Extremo:
An Eclipse plugin for modelling and meta-modelling assistance

Ángel Mora Segura[a], Juan de Lara[a]

[a]*Modelling & Software Engineering Research Group*
http://miso.es
Universidad Autónoma de Madrid (Spain)

Abstract

Modelling is a core activity in software development paradigms like Model-driven Engineering (MDE). Therefore, the quality of (meta-)models is crucial for the success of software projects. However, many times, modelling becomes a purely manual activity, which does not take advantage of information embedded in heterogeneous information sources, such as XML documents, ontologies, or other models and meta-models.

In order to improve this situation, we present Extremo, an Eclipse plugin aimed at gathering the information stored in heterogeneous sources in a common data model, to facilitate the reuse of information chunks in the model being built. The tool covers the steps needed to incorporate this knowledge within an external modelling tool, supporting the uniform query of the heterogeneous sources and the evaluation of constraints. Flexibility of the main features (e.g., supported data formats, queries) is achieved by means of extensible mechanisms. To illustrate the usefulness of Extremo, we describe a practical case study in the financial domain and evaluate its performance and scalability.

Keywords: Model-driven Engineering, Modelling process, Language Engineering, Modelling assistance

1. Introduction

This paper presents Extremo, an extensible assistant for modelling and meta-modelling. The main goal of the tool is to facilitate modelling (at any

Email addresses: Angel.MoraS@uam.es (Ángel Mora Segura), Juan.deLara@uam.es (Juan de Lara) meta-level) by enabling the reuse of heterogeneous information sources. For this purpose, Extremo provides flexible features to represent heterogeneous information in a uniform way, its querying, and its validation through constraint evaluation. The tool has been realized as an Eclipse plugin, and has been designed to be easily integrable with external modelling and meta-modelling plugins. This integration facilitates the reuse of the information chunks returned by queries using Extremo into a (meta-)model under construction.

The rest of this *Original Software Publication* paper is organized as follows. Section 2 motivates the need for modelling assistants like Extremo, explores their design alternatives and positions our approach with respect to related works based on a feature model. Section 3 gives an overview of Extremo, including its functionalities and architecture. Section 4 details its implementation. Section 5 provides a case study in the financial domain. Section 6 presents experiments evaluating performance and scalability. Finally, Section 7 draws conclusions and proposes lines for future work.

2. Motivation

Modelling is a core activity in Software Engineering processes, and essential in development paradigms like Model-driven Engineering (MDE) [1]. In MDE, models are actively used to automate the different phases of the development process. They are described using a modelling language, which is itself defined through a meta-model. Thus, modelling and meta-modelling are recurring activities in MDE, and the quality of the (meta-)models is crucial to ensure the success of MDE projects.

Many times, models in MDE are domain-specific, describing systems using the vocabulary and concepts of a highly specialized knowledge area, like logistics, finance, nuclear physics or marketing. Therefore, building domain-specific models and meta-models often requires deep domain expertise, which the engineers building the models may lack. Many times, useful information for a domain is scattered across heterogeneous resources, like ontologies, XML documents, CSV files or other models and meta-models. However, modelling tasks rarely take advantage of the knowledge embedded in these heterogeneous sources, and modelling tools often lack automated support to master this diversity [2].

Most programming IDEs provide assistants to facilitate code completion or search [3, 4]. However, nowadays automated assistance is not the norm within modelling environments. In the light of the above, next we characterize the design space for *modelling assistants* using a feature model [5],

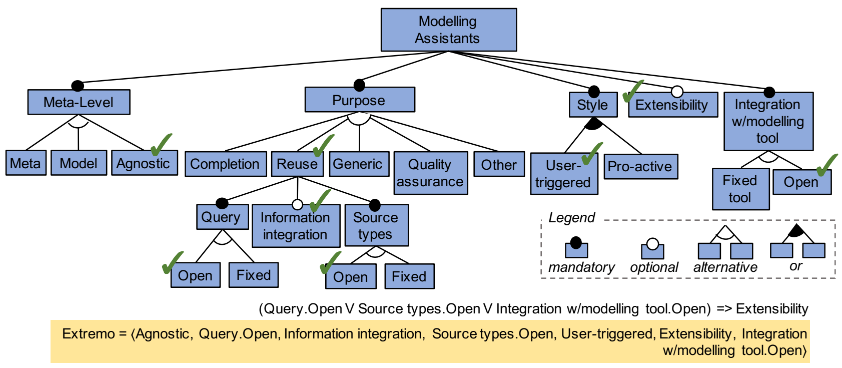

Modelling Assistants
Meta-Level
Meta
Model
Agnostic
Purpose
Completion
Reuse
Generic
Quality assurance
Other
Query
Open
Fixed
Information integration
Source types
Open
Fixed
Style
User-triggered
Pro-active
Extensibility
Integration w/modelling tool
Fixed tool
Open
Legend
mandatory
optional
alternative
or
(Query.Open V Source types.Open V Integration w/modelling tool.Open) => Extensibility
Extremo = (Agnostic, Query.Open, Information integration, Source types.Open, User-triggered, Extensibility, Integration w/modelling tool.Open)

Figure 1: Feature model characterizing modelling assistants.

shown in Figure 1. The Figure also shows the configuration chosen to design Extremo (using ticks on the model features).

Firstly (feature meta-level), assistants can be directed to help creating certain types of models (e.g., state-machines), work at the meta-model level, or be meta-level independent. The purpose of assistants can also vary. For example, modelling assistants have been proposed to complete models with respect to meta-models [6], to evaluate model quality and suggest improvements [7], generic modelling assistant frameworks [8], or for information reuse [2], among others. The assistant may need to be invoked by the user, or can be pro-active (feature style), can be extensible with new capabilities, and may be specific to a particular modelling tool, or integrable with other modelling tools.

Regarding assistants for information reuse (feature reuse), some tools extract information from specific data representations. For example, methods have been proposed to develop meta-models based on ontologies [9], or to search repositories of EMF models [10]. However, in practice, being tied to a specific technology is too restrictive, preventing modellers to access information represented in arbitrary formats. Moreover, some of these tools just provide search services, while developers would benefit from mechanisms to facilitate integrating the query results into the model being built (feature Information integration). The supported queries can be fixed, or the user may be able to provide specific queries by means of extensible mechanisms.

Extremo fills the gap of the lack of modelling assistants directed to information reuse [2]. It is level-agnostic, and so it can be used to create models at any meta-level. It is based on representing heterogeneous data using a common data model, which can be queried in a uniform way. Extremo achieves flexibility by means of extensible mechanisms, which allow defining new types of queries, adding support for new data formats, and integration with external modelling tools to achieve true information reuse.

3. Software Framework

Next, we detail the main functionalities of Extremo in Section 3.1 and describe its architecture in Section 3.2.

3.1. Software Features

In the following we describe the main functionalities of Extremo, based on the feature model presented in Figure 1.

Handling heterogeneous information sources. Extremo relies on a common data model, whose meta-model is shown in Figure 2. The model organizes (possibly nested) resources into repositories. Resources contain SemanticNodes, with both attributes (DataProperty) and references (ObjectProperty). Our model reifies the instantiation relation (describes/descriptors), which permits representing uniformly both models and meta-models, classes and objects, and attributes and slots, leading to simplicity and generality [11]. As the instantiation relation is not restricted to a fixed number of meta-levels, Extremo becomes *level agnostic*. This way, the data model can store information of both types and instances, which can be reused to build models or meta-models. We refer to [2] for further details on the data model.

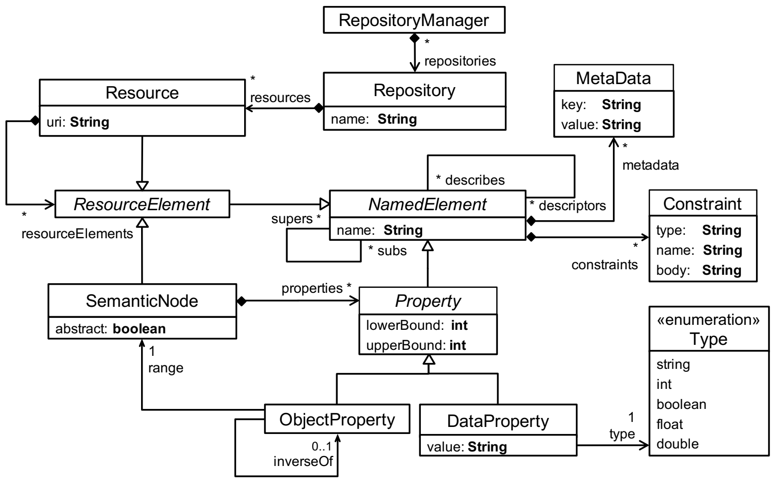


Figure 2: Extremo's common data model.

Flexible query. In order to perform queries over model elements conforming to many different schemas, the query mechanism supports the flexible exploration of the information gathered. Queries can be *atomic* or *composite* (made of other atomic or composite queries). The results of a composite query are formed by the combination of the results of the atomic queries. Atomic queries may follow two styles: *predicate*-based or *custom*. In the former, the iteration is driven by the engine, and the user provides a predicate that is evaluated on each element of the data model. In the latter, the user controls the iteration and has more flexibility on how outputs are presented. In addition, queries can specify input parameters to be filled by the user (e.g., like the name of a node to be sought). Depending on the parameter type (string, integer, etc.) the user may select services to be used (e.g., inexact string match or synonym-based search for string parameters), which are also extensible.

EXTREMO offers a pre-defined catalogue of query types, which includes queries to search for objects or properties having a certain name, all instances of a given NamedElement, nodes with properties having certain values, and untyped nodes, among many others. However, as specific domains may need from customized query types (e.g., finding all safe Petri net models [12]), EXTREMO offers extension mechanisms to define new query types and services for data types.

Extensibility. The extensibility mechanism is realized through Eclipse extension points serving abstract classes and enforcing the extensibility mechanism through *polymorphism*. EXTREMO offers extension points to support new data formats, types of queries, types of constraints, and for the integration with external modelling tools, among others.

The user can add support for a new data format by specifying a mapping from the format-specific structure to the common data model. To ensure correctness, EXTREMO provides a mechanism for the evaluation of heterogeneous types of constraints (e.g., OCL, XML schema restrictions). New types of constraints, associated to specific formats, can be added through extension points. To enable true reuse, the tool offers mechanisms to select elements from the data model and integrate them into models being built with external tools, as next item explains.

Integration with a modelling tool. The framework has been designed to flexibly integrate the information stored in the common data model with external modelling tools (typically, other Eclipse plugins). We currently support a reactive integration mode where the assistant needs to be explicitly invoked. The integration mechanism can be based either on menu commands or via drag and drop operations (for editors based on the Graphical Editing Framework [13]). Both integration modes provide the mapping between the common data model and the specific model representation (e.g., EMF, UML) in the external modelling tool.

3.2. Software Architecture

Figure 3 overviews the architecture of EXTREMO. It is an Eclipse plugin designed to maintain the separation of concerns between data import (converting the heterogeneous data into the common data model), the model extension, the definition of querying mechanisms, the reuse of query operations, the model persistence and the assistance during the modelling process.

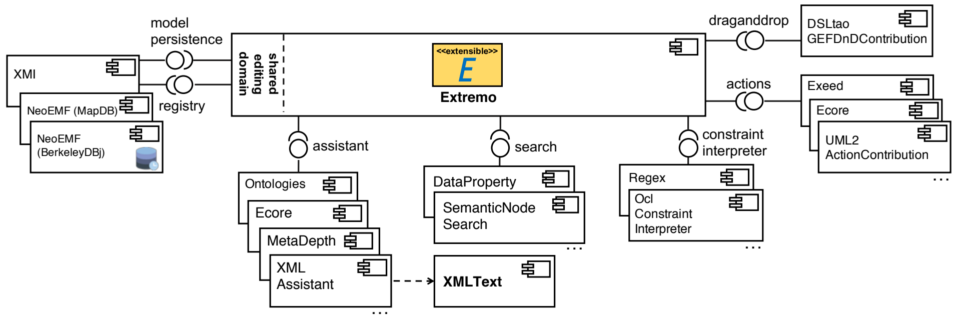
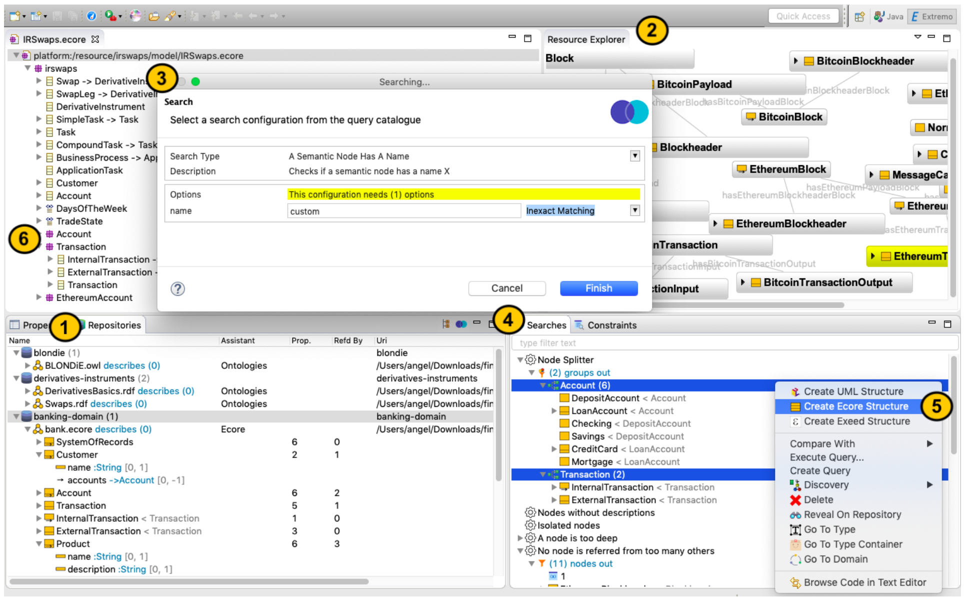


Figure 3: Architecture of EXTREMO.

First, we made available an extension point so that assistants handling specific data formats can be added. EXTREMO provides a framework for their creation, which permits their conceptual organization as model-to-model transformations (from the data format to the common data model). Hence, we provide an abstract class with a number of methods, which need to be overridden for the particular assistant, and act as rules in the transformation. EXTREMO comes with a number of predefined assistants supporting a variety of technical spaces, including the modelling space (in particular the Eclipse Modelling Framework (EMF) [14]), ontologies and XML [15]. The tool provides an extensible mechanism for the flexible query of the resources (search) and the evaluation of heterogeneous constraints (constraint interpreter). In addition, we enable the integration with a modelling tool by the addition of different items (actions or draganddrop) to the Eclipse IDE GUI. Finally, we provide extensions for different model persistence options to store the data model, including XMI and NoSQL datastores [16].

4. Implementation

Figure 4 shows EXTREMO in action. In particular, it displays the set of extensible features applied to the example we will present in Section 5.



IRSwaps.ecore

```
▼ platform:/resource/irswaps/model/IRSwaps.ecore
  ▼ irswaps
    ▶ Swap -> DerivativeIn
    ▶ SwapLeg -> Derivativel
      DerivativeInstrument
    ▶ SimpleTask -> Task
    ▶ Task
    ▶ CompoundTask -> Task
    ▶ BusinessProcess -> Ap
      ApplicationTask
    ▶ Customer
    ▶ Account
    ▶ DaysOfTheWeek
    ▶ TradeState
      Account
      Transaction
      ▶ InternalTransaction -
      ▶ ExternalTransaction -
      ▶ Transaction
    ▶ EthereumAccount
```

Searching...

Search

Select a search configuration from the query catalogue

Search Type	A Semantic Node Has A Name
Description	Checks if a semantic node has a name X
Options	This configuration needs (1) options
name	custom — Inexact Matching

Cancel — Finish

Repositories

Name	Assistant	Prop.	Refd By	Uri
▼ blondie (1)				blondie
▶ BLONDiE.owl describes (0)	Ontologies			/Users/angel/Downloads/fin
▼ derivatives-instruments (2)				derivatives-instruments
▶ DerivativesBasics.rdf describes (0)	Ontologies			/Users/angel/Downloads/fin
▶ Swaps.rdf describes (0)	Ontologies			/Users/angel/Downloads/fin
▼ banking-domain (1)				banking-domain
▼ bank.ecore describes (0)	Ecore			/Users/angel/Downloads/fin
▶ SystemOfRecords		6	0	
▼ Customer		2	1	
name :String [0, 1]				
→ accounts ->Account [0, -1]				
▶ Account		6	2	
▶ Transaction		5	1	
▶ InternalTransaction < Transaction		1	0	
▶ ExternalTransaction < Transaction		3	0	
▼ Product		6	3	
name :String [0, 1]				
description :String [0, 1]				

Searches — Constraints

```
type filter text
▼ Node Splitter
  ▼ (2) groups out
    ▼ Account (6)
        DepositAccount < Account
      ▶ LoanAccount < Account
        Checking < DepositAccount
        Savings < DepositAccount
      ▶ CreditCard < LoanAccount
        Mortgage < LoanAccount
    ▼ Transaction (2)
      ▶ InternalTransaction < Transaction
      ▶ ExternalTransaction < Transaction
  Nodes without descriptions
  Isolated nodes
▶ A node is too deep
▼ No node is referred from too many others
  ▼ (11) nodes out
      1
```

Figure 4: EXTREMO in action.

EXTREMO is an extensible Eclipse plugin that offers several views. The repository view (label 1) contains the information stored in the common data model, hence displaying the information coming from heterogeneous data sources. In the Figure, the view shows three repositories, with the resources they include and the assistant used for the import (Ontologies, Ecore). The contents of each resource can be visualized in this view, or graphically in the resource explorer (label 2). The latter view is based on Zest[1], a component for graph-based visualization.

Resources and repositories can be queried through a Search Wizard (the dialog window with label 3). This wizard permits selecting a query type from the catalogue, input required parameters and selecting appropriate services. The results of the query are displayed in the results view (label 4). This view organizes the results by query types, supports browsing the results, and incorporating chunks into the model being built. The latter

[1]https://www.eclipse.org/gef/zest/

action is done through a contextual menu, depicted with label 5 in the Figure. As the three first commands on the menu show, we have integrated Extremo with a UML model editor[2], the standard Ecore editor and Exeed, an enhanced version of the built-in EMF reflective tree-based editor[3]. In the Figure the information is being reused to create a meta-model using the Ecore tree editor (label 6). The Extremo distribution includes these three predefined action-based integrations, and a drag-and-drop integration with DSL-tao [17], but additional tools can be integrated using the Extremo extension points. The contextual menu with label 5 also includes commands to locate the query results in the repository, or to navigate to the node type, resource type or repository of a node. Finally, the constraint evaluation view (label 4, in background) supports checking the validity of the imported data.

Extremo is open-source and published under the Eclipse Public Licence[4]. More information can be found in this paper's metadata or in the project Wiki[5].

5. Example

Next, we illustrate the main features of Extremo to create a domain-specific language (DSL) for the financial domain. We chose this case study for being representative of a highly specialized domain, for which a meta-modelling expert may lack knowledge and need assistance. Hence, a tool like Extremo would be useful in order to obtain a more complete and accurate meta-model.

Description of the problem. The scope of the language is to cover the process involved in a *Plain Vanilla Swap*, the simplest version of an *Interest Rate Swap (IR Swap)* between companies. It is inspired by a well-known case study defined by the EDM Council[6]. Swaps are often used if a company wants to borrow money to finance itself at one type of interest rate (for example, fixed), but the lender prefers to offer a floating-rate loan because it is a better deal for itself. Then, the company decides to borrow at the floating rate and make a separate deal to obtain the fixed rate. The other

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

[3]Epsilon Exeed, `http://www.eclipse.org/epsilon/`

[4]`https://www.eclipse.org/legal/epl-2.0/`

[5]`https://github.com/angel539/extremo/wiki`

[6]`https://edmcouncil.org/`

deal is usually made with another bank, a facilitator firm or another company directly. The complete DSL also involves the execution of the swaps on a blockchain-based ecosystem, but we omit this part for simplicity.

Expected outcome. We aim to produce a meta-model for a DSL supporting models like the ones shown in Listing 1 for the *contractual terms* and Listing 2 for the *entities involved* in the process. The swap in Listing 1 defines two parts (also called legs) involved in the swapping process. These must specify some required information for establishing the swap, all of them gathered within the transfer entities in lines 5 and 9. The parameters include the accounts (identified by acc1 and acc2), the notional principal amount (50000 and 60000), the start year (2019), the schedule for the payments and other parameters. Listing 2 shows an excerpt of the account information required for the parties involved.

```
 1 swap InsuranceSwap{
 2   legs = [
 3     leg AustriaLeg{
 4       part = "Vienna Ins. Group",
 5       transfer(acc1, 50000, "2019", YEARLY, ...)
 6     },
 7     leg SpainLeg{
 8       part = "Allianz Seguros",
 9       transfer(acc2, 60000, "2019", YEARLY, ...)
10     }
11   ]
12 }
```

Listing 1: Contractual terms of a Swap.

```
1 customer "Vienna Ins. Group"{
2   accounts = [
3     account acc1{
4       number = "ABCD1234",
5       ...
6     },...
7   ]
8 }
9 customer "Allianz Seguros"{...}
```

Listing 2: Entities involved.

Procedure. An engineer with lack of knowledge in the financial domain may produce a meta-model with many omissions. Hence, this is a typical scenario where an assistant like EXTREMO becomes particularly helpful to complement a modelling tool. This way, we will use EXTREMO to build the meta-model for the DSL and reuse information from other useful information sources. This will be done following the steps shown in Figure 5:

1. *Identify heterogeneous information sources.* In a first step, we need to identify useful resources with relevant information for the domain. For the running example, we took: (*i*) resources from the Finance Domain Task Force of the OMG[7], specifically the FIBO collection, which includes

[7]https://www.omg.org/fdtf/projects.htm

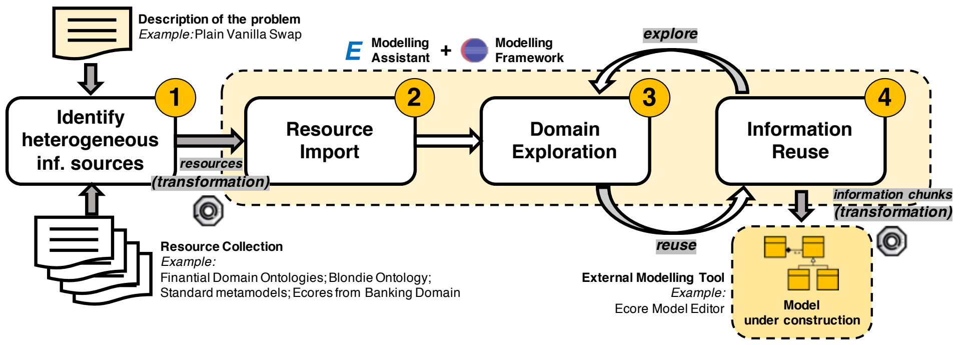


Figure 5: Steps for using EXTREMO as a (meta-)modelling assistant.

information on financial entities in OWL and RDF formats; (*ii*) *Blondie*, the Blockchain Ontology with Dynamic Extensibility[8], which contains information describing the interchange operation; (*iii*) Ecore files available on the OMG repository[9] with standard meta-models, such as BMM, BPEL or BPMN; and (*iv*) Ecore files available in repositories like gitHub and the ATL ecore zoo[10] with concepts of the banking domain.

2. *Resource Import.* In this step, the identified resources are imported into EXTREMO's common data model. This is automatically performed by the assistants of the given formats (the EcoreAssistant and the OntologyAssistant in this case). When a resource is imported, the assistant resolves the mapping between the particular format and the common data model. After the resources are imported, they are displayed in EXTREMO's repository view (label 1 in Figure 4), and are ready to be explored, queried and reused.

3. *Domain Exploration.* Once the resource collection has been imported, we can explore it using the queries and features presented in Section 4. For example, in a first step, it might be useful to obtain a high-level view of the resource content, by using the Nodes Splitter query, which splits the entities contained in the resources into inheritance hierarchies. Additionally, we can use the Resource Explorer to have a perspective of the entities contained in a resource (Figure 4, label 2). Finally, we might want

[8]`https://hedugaro.github.io/Linked-Blockchain-Data/`

[9]The OMG is the standarization body behind many modelling standards such as UML, SysML, MOF or BPMN. (`http://www.omg.org/spec/`)

[10]`http://web.imt-atlantique.fr/x-info/atlanmod/index.php?title=Ecore`

to look for existing entities related to e.g., *Swaps* using synonym-based search. The exploration of the domain depends on the level of expertise of the engineer but the final goal is to find useful entities to reuse in the meta-model under construction (Figure 4, labels 3 to 6).

4. *Information Reuse.* Finally, if we find a set of useful entities for our domain, we can add them to the (meta-)model under construction (Figure 4, label 6). Technically, when an information chunk is added to the model under construction, the integration with the modelling tool resolves the mapping between the common data model and the particular modelling technology. In the case of the example, we use the Ecore tree editor. Please note that steps 3 and 4 are iterative.

Result. Figure 6 shows an excerpt of the obtained meta-model we were able to build by reusing information from the chosen *resource collection.* In addition, a video demo showing this construction process can be found in the website of the project[11].

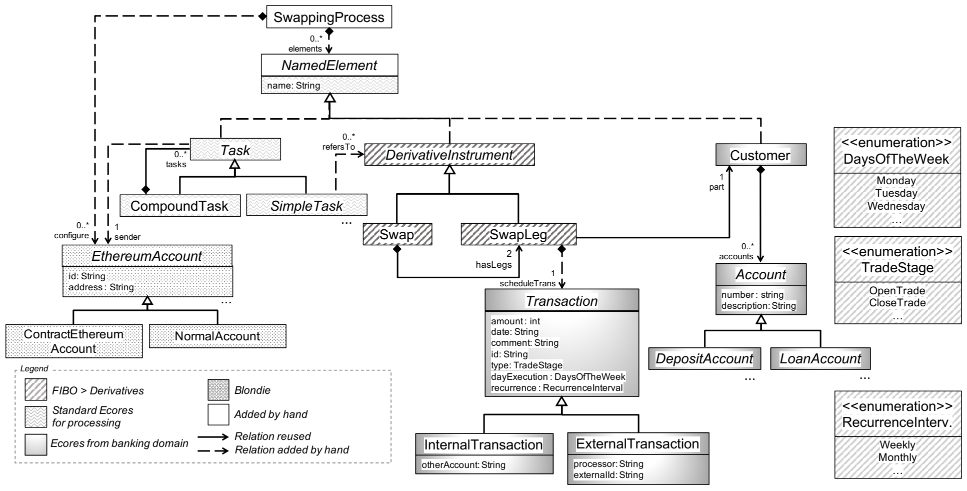


Figure 6: Excerpt of the resulting Plain Vanilla Swap process meta-model.

In detail, the meta-model has elements (such as Swap or SwapLeg) reused from the set of ontologies of the FIBO collection, elements reused from different Ecore meta-models from the banking domain (such as Transaction or Account), elements from Ecore meta-models from the process modelling domain (CompoundTask and Task) and elements (such as EthereumAccount) from

[11]`https://github.com/angel539/extremo/wiki/Case-Studies`

the Blondie ontology to enable the connection. Finally, it contains a set of enumerations created from a list of instances of semantic nodes without descriptions. Overall, roughly *90% of the classes* in our solution have been obtained by combining different representations and concepts. In addition, *most of the generalization relations* could also be reused using the information chunks returned by the queries, as well as *some of the association relations* by the combination of semantic nodes. The rest of the elements in the meta-model were added by hand, as it is usually required. Therefore, the example suggests that Extremo is useful as a help in the construction of a complex meta-model by reusing information from different heterogeneous information sources.

6. Scalability Assessment

In this section we report on two experiments that illustrate the efficiency and scalability of Extremo with respect to resource import and query (steps 2 and 3 of Figure 5).

The experiments were executed on a laptop running MacOS Mojave with an Intel Core i5 processor (1,3 GHz), 8 GB of DDR3 SDRAM (1600MHz), and a PCIe SSD. Extremo was running on Eclipse vMars.2 with Java SE Runtime Environment v1.8. Further details, including the models used in the experiments, screencasts taken during the execution, and the complete results obtained, are available at the website of the project[12].

1. *Resource Import.* We generated two sets of models using AtlanMod's random instantiator[13] with sizes ranging from 10 to 50,000 objects. For each size, we generated five models, and executed each experiment three times. For the first set we used a meta-model describing people, while the second used a meta-model of Petri nets. We imported the resource collection using the EcoreAssistant, which translates the elements from the EMF technical space into our data model, and measured the time taken.

   Figure 7 shows the results obtained. The tool scales well for medium size models, but requires a time in the order of minutes to import models with sizes bigger than around 40,000 objects. However, please note that importing the resource collection is a one-time operation, required only

[12] `https://github.com/angel539/extremo/wiki/Performance-Evaluation`

[13] `https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator`

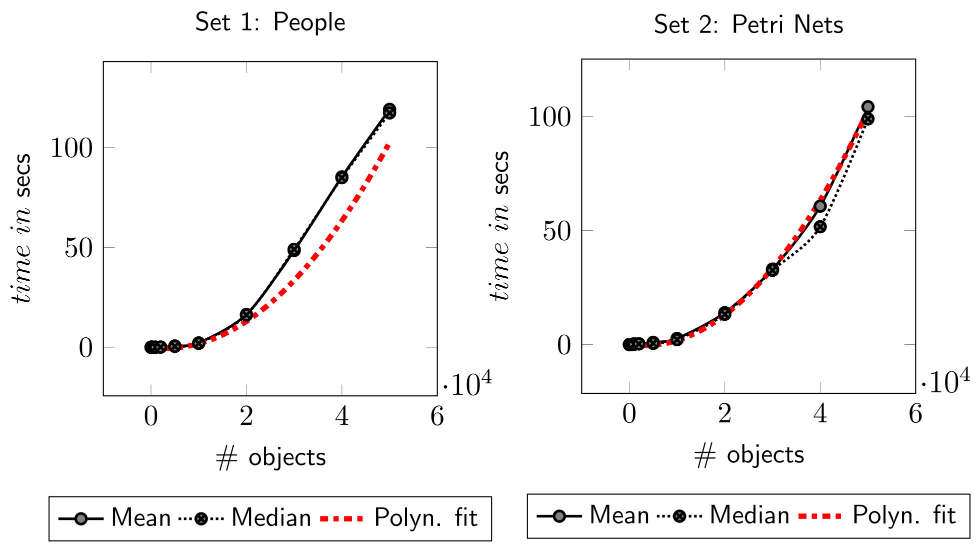


Figure 7: Results for the resource import experiment in secs

when a new data set is to be gathered (i.e., it is not required every time the tool is started). Moreover, many artefacts in the MDE technical space have moderate size. For example meta-models tend to be in the order of tens to hundreds of classes [18, 19].

2. *Domain Exploration: Querying.* Next, we executed a set of queries on the instances of the common data model loaded in the repository. First, we executed a predicate-based search (making the iteration internally, driven by the engine) to force the query to traverse the whole model looking for all objects with a concrete name. We performed this query with and without services. As previously mentioned, services are functions applied to certain data types that enhance the query mechanism, e.g., with synonym search or inexact matching for strings.

   Figure 8 depicts the results obtained, showing good performance and scalability. When the search uses a service bound to a query parameter, the service increments the time consumed by the operation, because the query delegates the resolution of the operation to the service. In the experiment, we relied on the Inexact matching service to perform the comparison of two strings using variations derived from their roots.

In addition, we performed similar experiments with the custom queries presented in our previous work [2]. These queries make the iteration ex-

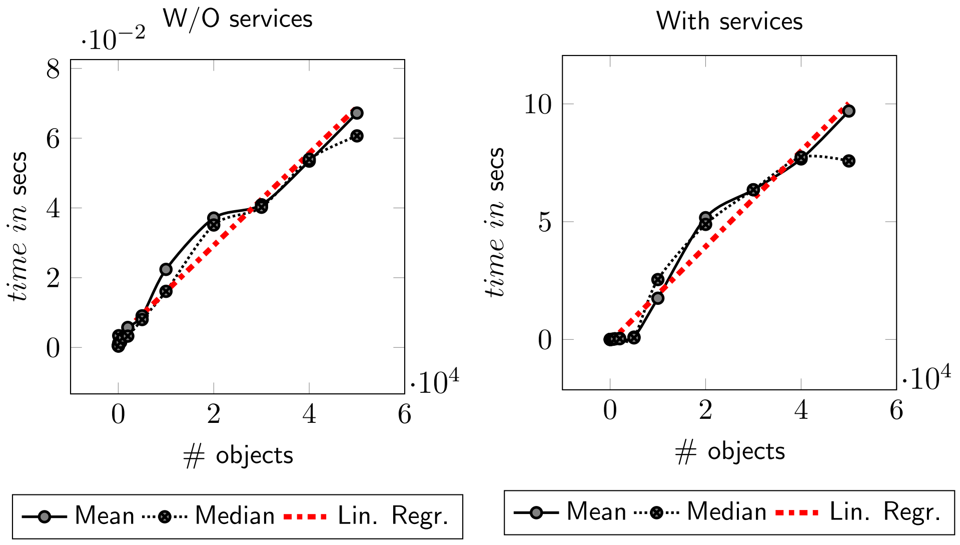


Figure 8: Results for the predicate-based queries without services and with services

ternally (driven by the user code). They receive the repository selected by the user and perform the operation with the possibility of aggregating the results into groups. While predicate-based queries need to traverse the complete resource, custom queries can interrupt the search whenever they find an element of interest. Hence, custom queries present execution times that can range from milliseconds to seconds depending on the query and the structure of the data model. For example, a query locating all isolated nodes takes about 100 seconds on the biggest People model, and less than 2 seconds for the biggest Petri nets model. More details can be found at EXTREMO's website.

6.1. Discussion and Threats to Validity

Overall, from the experiments we can conclude that EXTREMO has good performance for its envisioned usage scenarios. While resource import can take minutes, it is a one-time operation, only required when a new information repository is to be created. Queries are fast, typically in the order of a few seconds even for large models. While these experiments show good results, they were performed using EMF models as data sources. We plan to perform more extensive experiments with data from other technological spaces (CSVs, XML documents, Ontologies). However, as all these technologies are file-based, we expect similar times.

The models used in the experiments were synthetic, and so experiments using realistic models and data sources would be required to confirm these results. We are planning to conduct a systematic experiment, but for the example presented in Section 5, we obtained import times ranging from 15 milliseconds (for BusinessEntityModel.ecore, with a size of $3Kb$) to 58.7 seconds (for Swaps.rdf, with $5.5Mb$ of information). Details on the import times of the example are available on project website[14].

7. Conclusions

MDE relies on domain-specific modelling, which often requires knowledge of very specialized areas. However, modelling is currently performed mostly in a manual way. To improve this situation, we have proposed an extensible modelling assistant, able to gather information from heterogeneous sources in a common data model, which then can be queried in a uniform way. Extremo is designed to be easily integrated with existing modelling and meta-modelling Eclipse plugins, where the query results can be easily incorporated into the model being built. This paper has described the main features of the tool, presented an example in the financial domain, and reported on a performance evaluation, showing good results.

We are currently planning an empirical evaluation with users to analyse the benefits of using Extremo in terms of modelling productivity and quality. We will also work on improving the tool, regarding efficiency of resource import and exploring other types of assistance, for example pro-active.

Acknowledgements. We would like to thank the reviewers for their valuable comments. This work was partially supported by the Ministry of Education of Spain (FPU grant FPU13/02698), the Spanish Ministry of Science (RTI2018-095255-B-I00), and the R&D programme of the Madrid Region (S2018/TCS-4314).

References

[1] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, 2nd Edition, Morgan & Claypool, 2017.

[2] Á. M. Segura, J. de Lara, P. Neubauer, M. Wimmer, Automated modelling assistance by integrating heterogeneous information sources, Computer Languages, Systems & Structures 53 (2018) 90–120.

[14] `https://github.com/angel539/extremo/wiki/Case-Studies`

[3] Eclipse Code Recommenders, `http://www.eclipse.org/recommenders`.

[4] K. Mens, A. Lozano, Source code-based recommendation systems, in: Recommendation Systems in Software Engineering, Springer, 2014, pp. 93–130.

[5] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-oriented domain analysis (foda) feasibility study, Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990).

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

[7] J. J. López-Fernández, J. S. Cuadrado, E. Guerra, J. de Lara, Example-driven meta-model development, Software and System Modeling 14 (4) (2015) 1323–1347.

[8] A. Dyck, A. Ganser, H. Lichter, A framework for model recommenders - requirements, architecture and tool support, in: MODELSWARD, 2014, pp. 282–290.

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

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

[11] C. Atkinson, B. Kennel, B. Goß, The level-agnostic modeling language, in: Software Language Engineering (SLE), Vol. 6563 of Lecture Notes in Computer Science, Springer, 2010, pp. 266–275.

[12] T. Murata, Petri Nets: Properties, Analysis and Applications, Proc. IEEE 77 (4) (1989) 541–580.

[13] Eclipse Graphical Editing Framework, `https://eclipse.org/gef/`.

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

[15] P. Neubauer, A. Bergmayr, T. Mayerhofer, J. Troya, M. Wimmer, XMLText: From XML Schema to Xtext, in: Proceedings of SLE, 2015, pp. 71–76.

[16] A. Benelallam, A. Gómez, G. Sunyé, M. Tisi, D. Launay, Neo4emf, A scalable persistence layer for EMF models, in: Proc. ECMFA, Vol. 8569 of Lecture Notes in Computer Science, Springer, 2014, pp. 230–241.

[17] A. Pescador, A. Garmendia, E. Guerra, J. S. Cuadrado, J. de Lara, Pattern-based development of domain-specific modelling languages, in: MoDELS, 2015, pp. 166–175.

[18] J. D. Rocco, D. D. Ruscio, L. Iovino, A. Pierantonio, Mining metrics for understanding metamodel characteristics, in: MiSE, ACM, 2014, pp. 55–60.

[19] J. Mengerink, A. Serebrenik, R. R. H. Schiffelers, M. G. J. van den Brand, Automated analyses of model-driven artifacts: obtaining insights into industrial application of MDE, in: IWSM-Mensura, ACM, 2017, pp. 116–121.

Required Metadata

Nr.	**(executable) Software metadata description**	
S1	Current software version	$v1.0.0$
S2	Permanent link to executables of this version	$https://github.com/angel539/extremo/$
S3	Legal Software License	EPL-2.0
S4	Computing platform/Operating System	MacOSx Sierra and Windows 7 64-bit or later
S5	Installation requirements & dependencies	Eclipse Mars2.0 (4.5.2a), EMF 2.12.0, Java 8, NeoEMF 1.0.2
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	$https://github.com/angel539/extremo/wiki$
S7	Support email for questions	Angel.MoraS@uam.es

Table 1: Software metadata

Nr.	**Code metadata description**	
C1	Current code version	$v1.0.0$
C2	Permanent link to code/repository used of this code version	$https://github.com/angel539/extremo/$
C3	Legal Code License	EPL-2.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Eclipse Mars2.0 (4.5.2a), EMF 2.12.0, Java 8, NeoEMF 1.0.2
C6	Compilation requirements, operating environments & dependencies	Java 8, Apache Maven 3
C7	If available Link to developer documentation/manual	$https://github.com/angel539/extremo/wiki$
C8	Support email for questions	Angel.MoraS@uam.es

Table 2: Code metadata