,746 (805,097) to 4,071 (23,752) semantic
 866 nodes (object properties) that represent instances (references) of CATEGO-
 867 RIZATIONCLASS and thus potential candidates for instances (references) of
 868 (to) Block and Connector in SigPML.

869 **Additional comments.** EPML may be extended by either adding further
 870 eCl@ss-standard specific concepts, which represent instances of CATEGO-
 871 RIZATIONCLASS, to the meta-model or by expressing the concept of blocks
 872 and connectors as concrete (instead of abstract) classes. In more detail,
 873 the latter option would move the decision making-process of choosing de-
 874 sired eCl@ss-standard elements from meta-model level to model-level. Al-
 875 though EXTREMO supports such cases by the means of level-agnostic data
 876 handling, we choose to constrain EPML at meta-model level to limit the
 877 set of possible types, which can be instantiated at model-level, and thus
 878 fit the purpose of modelling conveyor-belt production systems.

⁹Note that the “number of instances” of constraints refers to the number of constraints defined at meta-level and validated at model-level.

Common Data Model concept	Number of instances	
	Meta-level (types)	Model-level (data)
Resource	1	1
SemanticNode	525	487,746 (4,071)
ObjectProperty	500	805,097 (23,752)
DataProperty	26	487,745
Constraint	88	820,356

Table 4: Instances of imported Common Data Model concepts within the industrial production system modelling case study.

879 *5.2.3. Summary of the demonstration cases*

880 The processes for immigration case study (Section 5.2.1) imports and queries
881 data from different technical spaces, i.e., Ecore meta-models, CSV files, and
882 OWL specifications and the industrial case study (Section 5.2.2) considers XML
883 schemas, XML instances, and Ecore meta-models. Thus, in the first one, we
884 consider a greater variety of technical spaces and smaller models. In contrast,
885 in the second one we address the importing of a lower variety of technical spaces
886 but larger models, i.e., XML instances in the size of multiple gigabytes. Then, in
887 the first case study, we evaluate the ability of EXTREMO in providing assistance
888 during the modeling of resources that are originated from a variety of technical
889 spaces, and in the second case study, we evaluate the applicability of EXTREMO
890 in assistance-scenarios that require dealing with industrially-sized resources.

891 Table 5 summarizes the number of meta-classes obtained from each assist-
892 tant in both cases. In the first one, a total of 6 meta-classes were obtained
893 according to the domain-specific concepts, 12 meta-classes were obtained from
894 different ecores and the rest were added by hand. In the second one, a total
895 of 16 meta-classes were obtained from different schemas and descriptions us-
896 ing the XSD assistant and the rest from ecores. Overall, most content in both
897 meta-models was reused from the available resources, which were taken from 4
898 different technical spaces.

899 From these demonstration cases, we can answer *RQ1: How useful is EX-*
900 *TREMO to solve practical problems in language engineering?* by stating that
901 EXTREMO was helpful in locating elements within heterogeneous resources that
902 helped to create the meta-models. These elements could be directly inserted in
903 the final meta-model. For both cases, most elements in the meta-models were
904 reused from the artefacts in the repository.

905 *5.3. Evaluating format extensibility*

906 In order to evaluate format extensibility, we perform an analytical evalua-
907 tion of the degree of coverage of the data model of common features found in
908 information modelling approaches [48, 49]. Figures 26 and 27 show a feature

Table 5: Evaluating the usefulness of EXTREMO: results of the experiments

Measures	Case 1: Processes for immigration	Case 2: Production Systems
Size of metamodel obtained	27 metaclasses	23 metaclasses
From Ecore Assistant	13 metaclasses	7 metaclasses
From Ontologies Assistant	4 metaclasses	N/A
From CSV Assistant	2 metaclasses	N/A
From XSD/XML Assistant	N/A	16 metaclasses
Manually added	8	N/A

909 diagram (splitted in two parts for readability) displaying these features. Our
 910 aim is not to be fully exhaustive, but to cover a reasonable design space for
 911 information modelling approaches.

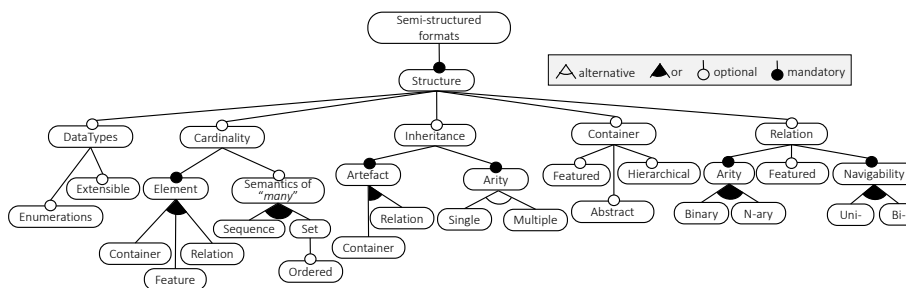


Figure 26: Feature model of characteristics of structured formats (1/2).

912 Figure 26 shows features related to the space of possible supported structure
 913 primitives. Formats to represent semi-structured information are based on some
 914 kind of container for data (models, nodes, objects, classes), and on relations
 915 among them (features Container and Relation) [49, 50]. Containers may support
 916 features (fields, references), may support nesting, and have ways to control
 917 their instantiability (e.g., abstractness tag). Relations have an arity, which is
 918 typically either binary (to model relations between exactly two containers) or
 919 n-ary (to model multi-container relations). Similar to containers, some systems
 920 may allow relations to own features. Relations may be navigable either in one
 921 or both directions [51, 52].

922 Some systems support some form of inheritance to reuse information [51].
 923 Inheritance relations can normally be set either between containers or relations,
 924 and be single (at most one super) or multiple (any number of super elements).
 925 Often, systems support a notion of cardinality, to specify the expected range of
 926 values of a given element can take. Typically, cardinalities can be attached to
 927 containers, relations or features. Some systems permit specifying the semantics
 928 of “many” cardinality: a set (no repetition allowed) optionally ordered, or a

929 sequence (repetitions allowed) [51, 52]. Finally, many systems may have prede-
 930 fined data types (like integer, String, etc), have support for enumerations, and
 931 be extensible with new data types, provided by the user.

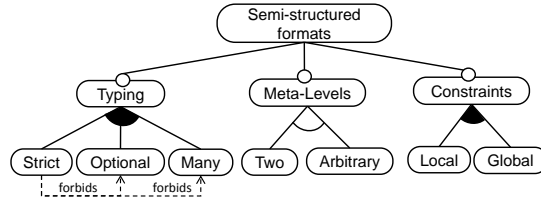


Figure 27: Feature model of characteristics of structured formats (2/2).

932 Figure 27 captures additional features. Some systems organize their elements
 933 in meta-layers, typically two (e.g., models and meta-models), but others support
 934 an arbitrary number of them. If meta-levels are supported, some kind of typing
 935 is needed between entities in different layers. In some cases, this typing can
 936 be optional, or allowed to be multiple. In other cases, the typing is strict,
 937 meaning that each element of a certain level is typed exactly by one entity at
 938 the level above [53]. This precludes optional typing, multiple typings and typing
 939 relations within the same level [54]. Finally, some systems support constraints,
 940 typically defined on one meta-level, and evaluated in some meta-level below.
 941 These constraints can be local (i.e., involving features of just one entity) or
 942 global (if entities of the whole resource need to be access).

943 Once we have set the design space for information modelling formats, we
 944 analyse how different technologies are positioned in it, and the degree of coverage
 945 of our data model. A summary of such analysis is shown in Table 6.

Table 6: Features of some information modelling technologies: DataType (E=Extensible, F=Fixed, EN=Enumerations), Cardinality (C=Container, F=Feature, R=Relation, Sem=Configurable semantics of many), Inheritance (C=Container, R=Relation, S=Single, M=Multiple), Container (F=Featured, A=Abstract, H=Hierarchical), Relation (Bin=Binary, N=N-ary, F=Featured, U=Unidirectional, B=Bidirectional), Typing (S=Strict, O=Optional, M=Many), Meta-Levels (2=Two, A=Arbitrary), Constraint (L=Local, G=Global).

System	DT	Card	Inh	Container	Relation	Typing	Levels	Const
EMF	E, EN	F, Sem	C, M	F, A, H	Bin, U, B	S	2	L, G
UML	E, EN	F, Sem	C, M	F, A, H	N, U, B, F	M	2	L, G
METADEPTH	F, EN	C, F, R, Sem	C, M	F, A, H	Bin, U, B, F	O	A	L, G
XSD	E, EN	F, Sem	C, S	F, H	Bin, U	S	2	L
OWL	E, EN	F, Sem	C, M	F, A	Bin, U, B	O, M	A	L
CSV	F	-	-	F	-	-	-	-
EXTREMO	F	F	C, M	F, A, H(Rsr)	Bin, U, B	O, M	A	L, G

946 In the modelling technical space, we have taken three representatives: EMF,
 947 UML and METADEPTH. It can be seen that EMF allows enumerations and
 948 extensible data types, permits cardinalities on features and fine-tuning the se-
 949 mantics of “many”. It supports multiple inheritance on classes, classes (but not
 950 references) may have features, and can be abstract. Both packages and classes

951 can be organized hierarchically (classes may define containment references that
952 contain other classes). Relations are binary, and can be bidirectional (emulated
953 through opposite references). The typing is strict and on a two meta-level archi-
954 tecture. Constraints can be expressed by OCL and can be both local and global.
955 UML offers similar features, but includes N-ary, featured relations (association
956 classes), and typing can be multiple (through overlapping generalization hier-
957 archies). Finally, METADEPTH permits adding cardinalities on nodes, relations
958 (edges) and features. Edges can have features, and there is optional typing on
959 an arbitrary number of meta-levels.

960 In addition to the modelling technical space, we are interested in evaluating
961 representatives from other technical spaces. First, we selected XSD which offers
962 enumerations and extensible data types as well as cardinalities on features with
963 different configurations for the semantics of “many”. Furthermore, XSD offers
964 for element types the following three possibilities: single inheritance, features,
965 and nesting of element types (i.e., hierarchies). XSD only allows binary uni-
966 directional references. Typing in XSD is considered to be strict, except open
967 points in XSD descriptions which allow for any valid XML structure. XSD fol-
968 lows the classical two-level approach and allows for local constraints. For global
969 constraints, additional format languages such as Schematron have to be used.
970 OWL has similar features, but it allows for multiple inheritance between classes
971 which may be also abstract classes. There is no explicit hierarchy based on
972 nesting classes. Relations in OWL may be defined as bi-directional and addi-
973 tion to many other relationship types. Interestingly, OWL allows for optional
974 typing as well as multiple types. Furthermore, arbitrary modeling levels may
975 be defined with OWL, to be more precise, with OWL Full. Local constraints
976 are supported, however, for global constraints, additional constraint languages
977 such as SHACL [55] have to be used.

978 It can be seen that the data model of EXTREMO supports most features,
979 with some limitations that can either be overcome, or are not important for
980 EXTREMO’s goals, as explained next. First, EXTREMO’s data types are currently
981 not extensible. Instead, unsupported data types (e.g., currency) need to be
982 mapped to an existing one (e.g., String) and the can be annotated using `MetaData`
983 objects. Similarly, enumerations need to be stored as Strings. However, this
984 is not problematic when issuing queries. Cardinality can only be placed on
985 features, and the semantics of “many” is not configurable. However, this is not
986 an important feature to issue queries, and can be reflected using `MetaData` or
987 `Constraint` objects. Inheritance is on containers and can be multiple. This is in-
988 line with most analyzed systems. Containers can have features and be abstract.
989 However, only resources can be hierarchial. Nonetheless, hierarchy of semantic
990 nodes can be emulated by `ObjectProperties`. Relations (`ObjectProperties`) are binary,
991 can be bidirectional (by declaring opposites) and may have features. N-ary or
992 featured associations can be emulated by adding an intermediate `SemanticNode`.
993 This is the strategy we followed when building the assistant for METADEPTH.
994 EXTREMO’s typing is optional and multiple, supporting an arbitrary number of
995 levels. Finally there is support for both local and global constraints.

996 Please note that the common data model can also accommodate data formats

997 with no explicit descriptions, like e.g. CSV. In such a case, each data row would
998 be imported as a semantic node, and each cell as a data property.

999 Altogether, from this analytical evaluation, we can conclude that most com-
1000 mon features of information modelling approaches can be directly mapped to
1001 our common data model, or can be emulated. Therefore, we can answer *RQ2:*
1002 *How capable is Extremo to represent information coming from different techno-*
1003 *logical spaces?* by stating that EXTREMO will be able to accommodate most
1004 commonly used information modelling approaches.

1005 5.4. Evaluating the integration with external tools

1006 The idea of EXTREMO is to be a modelling assistant easy to integrate in
1007 other (meta-)modelling tools. Hence, we have assessed to what extent this
1008 integration is feasible, by integrating EXTREMO with a set of tools developed
1009 by third-parties. In some cases, the original code was not accessible, while in
1010 others it was. In the first case, we used the UML model editor¹⁰ (as shown in
1011 Figure 17), the standard Ecore editor and Exeed, an enhanced version of the
1012 built-in EMF reflective tree-based editor that enables developers to customize
1013 the labels and icons of model elements¹¹. All these solutions are based on a
1014 TreeEditor, an extensible part of the Eclipse infrastructure. Since a drag and drop
1015 integration is not possible because of restrictions to access to the original code,
1016 the solution was performed by means of the action extension point. Each of
1017 these integrations costed 234 lines of Java code (LOCs) in average, which can
1018 be considered as very light.

1019 In the second case, we used DSL-tao, which was built by our team. In this
1020 case, the code was available, and performing a solution by means of the drag and
1021 drop extension point. costed 134 lines of Java code (LOC).

1022 Table 7 shows details on the number of LOCs for each integration. The
1023 integration mechanisms by means of actions and drag and drop are already provided
1024 by the tool (marked with an asterisk) and do not need to be provided by the
1025 developer. Therefore, most of the code needed was related to the transformation
1026 from the instances of our data model (Figure 2) to the classes of the modelling
1027 tool (cf. Figures 16 and 18). In the case of the integration made by means of
1028 actions, the method execute needs to resolve the editor part that will receive the
1029 portion of the model instance and the selected elements from the views before to
1030 create the new elements of the transformation (shown in row 3). The necessary
1031 LOCs to transform nodes, data and object properties are detailed in rows 4-6
1032 of the table.

1033 Thus, from this study, we can answer *RQ3: How integrable is EXTREMO?*
1034 by stating that integration of EXTREMO is lightweight for modelling tools based
1035 on tree or GEF-based editors.

¹⁰UML2-MDT, www.eclipse.org/modeling/mdt

¹¹Epsilon Exeed, <http://www.eclipse.org/epsilon/>

	DSL-tao	EcoreEditor	UML2Editor	ExeedEditor
<i>Ext. Point Used</i>	drop	actions	actions	actions
<i>Ext. Point Integration</i>	59*	49*	49*	49*
<i>Tree Selection Solver</i>	-	163	165	163
<i>SemanticNode</i>	24	8	4	8
<i>DataProperty</i>	27	33	4	33
<i>ObjectProperty</i>	24	9	32	9

Table 7: LOCs for integrating EXTREMO with other tools

1036 *5.5. Discussion and threats to validity*

1037 As we have seen in the three preceding subsections, we were able to use EX-
1038 TREMO to help in constructing DSLs by reusing heterogeneous artefacts (some
1039 of which had large size); we analysed the degree in which the data model of
1040 EXTREMO is able to accommodate possible information modelling approaches;
1041 and how easy is it to integrate EXTREMO with external (meta-)modelling tools.
1042 While the results are positive, there are of course also potential threats to the
1043 validity of the experiments. According to Wohlin et al. [56], there are four basic
1044 types of validity threats that can affect the validity of our study. We cover each
1045 of these in the following paragraphs.

1046 *5.5.1. Construct Validity*

1047 Construct validity is concerned with the relationship between theory and
1048 what is observed. The demonstration cases in Section 5.2 focussed on evaluating
1049 the use of EXTREMO with assistants for different technologies, and standards
1050 (like eCl@ss) developed by third parties. However, although taking realistic
1051 requirements, the DSLs to be constructed were devised by us. Therefore, further
1052 studies would need to be performed by constructing DSLs with requirements
1053 specified by third parties.

1054 The evaluation of the demonstration cases focussed on DSL construction.
1055 However, it used artefacts acting as descriptors (XSD) and at the model/data
1056 level (XML documents). While this shows that EXTREMO can be used to extract
1057 information at the model level, a further study would be needed to assess the
1058 usefulness of EXTREMO for domain-specific modelling. However, please note
1059 that creating a meta-model is a structural modelling activity already.

1060 *5.5.2. Conclusion Validity*

1061 Conclusion validity is concerned with the relationship between the treat-
1062 ment and the outcome. We considered two demonstration cases from different
1063 domains, seven format languages from four technical spaces, and integrated our
1064 approach with four modeling editors. While these numbers may be not enough
1065 to reason about statistical relevance, they still show a clear tendency of the
1066 usefulness, applicability, and integrability of our approach.

1067 *5.5.3. Internal Validity*

1068 Internal validity checks whether the treatment used in the experiment actu-
1069 ally causes the outcome. We were the performers of both demonstration cases.
1070 While the performer of one of the case study was not involved in the devel-
1071 opment of EXTREMO, a user study would be needed to further assess the tool
1072 usability and the subjective usefulness of EXTREMO. However reporting on a
1073 user study would deserve a separate publication, and we will tackle this issue
1074 in future work. Similarly, the integration of EXTREMO with external tools was
1075 also performed by us. Although lightweight in terms of LOC, it could be more
1076 demanding for other developers in terms of effort.

1077 Another aspect is that we set the class as the unit of reuse, neglecting prop-
1078 erties. We believe this is a good indicator as the number of classes outperforms
1079 that of properties.

1080 Having good resources available is crucial for the approach to work properly.
1081 We did not evaluate how easy is it to perform this phase of resource collection
1082 (since this phase is out of the scope of our tool), but we evaluated how large
1083 was the context of the repository, though.

1084 *5.5.4. External Validity*

1085 Regarding external validity (i.e., generalizability of the results), we did not
1086 include an explicit evaluation of query extensibility, because the extension points
1087 we have defined permit adding new queries by using arbitrary Java code. Table 9
1088 in the appendix lists a collection of queries we have defined by implementing the
1089 extension point and that covers a set of accepted quality criteria in conceptual
1090 modelling [24].

1091 For RQ3 (integrability) we did not evaluate the integration of EXTREMO
1092 with text-based modelling tools, e.g., built with Xtext, but we have assessed
1093 to what extent this integration is feasible, by integrating EXTREMO with a set
1094 of tools developed by third-parties (and also developed by us). In addition, we
1095 integrated EXTREMO with other tools within Eclipse, but not with tools in other
1096 IDEs, like JetBrains. While EXTREMO is an Eclipse plugin, its Core subsystem
1097 (described in Section 4.1) is largely independent from it (in contrast to the UI
1098 subsystem 4.2). Hence, migrating the Core into JetBrains would require little
1099 effort, but the UI subsystem (dealing with visualization and interaction with
1100 resources and query results) would need to be redesigned.

1101 We did not present a formal evaluation of scalability or performance, which
1102 are left for future work. Regarding the former, the XML artefacts considered in
1103 the second demonstration case reached a size of 55Mb. Moreover, resources are
1104 imported and persisted using NeoEMF, a model persistence solution designed
1105 to store models in NoSQL datastores, which is able to handle very large models
1106 efficiently (e.g., models of more than 40.000.000 elements were reported to be
1107 created in [57]).

1108 Regarding performance, our experience and preliminary evaluations indicate
1109 that resource import time is linear in the size of the resource. Typically, it takes
1110 a few seconds for resources of sizes in the order of hundreds of elements. While
1111 we plan to optimize this performance, this is a one-time operation, and once

1112 a resource is imported, it can be handled through NeoEMF. Regarding query
1113 performance, those that need to traverse the whole resource are in the order of
1114 one second for sizes up to thousands of elements. However, they may become a
1115 bottleneck for larger resources. To alleviate this issue, we cache both the input
1116 parameters for predicate-based searches (init operation shown in Figure 6) and
1117 the query results, while NeoEMF lazy-loading mechanisms that transparently
1118 brings into memory model elements only when they are accessed. Further op-
1119 timizations to speed-up queries, e.g., based on the creation of derived features
1120 and indexes for the resources [58], are left for future work.

1121 6. Related work

1122 The increasing complexity of software development has prompted the need
1123 for code recommenders for example, for API usage, or program quick fix. How-
1124 ever, although code recommenders are increasingly used in programming IDEs [7,
1125 8], there is lack of such assistive tools for (meta-)modelling in MDE.

1126 The closest work to our proposal is [59, 60, 61], where a generic architec-
1127 ture for model recommenders is proposed. The architecture is extensible, in the
1128 sense that different recommender strategies can be plugged-in. In contrast, the
1129 extensibility of our approach is in the supported data source, while we specifi-
1130 cally focus on the extraction of knowledge from these sources. In addition, our
1131 approach supports out-of-the-box visualization and extensible query facilities.

1132 Other approaches to model recommendation focus on proposing suitable
1133 ways to complete a model with respect to a meta-model [62]. Hence, using
1134 constraint solving techniques, the system proposed ways to complete a model
1135 so that it becomes a valid instance of a meta-model. In [63] the authors use
1136 ontologies in combination with domain-specific modelling, and hence can use
1137 ontology reasoners to provide reasoning services like model validation, inconsis-
1138 tency explanation, and services to help in the model construction phase.

1139 Some approaches propose a common architecture to index and represent
1140 models, with the purpose of reuse. For example, in [64] the authors transform
1141 SysML models into a “universal” representation model called RHSP, which can
1142 be queried to discover reusable models. They support queries based on par-
1143 tial models (and model similarity search) and natural language (similar to our
1144 synonym searches). In our case the queries are extensible, and our data model
1145 provides richer support for representing model features, including constraints.

1146 Instead of using a common data model, an alternative design would have
1147 been to use model adapters, in the style of the Epsilon model management
1148 languages [65]. In this approach, the languages do not access the particular
1149 information technology (EMF, XML) directly, but through a model connectivity
1150 layer. This is an interface that can be implemented to enable uniform access to
1151 different technologies. We opted for a common data model, where the different
1152 heterogeneous elements are reified uniformly, and stored using NeoEMF, hence
1153 providing scalability and performance.

1154 Storing artefacts in a database, to enable their flexible query has also been
1155 applied to source code [15, 66]. In our case, the artefacts come from different

1156 heterogeneous sources, and hence we need to transform them into the common
1157 data model. Our query approach is extensible, based on extension points.

1158 Other works have considered the exchange of models/data between different
1159 meta-modelling tools [67] or technical spaces [68]. In [67] the authors propose
1160 a solution that creates transformation between different meta-modelling tech-
1161 nologies by means of declarative mappings. Our approach differs from these
1162 works by mapping the technical spaces into a common data model instead of
1163 establishing mappings between individual technical spaces. As a result, our ap-
1164 proach is independent of a single technical space and thus enables the import,
1165 persistence, and querying of interdependent concepts that are originated from
1166 distinctive technical spaces and may be lost within single mappings. In [68]
1167 the maintenance of intra-space transformations is improved by automating the
1168 discovery and reuse of mappings between schema elements. In contrast, EX-
1169 TREMO provides assistance during the import of artifacts from different tech-
1170 nical spaces and the creation of new languages and models that are based on
1171 existing technical spaces, such as Ecore, regardless of their originating technical
1172 space. Although EXTREMO requires to specify assistants for different technical
1173 spaces, existing EMF-based work that bridges technical spaces, such as XML-
1174 TEXT [42] for XML schema, can be reused and (only) requires the specification
1175 of a mapping within the same technical space, i.e., Ecore in case of EXTREMO
1176 and XMLTEXT.

1177 Some researchers have exploited ontologies for creating DSLs [69]. For ex-
1178 ample, in [70] the authors advocate the use of (OWL) ontologies in the domain
1179 analysis phase of DSL construction. As they target textual DSLs, they propose
1180 a tool for the automated generation of a textual grammar for the DSL. In a
1181 similar vein, in [71], the authors generate meta-model design templates from
1182 OWL ontologies, which are later manually refined into domain meta-models.
1183 In our approach, we assume that not all the required information to create a
1184 meta-model is present in one ontology, but typically such information is scat-
1185 tered in informational resources of different kinds, like ontologies, RDF data, or
1186 meta-models.

1187 Combining modeling approaches from MDE with ontologies has been studied
1188 in the last decade [72]. There are several approaches to transform Ecore-based
1189 models to OWL and back, e.g., cf. [73, 74]. In addition, there exist approaches
1190 that allow for the definition of ontologies in software modeling languages such
1191 as UML by using dedicated profiles [75]. Moreover, there are approaches which
1192 combine the benefits of models and ontologies such as done in [76, 77] for rea-
1193 soning tasks. Not only the purely structural part of UML is considered, but
1194 some works also target the translations of constraints between these two tech-
1195 nical spaces by using an intermediate format [78]. For the data import, we
1196 may build on these mentioned approaches, but we focus on recommendation
1197 services exploiting the imported data from different technical spaces to build
1198 domain-specific modeling languages.

1199 Finally, there are some approaches directed to search relevant models within
1200 a repository. Their aim is slightly different from our goal, which is looking for rel-
1201 evant information within a repository. Moogle [34] is based on textual, “Google-

1202 like” queries, similar to ours. As they focus on EMF model-level queries, they
 1203 use the meta-model information for filtering, like we do as well. However, our
 1204 queries are extensible, and hence new types of queries can be defined. Moreover,
 1205 their results are shown in textual format and we parse and aggregate the results
 1206 as well as offer graphical visualization. EMF query is directed to search EMF
 1207 models [79], using OCL queries or text-based search. The latter may include
 1208 regular expressions, but does not look for relevant synonyms as we do. More-
 1209 over, our extensible approach supports technologies like Ecore, OWL and RDF.
 1210 Furthermore, there are dedicated approaches offering search capabilities tailored
 1211 for a specific modelling domain such as [80, 81]. Although these approaches al-
 1212 low to reason on behavioral similarity aspects, we aim for general model search
 1213 support independently of the modelling domain and technical space.

Work	Assistance	Heterogeneous Sources	Common Model	Queries
Dyck et al. [59, 60, 61]	✓	✗	✗	✗
Sen et al. [62]	✓	✗	✗	✗
Walter et al. [63]	✓	~ (OWL)	✗	✗
Mendieta et al. [64]	✗	~ (SysML)	✓	Not extensible
Kern, Dimitrieski et al. [67, 68]	✗	✓	✗	✗
Ontology-based DSL development [70, 71]	✗	~ (OWL)	✗	✗
Moogle [34]	✗	~ (EMF)	✗	Not extensible
EXTREMO	✓	✓	✓	Extensible

Table 8: Summary comparison of EXTREMO and closest related works

1214 Table 8 presents a feature-based summary of EXTREMO and the closest
 1215 related works. In summary, our approach is novel in that it provides an as-
 1216 sistant system that profits from the integration and querying of heterogeneous
 1217 information sources. Although some approaches have focussed on using specific
 1218 technologies, such as Ontologies [63, 69, 70, 71], to build (meta-)models, our ap-
 1219 proach is more general as a result of supporting different technologies. Moreover,
 1220 there exist approaches that establish bridges between technical spaces [67, 68],
 1221 our contribution differs by providing a common data model to store, query, and
 1222 establish assistance for information from different technical spaces. Further, in
 1223 contrast to other existing approaches, which have devised query mechanisms
 1224 to search for relevant models in a repository [34], our querying mechanism is
 1225 extensible. Finally, some approaches to provide model assistance are based on
 1226 model completion (w.r.t. a meta-model) [62] or provide a generic mechanism
 1227 to plug-in assistants [59, 60, 61]. Contrarily, we contribute a specific architec-
 1228 ture to support assistance that is based on querying heterogeneous information
 1229 sources.

1230 7. Conclusions and future work

1231 In this paper, we have presented EXTREMO, an extensible assistant for mod-
 1232 elling and meta-modelling. The system is able to gather information from differ-
 1233 ent technological spaces (like ontologies, RDF, XML or EMF), by representing
 1234 this information under a common data scheme. This enables their uniform
 1235 querying and visualization. EXTREMO is independent of the particular mod-
 1236 elling tool, but easily integrated with them due to its modular architecture

1237 based on extension points. We have shown its integration with DSL-tao and
1238 several other tools, and used it for the construction of DSLs in the e-Government
1239 and production systems domain. We have performed an evaluation of several
1240 aspects, showing good results.

1241 In the future, we plan to connect EXTREMO with meta-model repositories,
1242 such as MDEFoorge [82]. EXTREMO currently supports a re-active integration
1243 mode, where the assistant is explicitly invoked.

1244 Similar to [60], we would also like to explore pro-active modes for assistance.
1245 For this purpose, we plan to use recommendation techniques based on rich
1246 contextual models, which take into account not only the current model state,
1247 but also the user interaction with the IDE [83]. We are currently considering a
1248 user study, made of two parts. First, we will evaluate the perceived usefulness
1249 of EXTREMO by engineers in order to perform different modelling tasks (e.g.,
1250 construct or modify a model). Second, we will compare the quality of the
1251 resulting models, and the effectiveness of the modelling task, with respect to not
1252 using assistance. Finally, we plan to improve query efficiency by using model
1253 indexes [58].

1254 **Acknowledgements.** We would like to thank to the reviewers for their valu-
1255 able comments. This work was supported by the Ministry of Education of
1256 Spain (FPU grant FPU13/02698); the Spanish MINECO (TIN2014-52129-R);
1257 the R&D programme of the Madrid Region (S2013/ICE-3006); the Austrian
1258 agency for international mobility and cooperation in education, science and re-
1259 search (OeAD) by funds from the Austrian Federal Ministry of Science, Research
1260 and Economy - BMWFW (ICM-2016-04969); and by the Christian Doppler
1261 Forschungsgesellschaft, the Federal Ministry of Economy, Family and Youth and
1262 the National Foundation for Research, Technology and Development, Austria.

1263 References

- 1264 [1] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering
1265 in Practice, 2nd Edition, Morgan & Claypool, 2017.
- 1266 [2] D. C. Schmidt, Guest editor's introduction: Model-driven engineering,
1267 Computer 39 (2) (2006) 25–31.
- 1268 [3] A. R. da Silva, Model-driven engineering: A survey supported by the unified
1269 conceptual model, Computer Languages, Systems & Structures 43 (Supple-
1270 ment C) (2015) 139 – 155. doi:[https://doi.org/10.1016/j.cl.2015.
1271 06.001](https://doi.org/10.1016/j.cl.2015.06.001).
- 1272 [4] UML 2.5 OMG specification, <http://www.omg.org/spec/UML/2.5/>.
- 1273 [5] S. Kelly, J. Tolvanen, Domain-Specific Modeling - Enabling Full Code Gen-
1274 eration, Wiley, 2008.
- 1275 [6] J. E. Hutchinson, J. Whittle, M. Rouncefield, Model-driven engineering
1276 practices in industry: Social, organizational and managerial factors that
1277 lead to success or failure, Sci. Comput. Program. 89 (2014) 144–161.

- 1278 [7] Eclipse Code Recommenders, <http://www.eclipse.org/recommenders>.
- 1279 [8] M. P. Robillard, R. J. Walker, T. Zimmermann, Recommendation systems
1280 for software engineering, *IEEE Software* 27 (4) (2010) 80–86.
- 1281 [9] J. Bézivin, Model driven engineering: An emerging technical space, in:
1282 Generative and Transformational Techniques in Software Engineering, In-
1283 ternational Summer School, GTTSE, Vol. 4143 of Lecture Notes in Com-
1284 puter Science, Springer, 2005, pp. 36–64.
- 1285 [10] I. Kurtev, J. Bézivin, M. Aksit, Technological spaces: An initial appraisal,
1286 in: International Symposium on Distributed Objects and Applications,
1287 DOA 2002, 2002.
1288 URL <http://doc.utwente.nl/55814/>
- 1289 [11] Á. M. Segura, A. Pescador, J. de Lara, M. Wimmer, An extensible meta-
1290 modelling assistant, in: IEEE EDOC, IEEE Computer Society, 2016, pp.
1291 1–10.
- 1292 [12] J. de Lara, E. Guerra, Deep meta-modelling with metadepth, in: TOOLS,
1293 Vol. 6141 of Lecture Notes in Computer Science, Springer, 2010, pp. 1–20.
- 1294 [13] eCl@ss Standard 9.0, <http://wiki.eclass.eu/>.
- 1295 [14] P. A. Ménard, S. Ratté, Concept extraction from business documents for
1296 software engineering projects, *Autom. Softw. Eng.* 23 (4) (2016) 649–686.
- 1297 [15] E. Linstead, S. K. Bajracharya, T. C. Ngo, P. Rigor, C. V. Lopes, P. Baldi,
1298 Sourcerer: mining and searching internet-scale software repositories, *Data*
1299 *Min. Knowl. Discov.* 18 (2) (2009) 300–336.
- 1300 [16] S. Subramanian, L. Inozemtseva, R. Holmes, Live API documentation, in:
1301 ICSE '14, ACM, 2014, pp. 643–652.
- 1302 [17] C. Treude, M. P. Robillard, Augmenting api documentation with insights
1303 from stack overflow, in: Proceedings of the 38th International Conference
1304 on Software Engineering, ICSE '16, ACM, New York, NY, USA, 2016, pp.
1305 392–403.
- 1306 [18] F. Basciani, J. D. Rocco, D. D. Ruscio, L. Iovino, A. Pierantonio, Auto-
1307 mated clustering of metamodel repositories, in: CAiSE, Vol. 9694 of Lecture
1308 Notes in Computer Science, Springer, 2016, pp. 342–358.
- 1309 [19] A. Pescador, A. Garmendia, E. Guerra, J. S. Cuadrado, J. de Lara, Pattern-
1310 based development of domain-specific modelling languages, in: MODELS,
1311 2015, pp. 166–175.
- 1312 [20] K. Czarnecki, M. Antkiewicz, Mapping features to models: A template ap-
1313 proach based on superimposed variants, in: GPCE, Springer-Verlag, Berlin,
1314 Heidelberg, 2005, pp. 422–437.

- 1315 [21] A. Polzer, D. Merschen, G. Botterweck, A. Pleuss, J. Thomas, B. Hedenetz,
1316 S. Kowalewski, Managing complexity and variability of a model-based em-
1317 bedded software product line, *Innovations in Systems and Software Engi-
1318 neering* 8 (1) (2012) 35–49.
- 1319 [22] J. S. Cuadrado, E. Guerra, J. de Lara, A component model for model trans-
1320 formations, *IEEE Transactions on Software Engineering* 40 (11) (2014)
1321 1042–1060.
- 1322 [23] N. Moha, Y. Guéhéneuc, L. Duchien, A. L. Meur, DECOR: A method
1323 for the specification and detection of code and design smells, *IEEE Trans.
1324 Software Eng.* 36 (1) (2010) 20–36.
- 1325 [24] D. Aguilera, C. Gómez, A. Olivé, Enforcement of conceptual schema quality
1326 issues in current integrated development environments, in: *Proc. CAiSE*,
1327 Vol. 7908 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 626–
1328 640.
- 1329 [25] I. Kurtev, M. Aksit, J. Bézivin, Technical Spaces: An Initial Appraisal, in:
1330 *Proc. of CoopIS*, 2002.
- 1331 [26] C. Atkinson, T. Kühne, Reducing accidental complexity in domain models,
1332 *Software and System Modeling* 7 (3) (2008) 345–359.
- 1333 [27] C. Atkinson, B. Kennel, B. Goß, *The Level-Agnostic Modeling Language*,
1334 Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 266–275.
- 1335 [28] J. de Lara, E. Guerra, *A Posteriori* typing for model-driven engineering:
1336 Concepts, analysis, and applications, *ACM Trans. Softw. Eng. Methodol.*
1337 25 (4) (2017) 31:1–31:60.
- 1338 [29] OMG, SMOF 1.0, <http://www.omg.org/spec/SMOF/1.0/> (2013).
- 1339 [30] Z. Diskin, S. Kokaly, T. Maibaum, Mapping-aware megamodeling: Design
1340 patterns and laws, in: *Proc. SLE*, Vol. 8225 of *Lecture Notes in Computer
1341 Science*, Springer, 2013, pp. 322–343.
- 1342 [31] J. de Lara, E. Guerra, R. Cobos, J. Moreno-Llorena, Extending deep meta-
1343 modelling for practical model-driven engineering, *Comput. J.* 57 (1) (2014)
1344 36–58.
- 1345 [32] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, *EMF: Eclipse Mod-
1346 eling Framework*, Addison-Wesley, 2008.
- 1347 [33] OMG, MOF 2.5.1, <http://www.omg.org/spec/MOF/2.5.1/> (2016).
- 1348 [34] D. Lucrédio, R. P. de Mattos Fortes, J. Whittle, MOOGLE: a metamodel-
1349 based model search engine, *Software and System Modeling* 11 (2) (2012)
1350 183–208.
- 1351 [35] OCL 2.4. specification, <http://www.omg.org/spec/OCL/>.

- 1352 [36] M. F. Porter, An algorithm for suffix stripping, *Program* 40 (3) (2006)
1353 211–218.
- 1354 [37] M. Lesk, Automatic sense disambiguation using machine readable dictio-
1355 naries: how to tell a pine cone from an ice cream cone, in: *SIGDOC*, ACM,
1356 1986, pp. 24–26.
- 1357 [38] G. A. Miller, Wordnet: A lexical database for english, *Comm. ACM* 38 (11)
1358 (1995) 39–41.
- 1359 [39] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi,
1360 M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, I. Horrocks, Capturing
1361 industrial information models with ontologies and constraints, in: *Proc*
1362 *ISWC*, Part II, Vol. 9982 of *Lecture Notes in Computer Science*, 2016, pp.
1363 325–343.
- 1364 [40] B. Motik, I. Horrocks, U. Sattler, Adding integrity constraints to OWL, in:
1365 *Proc. OWLED*, Vol. 258 of *CEUR Workshop Proceedings*, CEUR-WS.org,
1366 2007.
1367 URL <http://ceur-ws.org/Vol-258>
- 1368 [41] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, D. Launay, Neo4emf, A
1369 scalable persistence layer for EMF models, in: *Proc. ECMFA*, Vol. 8569 of
1370 *Lecture Notes in Computer Science*, Springer, 2014, pp. 230–241.
- 1371 [42] P. Neubauer, A. Bergmayr, T. Mayerhofer, J. Troya, M. Wimmer, XML-
1372 Text: From XML Schema to Xtext, in: *Proceedings of SLE*, 2015, pp.
1373 71–76.
- 1374 [43] P. Neubauer, R. Bill, T. Mayerhofer, M. Wimmer, Automated generation of
1375 consistency-achieving model editors, in: *IEEE SANER*, 2017, pp. 127–137.
- 1376 [44] P. Neubauer, R. Bill, M. Wimmer, Modernizing domain-specific languages
1377 with xmltext and intelledit, in: *IEEE SANER*, 2017, pp. 565–566.
- 1378 [45] Eclipse Graphical Editing Framework, <https://eclipse.org/gef/>.
- 1379 [46] P. Runeson, M. Host, A. Rainer, B. Regnell, *Case Study Research in Soft-*
1380 *ware Engineering: Guidelines and Examples*, 1st Edition, Wiley Publishing,
1381 2012.
- 1382 [47] OMG Business Process Model and Notation, <http://www.bpmn.org/>.
- 1383 [48] J. Mylopoulos, *Characterizing Information Modeling Techniques*, Springer
1384 Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 17–57.
- 1385 [49] H. Kern, A. Hummel, S. Kühne, Towards a comparative analysis of
1386 meta-metamodels, in: *Proceedings of the compilation of the co-located*
1387 *workshops, DSM’11, TMC’11, AGERE!’11, AOOPEs’11, NEAT’11, and*
1388 *VMIL’11*, Portland, OR, USA, October 22 - 27, 2011, 2011, pp. 7–12.

- 1389 [50] P. Buneman, Semistructured data, in: Proceedings of the Sixteenth ACM
1390 SIGACT-SIGMOD-SIGART Symposium on Principles of Database Sys-
1391 tems, ACM Press, 1997, pp. 117–121.
- 1392 [51] A. Olivé, Conceptual modeling of information systems, Springer, 2007.
1393 doi:10.1007/978-3-540-39390-0.
- 1394 [52] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web: From Relations to
1395 Semistructured Data and XML, Morgan Kaufmann, 1999.
- 1396 [53] C. Atkinson, T. Kühne, Profiles in a strict metamodeling framework, Sci.
1397 Comput. Program. 44 (1) (2002) 5–22.
- 1398 [54] T. Kühne, Matters of (meta-)modeling, Software and System Modeling
1399 5 (4) (2006) 369–385.
- 1400 [55] Shapes Constraint Language (SHACL), [https://w3c.github.io/
1401 data-shapes/shacl/](https://w3c.github.io/data-shapes/shacl/).
- 1402 [56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, Experimentation in Software Engineering, Springer, 2012.
- 1404 [57] G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Improving memory efficiency
1405 for processing large-scale models, in: BigMDE, University of York, York,
1406 UK, United Kingdom, 2014.
1407 URL <https://hal.inria.fr/hal-01033188>
- 1408 [58] K. Barmpis, D. S. Kolovos, Towards scalable querying of large-scale models,
1409 in: ECMFA, Vol. 8569 of Lecture Notes in Computer Science, Springer,
1410 2014, pp. 35–50.
- 1411 [59] A. Dyck, A. Ganser, H. Lichter, Enabling model recommenders for
1412 command-enabled editors, in: MDEBE, 2013, pp. 12–21.
- 1413 [60] A. Dyck, A. Ganser, H. Lichter, A framework for model recommenders -
1414 requirements, architecture and tool support, in: MODELSWARD, 2014,
1415 pp. 282–290.
- 1416 [61] A. Dyck, A. Ganser, H. Lichter, On designing recommenders for graphical
1417 domain modeling environments, in: MODELSWARD, 2014, pp. 291–299.
- 1418 [62] S. Sen, B. Baudry, H. Vangheluwe, Towards domain-specific model editors
1419 with automatic model completion, Simulation 86 (2) (2010) 109–126.
- 1420 [63] T. Walter, F. S. Parreiras, S. Staab, An ontology-based framework for
1421 domain-specific modeling, Software and Systems Modeling 13 (1) (2014)
1422 83–108.
- 1423 [64] R. Mendieta, J. L. de la Vara, J. Llorens, J. Álvarez-Rodríguez, Towards
1424 effective sysml model reuse, in: Proc. MODELSWARD, SCITEPRESS,
1425 2017, pp. 536–541.

- 1426 [65] R. F. Paige, D. S. Kolovos, L. M. Rose, N. Drivalos, F. A. C. Polack, The
1427 design of a conceptual framework and technical infrastructure for model
1428 management language engineering, in: ICECCS, IEEE Computer Society,
1429 2009, pp. 162–171.
- 1430 [66] E. Hajiyev, M. Verbaere, O. de Moor, K. D. Volder, Codequest: querying
1431 source code with datalog, in: Proc. OOPSLA 2005, ACM, 2005, pp. 102–
1432 103.
- 1433 [67] H. Kern, F. Stefan, V. Dimitrieski, M. Ćeliković, Mapping-based exchange
1434 of models between meta-modeling tools, in: DSM, ACM, New York, NY,
1435 USA, 2014, pp. 29–34.
- 1436 [68] V. Dimitrieski, M. Ćeliković, N. Igić, H. Kern, F. Stefan, Reuse of rules
1437 in a mapping-based integration tool, in: SoMet, Springer International
1438 Publishing, Cham, 2015, pp. 269–281.
- 1439 [69] R. Tairas, M. Mernik, J. Gray, Using ontologies in the domain analysis of
1440 domain-specific languages, in: TWOMDE, 2008, pp. 20–31.
- 1441 [70] I. Ceh, M. Crepinsek, T. Kosar, M. Mernik, Ontology driven development
1442 of domain-specific languages, *Comput. Sci. Inf. Syst.* 8 (2) (2011) 317–342.
- 1443 [71] A. Ojamaa, H. Haav, J. Penjam, Semi-automated generation of DSL meta
1444 models from formal domain ontologies, in: MEDI, 2015, pp. 3–15.
- 1445 [72] D. Gasevic, D. Djuric, V. Devedzic, *Model Driven Engineering and Ontol-
1446 ogy Development*, Springer, 2009.
- 1447 [73] T. Walter, F. S. Parreiras, G. Gröner, C. Wende, OWLizing: Transforming
1448 Software Models to Ontologies, in: ODISE, 2010, pp. 7:1–7:6.
- 1449 [74] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Rets-
1450 chitzegger, W. Schwinger, M. Wimmer, Lifting Metamodels to Ontologies:
1451 A Step to the Semantic Integration of Modeling Languages, in: MoDELS,
1452 2006, pp. 528–542.
- 1453 [75] M. Milanovic, D. Gasevic, A. Giurca, G. Wagner, V. Devedzic, Towards
1454 Sharing Rules Between OWL/SWRL and UML/OCL, ECEASST 5.
- 1455 [76] F. S. Parreiras, S. Staab, A. Winter, On marrying ontological and meta-
1456 modeling technical spaces, in: FSE, 2007, pp. 439–448.
- 1457 [77] F. S. Parreiras, S. Staab, Using ontologies with UML class-based modeling:
1458 The TwoUse approach, *DKE* 69 (11) (2010) 1194–1207.
- 1459 [78] D. Djuric, D. Gasevic, V. Devedzic, V. Damjanovic, A UML Profile for
1460 OWL Ontologies, in: MDAFA, 2004, pp. 204–219.
- 1461 [79] EMF Query, <https://projects.eclipse.org/projects/modeling.emf.query>.

- 1462 [80] B. Bislimovska, A. Bozzon, M. Brambilla, P. Fraternali, Textual and
1463 content-based search in repositories of web application models, *TWEB* 8 (2)
1464 (2014) 1–11.
- 1465 [81] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, J. Mendling,
1466 Similarity of business process models: Metrics and evaluation, *Inf. Syst.*
1467 36 (2) (2011) 498–516.
- 1468 [82] J. D. Rocco, D. D. Ruscio, L. Iovino, A. Pierantonio, Collaborative repos-
1469 itories in model-driven engineering, *IEEE Software* 32 (3) (2015) 28–34.
- 1470 [83] M. Gasparic, G. C. Murphy, F. Ricci, A context model for ide-based recom-
1471 mendation systems, *Journal of Systems and Software* 128 (2017) 200–219.

1472 **Appendix: List of queries**

1473 Table 9 shows the list of pre-defined queries of EXTREMO, which were de-
 1474 fined by implementing the provided extension points. The user may provide ad-
 1475 ditional queries by implementing the extension point (in Java). Some of these
 1476 queries come from catalogues of accepted quality criteria in conceptual mod-
 1477 elling [24]. We divide them into predicate and custom, and depict the element
 1478 they inspect (filterBy).

SearchConfiguration	filterBy	Description
Predicate Based Search		
A NamedElement has a name	NamedElement	Checks if a NamedElement object has a name that matches with a value. Options: name: PrimitiveTypeParam typed as string
All instances of a NamedElement	NamedElement	Returns all the NamedElements that are instances of another one. Options: type: ModelTypeParam and recursive: PrimitiveTypeParam typed as boolean
A node with a property with value X	SemanticNode	Checks if a SemanticNode object has a data property which has a concrete value. Options: value: PrimitiveTypeParam typed as string
A node has more than a number of parents	SemanticNode	Checks if a SemanticNode object has more than a number of supers instances. Options: parents: PrimitiveTypeParam typed as int
A node has more than a number of children	SemanticNode	Checks if a SemanticNode object has more than a number of subs instances. Options: children: PrimitiveTypeParam typed as int
Attributes are overloaded	SemanticNode	Checks if a SemanticNode object contains more than a number of DataProperties. Options: maxattrs: PrimitiveTypeParam typed as int
References are overloaded	SemanticNode	Checks if a n: SemanticNode object contains more than a number of ObjectProperties. Options: maxrefs: PrimitiveTypeParam typed as int
An abstract node without children	SemanticNode	Checks if a SemanticNode object is abstract and there are no supers instances.
An abstract node with an unique child	SemanticNode	Checks if a SemanticNode object is abstract and it has only a supers instance.
Data Properties Value	DataProperty	Checks if a DataProperty object has a concrete value. Options: valuefield: PrimitiveTypeParam typed as string
Data Properties Value Range	DataProperty	Checks if the integer value of a DataProperty object ranges between a minimum and a maximum. Options: minvaluefield: PrimitiveTypeParam typed as int and maxvaluefield: PrimitiveTypeParam typed as int
Custom Search		
Nodes without descriptions	Resource	For a resource, split the nodes in two groups. The first one refers to the nodes with descriptions are the second one refers to the nodes without descriptions. Options: resource: ModelTypeParam typed as Resource
Isolated nodes	Resource	Checks if a resource contains nodes that are isolated. Options: resource: ModelTypeParam typed as Resource
A hierarchy is too deep	Resource	Checks if a node is too deep on a level of hierarchy. Options: maxdepth: PrimitiveTypeParam typed as int
No node is referred from too many others	Resource	In a resource, checks if there is a node that is referred from too many others. Options: maxrefs: PrimitiveTypeParam typed as int and resource: ModelTypeParam typed as Resource
Hierarchy Splitter	Resource	For a resource, split the nodes in groups. In every group a inheritance hierarchy is left. Options: resource: ModelTypeParam typed as Resource

Table 9: List of simple search configurations (queries)