Model Driven Engineering for implementing the ISO 19100 series of international standards

Cyril Faucher, Jean-Yves Lafaye

MODEL-DRIVEN ENGINEERING FOR IMPLEMENTING THE ISO 19100 SERIES OF INTERNATIONAL STANDARDS

CYRIL FAUCHER[1] & JEAN-YVES LAFAYE[2]

[1] IRISA/INRIA, Campus de Beaulieu, 35042, Rennes cedex, France
[2] Laboratoire L3i, Université de La Rochelle, 17042, La Rochelle cedex, France

Tel. +33 2 99 84 73 77, Fax. +33 2 99 84 71 71
Email. cyril.faucher@irisa.fr

ABSTRACT: In this paper we discuss the implementation of the ISO 19100 series of standards with use of Model-Driven Engineering (MDE) techniques. We expose how MDE is beneficial for prototyping and building an end-user application dedicated to the sharing of environmental data (especially within the context of coastal zones). We present with an illustrating example of a way to use ISO 19109, 19110 and 19117 models in order to generate a rich client application. Our work takes the geographic features, the data structures and the data display concerns into account, by means of using the various facets of the ISO 19100 series.

ISO 19100 standards, metamodelling, MDE, georeferenced data, information system.

INTRODUCTION:

Among the geographic information community, there are many applications software and numerous data formats that implement the international standards (ISO). Each application software or format comes with its own pros and cons, but unfortunately, applications software usually present as black boxes and their interoperability is generally not ensured. Data providers from a coastal zone need to populate their information systems with more and more datasets. The types of these datasets may be very heterogeneous such as maps, numerical marine information or textual fishery laws. Thus cataloguing systems should be highly flexible and generic. A standard exchange format is not sufficient for building an efficient and robust information system: the design of such information systems should ground on a standard design framework. Besides, there is a need for accessing the application structure in order to manage system interoperability. Metamodelling and more widely Model-Driven Engineering (MDE) provide tools to manage applications thanks to a decomposition in several modelling layers.

The TC 211[1] ISO Technical Committee has specified and published the ISO 19100 series of standards that deal with geographic information. This series is composed of several specifications about data definition, data structure and metadata for georeferenced objects. The specification of the ISO 19100 series uses the UML (OMG, 2004) notation for all elements in the series and for the metamodels too, e.g.: 19109 (ISO/TC-211, 2005a), 19110 (ISO/TC-211, 2005b) and 19117 (ISO/TC-211, 2005c).

In this paper we discuss the implementation of these standards within the scope of MDE techniques. We explain the benefits of using MDE for prototyping and building an end-user application dedicated to the sharing of environmental data. We show the special interest of MDE, when dealing with numerous concepts of wide application software. Our work exploits the various facets of the ISO 19100 series and addresses the metadata catalogue, the geographic features, the data structures and the data display. Our work is part of an open-

[1] www.isotc211.org

source project called *Emios* (Environmental Memory Interoperable Open Service). *Emios* is a framework that aims at providing a complete range of services for storing and sharing information about research activities. More precisely, *Emios* exploits the concept of *Environmental Memory* (Guarnieri and al. 2003). *Emios* consists of a set of Eclipse plug-ins based on metamodelling tools like EMF, GMF[2] or Topcased[3].
The paper is organised as follows. The first section presents with general MDE concepts and focuses on model transformation and on some application prototyping tools. The relationship between MDE and Geographic Information is discussed in the second section. In Section 3, we take advantage of presenting an example, to show how to use the metamodels from the ISO 19100 series, for prototyping applications dedicated to the sharing of environmental data. We pay a special attention to the way ISO 19117 can be used for configuring the display of metadata attributes. We focus on reusability and interoperability concerns and conclude in Section 4, by sketching the future of our work.

MODEL-DRIVEN ENGINEERING:

Motivation

Model-Driven Engineering (MDE) is a relevant technique for modelling conceptual features in order to specify and implement a future application. When analysing the information system, the domain is usually described through UML (data) models. When dealing with application software design, we need to operate on these UML models, and then an upper level description language is required. This upper level language precisely is the UML metamodel, which is a model about UML models just the way metadata may be defined as data about data.

Main principles

The two prominent MDE concepts are the model and metamodel concepts. A model is an abstraction of the real world dedicated to some special goal. According to the MDE terminology, a system is represented by a model. A metamodel gives a standard (semi)formal specification for a set of features shared by several models. It provides the user with a guideline and a carrier of types to build his own models. A model is said to conform to its metamodel. These principles are illustrated by the OMG's four layers MDA architecture (Miller and Mukerji, 2003). Level M0 is for the real systems represented by models at level M1 that conform to M2 metamodels, themselves being conformant to the M3 meta-metamodel which is self conformant. Self conformance at level M3 is a key point (Bézivin 2003). It is then straightforward to specify any relationship between models by using the concepts and the rules (grammar) of their specific metamodels that all conform to the same unique top meta-metamodel. Model transformation actually exploits such facilities. More precisely, MDA aims at generating "platform-specific models" (PSM) from "platform independent models" (PIM). The QVT (Query View Transform) language has been especially created (OMG, 2005b) to allow the specification of such transformation. Choosing the MOF (MetaObject Facility) recommendation (OMG, 2006) at level M3 permits to refer to many standard metamodels: UML, CWM (OMG, 2003)... Similarly, projections between

[2] www.eclipse.org/modeling/
[3] www.topcased.org

technical spaces are made simple e.g.: XML (for instance the XMI standard (OMG, 2005a)) or Java (Dirckze, 2002) for a JMI specification of a Java API for model handling.
Model transformation is achieved by running programs whose inputs are made of one (or several) source model(-s) accompanied by the corresponding metamodel(-s) and by the target metamodel. The program code embodies the transformation process, and the outputs are the new models that conform to the given target metamodel. Among the numerous tools dedicated to model transformation, let's quote the Atlas Transformation Language (ATL) (Bézivin and al. 2003), QVT and Kermeta: an action meta-language (Muller and al. 2005).
MDE facilitates the creation of new concepts at the application model level, thanks to the abstraction. Besides, the new concepts are directly managed at the M2 level without need of processing at the M1 level. An analogy can be made with Relational Data Base Management Systems (RDBMS) in which. The usage of common metamodels and a standard way to save models via XMI guarantee the interoperability between systems.

Prototyping an application thanks to MDE tools
Within the MDE community, many efforts are made to provide tools for prototyping applications. Relevant results are obtained by the Eclipse community within the project called *modeling*. Eclipse[4] is an open-source application written in Java that provides some basic services like a file system navigator, file editors and an extension mechanism based on plug-ins. Thus the Eclipse community, through different projects, offers abilities around the Web Application: html, xml and php editors, moreover Java editor and compiler or file version system. Any user can create his own plug-ins and add facilities to an instance of the Eclipse workbench. The Eclipse project EMF (Eclipse Modeling Framework) aims at providing a collection of tools and several model generation processes in order to create metamodels and manipulate their conformant models. Actually, EMF allows to generate the Java source code that manages the metamodel instances (models) thanks to a configuration file named *genmodel*. The generated source code also provides a rich client application in order to create and fill models through a tree editor and a property view. The resulting implementation is in accordance with the "Model View Controller pattern" (MVC) that enhances the source code reusability. The model part uses the XMI-OMG standard and allows to save models in a ready-to-exchange format. The final code is as a set of Eclipse plug-ins that may be easily deployed in other Eclipse applications.
A tree editor is useful, but not always proves to be friendly when the given model actually is not a tree. So, more sophisticate graphical editors might be appreciated. Some MDE open-source projects treat of the matter such as GMF (Graphical Modeling Framework) and Topcased. Here, we use the Topcased toolkit. The principle of this tool is to define a configuration file that stores a mapping between the graphical concepts and the concepts in the metamodel. This file is based on both the metamodel and the *genmodel* files. After the configuration step, a generation process is performed and the result is a plug-in set too. The generated source code may be finally customised to fulfil the end-user requirements. An annotation mechanism allows a further generation that takes the changes into account. These techniques have been used to generate UML graphical editors for designing UML models such as: class diagrams, state charts...

[4] www.eclipse.org

MODEL-DRIVEN ENGINEERING AND GEOGRAPHIC INFORMATION:

Overview

The TC 211 ISO Technical Committee worked on geographic information standard specification and provided the ISO 19100 series. It is composed of several specifications about data definition, data structure and metadata for georeferenced objects. It treats of data processing, analysis, display, storage, sharing and transfer. These standards are defined through UML models and the TC 211 provides metamodels for all 19109, 19110 and 19117 standards. ISO 19109 models provide the description of geographic features. ISO 19110 gives a metamodel which is the framework for building metadata application models. ISO 19117 presents the portrayal specification for feature types or feature instances. These three metamodels are dependent: ISO 19110 inherits from ISO 19109 and ISO 19117 references ISO 19109 elements in order to configure the display. Some works discuss about the implementation of ISO 19100 standards by using metamodelling techniques (Einspanier, 2004). A part of the *Emios* project is dedicated to the generation of an API from a metamodel based on the ISO 19100 specifications. We specify metamodels according to the UML models provided by the TC 211. Our approach differs from GeoAPI[5] project, but we will not discuss the discrepancy here. The better way to take benefit of EMF tools is to promote ISO models to *Ecore* metamodels (thus stored in an *Ecore* file). All the concepts present in standards have been translated, but some slight changes have been done to fix some modelling mistakes. The following examples are based on ours metamodels.

Towards model-driven Metadata application

Figure 1 shows ISO 19100 concepts according to the three MDA abstraction levels. The conformance relationship means that a model satisfies the rules that are defined in its corresponding metamodel. In the ISO 19101 specification, the relationships between metamodel and model elements are defined in another way, namely using references that link elements from one level to elements of another. As can be seen in the Figure 1, we have chosen to strictly conform to the MDA paradigm that is closed to the implementation process. The metamodel level (M2) is composed of ISO 19109, 19110, 19117 and all the other metamodels are included in the ISO 19100 series.

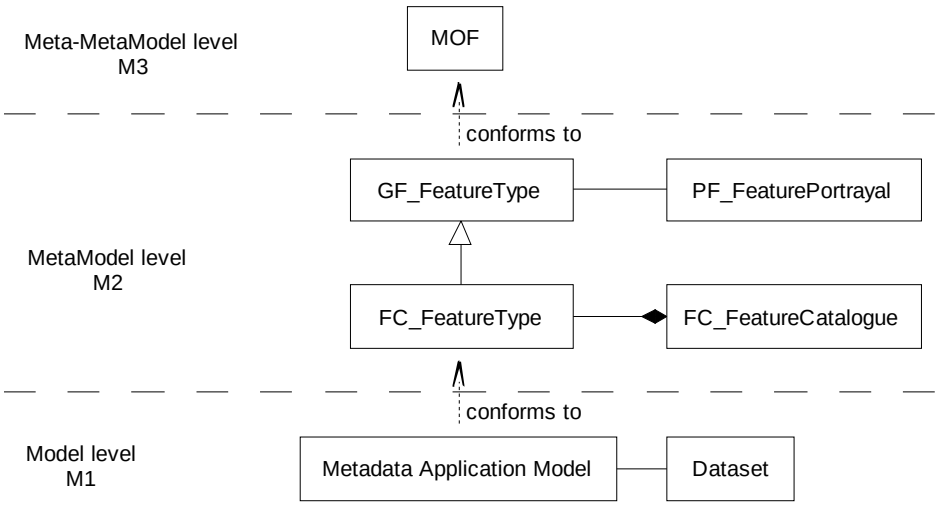

Figure 1. The three abstraction levels for modelling an application according to ISO 19100 concepts

[5] http://geoapi.sourceforge.net/

All metamodels conform to the MOF-OMG specification at level M3. The general concepts defining georeferenced feature are defined by the metamodels, then at the M2 level. Actually, through the concepts of GF_FeatureType, FC_FeatureType and PF_FeaturePortrayal, the application designer may choose the features he wants to retrieve in the final application (i.e.: at the application model level (M1)). In the example below, the application model concerns metadata, we can see that metadata elements are defined in the terms of *FC_FeatureType* (Feature Catalogue - Feature Type). Dataset are considered all set at the same level: the metadata application model, because its definition and metamodel is defined at the M2 level.
Note: Conversely, if we had considered the metadata as a meta-information on datasets, then model elements corresponding to datasets should have conformed to the metadata application model and the representation should be different.

HOW TO USE ISO 19117 TO GENERATE A GRAPHICAL APPLICATIVE LAYER:

The ISO 19117 metamodel

ISO 19117 defines a feature-centred rule based portrayal mechanism. Actually, the ISO 19117 allows to specify if a *Feature Portrayal* is linked to either a *Feature Type* or a *Feature Instance*. Figure 2 below shows an excerpt of the ISO 19117 metamodel.

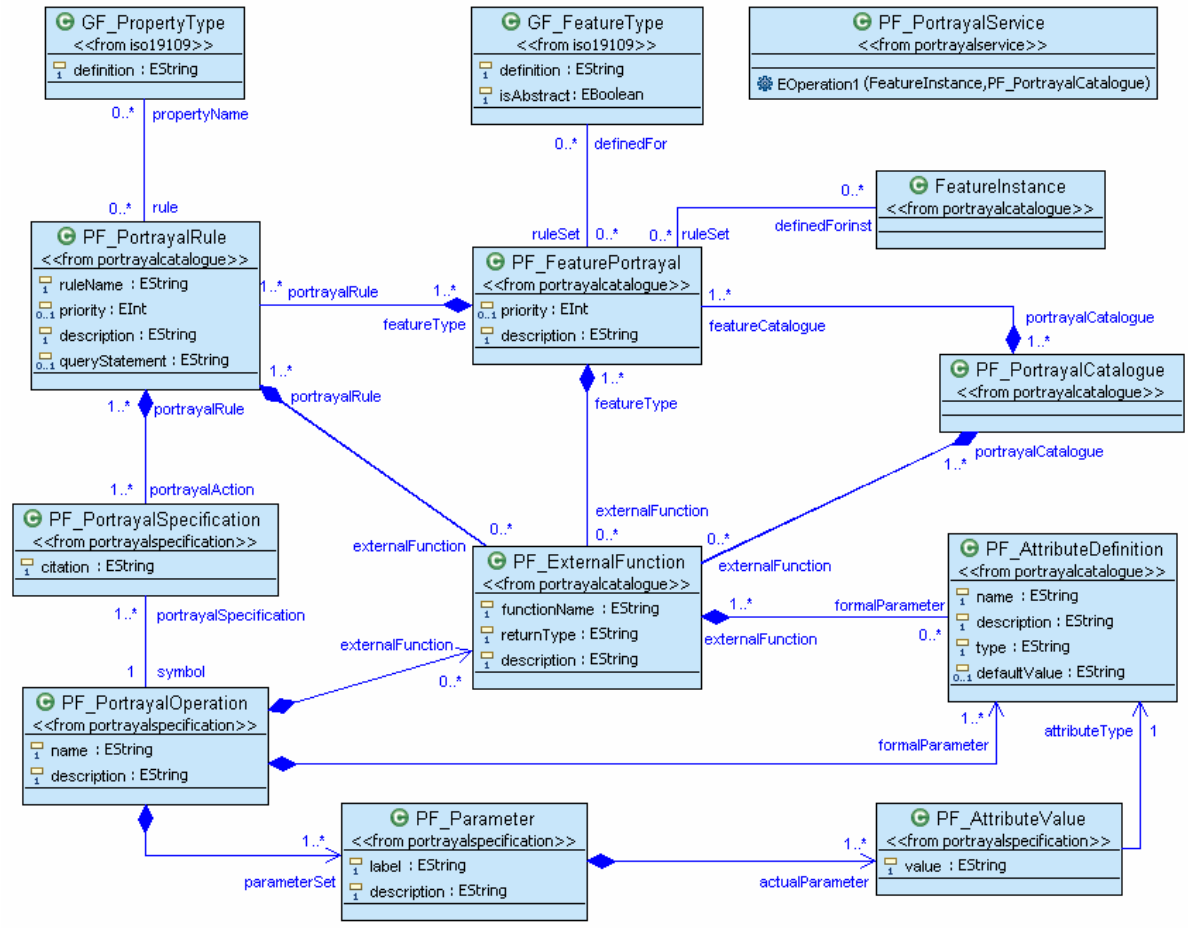

Figure 2. ISO 19117 metamodel excerpt (ISO/TC-211, 2005c)

A *PF_PortrayalCatalogue* contains a collection of feature portrayals. The connection between ISO 19117 and ISO 19109 is achieved by the *PF_FeaturePortrayal* class. A *PF_FeaturePortrayal* may be defined to portray at two granularity levels, i.e.: *GF_FeatureType* (e.g.: abstract notion of "city") and *FeatureInstance* (e.g.: the concrete city: "Paris"), this ability is a key point that allows both a generic and a specific customisation. The portrayal mechanism is a dynamic function. Thus into the final application, a *PF_FeaturePortrayal* is performed when the system needs to display a feature type or a feature instance. A *PF_FeaturePortrayal* contains a set of rules (*PF_PortrayalRule*) that are performed when a feature portrayal is called. The portrayal information is handled as *PF_PortrayalSpecification* that applies according to *PF_PortrayalRule*: a portrayal rule is evaluated, precisely the *queryStatement* attribute, and if its value is "true" then the portrayal specification is applied. The portrayal rules may be expressed using the OCL constraint language and UML action language. Finally, the action specifying the display is given by the *PF_PortrayalOperation*. *PF_PortrayalOperation* are collected under the *portrayalAction* reference. *PF_ExternalFunction* specifies the interface with external functions such as Java static calls. The external functions are used as helpers and are called by a *PF_PortrayalOperation.*

Graphical editor generation process

In the following, we show how to use the ISO 19100 metamodels and MDE tools to generate a dedicated metadata graphical editor. Figure 3 shows the generation process of the graphical editor configuration models.

The generation of the graphical editor comprises three steps. The first one instantiates the ISO 19100 metamodel in order to obtain a set of models that allow in a second step the generation of a new set of models corresponding to the graphical editor configurators. The final source code generation is performed in the third step.

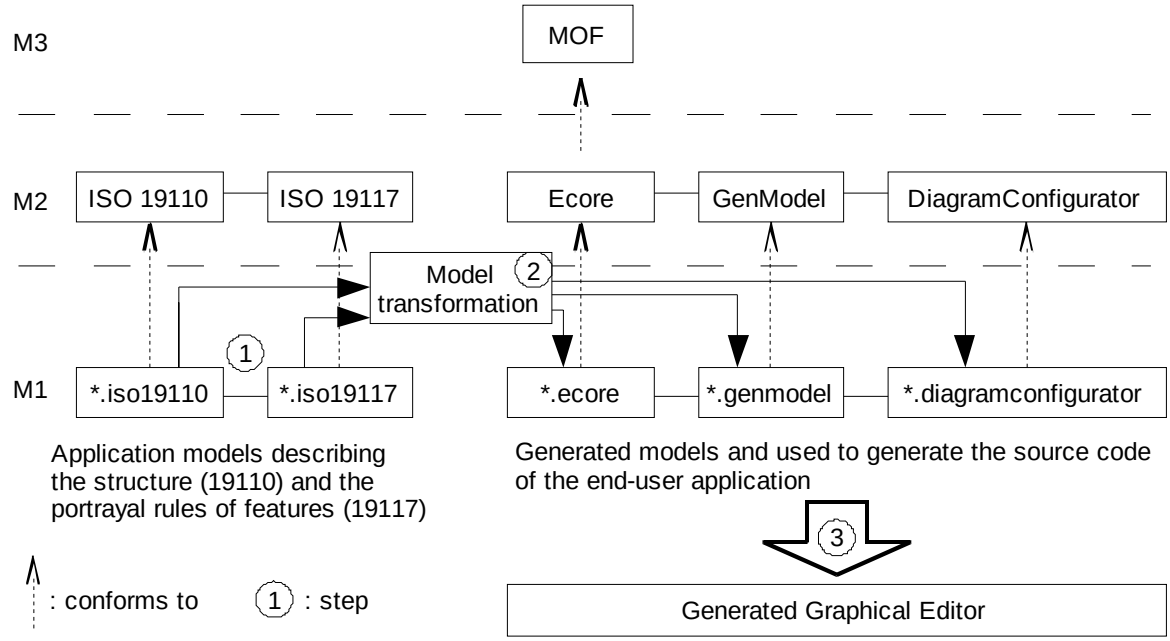

Figure 3. The generation process of the graphical editor

The transition between the first and the second step uses a model transformation. In this transformation, the inputs are a structure model (ISO 19110 model) and the portrayal specification (ISO 19117 model). The output of the transformation is of three models. The first one is an *Ecore* model that represents the structure in terms of *Ecore*. The second is the *genmodel* that configures a part of the source code generation. Finally, the third model is the *diagramconfigurator* that defines the mapping between the graphical and metamodel concepts. The model transformation is performed thanks to the action meta-language: Kermeta and it is based on interpretation rules that allow the mapping between the elements. All the generated models are actually used to generate of the graphical editor source code.
The two models that are defined during the first step correspond to the "modelling" part of the process. Actually, the ISO 19110 and ISO 19117 models represent the application model and use the domain specific languages that are defined in the ISO 19110 and ISO 19117 metamodels. The ISO 19110 model represents the "structure" layer of the application model. It defines the *Feature Types* required by the domain modelling. Besides, the ISO 19117 model represents the "portrayal" layer, which maps feature elements or instances to their graphical counterpart. During the model transformation, generating the output models, corresponds to the "configuration" part of the process. Some configuration parameters may be modified to fulfil the end-user's requirements. Finally the generated source code (Java) can be overloaded. We can see that each model (modelling and configurator) has its own capability without introducing any redundancy between them, in other words we enhance the separation of concepts. Moreover, all the models are defined and stored in a homogeneous way, hence improving the interoperability and the writing of programs that manage them.
The structure model is an instance of the ISO 19110 metamodels. Thus, the model transformation, that converts the structure model to an *Ecore* model, consists of the mapping between a *FeatureType* instance and becomes an *EClass* instance (*EClass* comes from *Ecore* concepts), e.g.: the "MD_Metadata" *FeatureType* becomes an EClass named "MD_Metadata". The feature attributes are translated into *EAttribute*, the feature associations into *EReference*.
The configuration diagram model has been built from the content of the *ExternalFunction*. Actually, we have defined a set of external functions corresponding to the initialisation methods of the configuration diagram model, e.g.: setColour(int R, int G, int B), setFont(String type, int size), setFigureType(String type). The external functions are interpreted with Kermeta which returns the configuration diagram model. The priority attribute from the *PortrayalRule* class is used to define the default display (i.e.: priority = 0). The configuration diagram model contains only parameters from these default configurations. In order to take the other rules into account, *EditPart* policies are generated and inserted in the source code. The *EditPart* policies mechanism is provided from Topcased.

Case study requirements

We designed a case study in order to make use of the technique. Thus, thanks to the EMF capabilities, we instantiate the ISO 19110 metamodel in order to get a Metadata conceptual schema with a "MD_Metadata" and a "MD_Identification" *FeatureType*. We add three feature attributes in the "MD_Metadata" *FeatureType*: fileIdentifier, date and contact. We also add two feature attributes into the "MD_Identification" *FeatureType*: abstract, purpose. "MD_Metadata" contains at least one "MD_Identification". We want to display "MD_Identification" properties in a yellow box. If the properties are not filled, the background of the box should become red to say a required parameter is empty. If the text

size of the abstract attribute is more than 25 characters then only 25 characters should be displayed with "…".
Then, "yellow" is set as the default colour (priority = 0) and is recorded in the configuration model. Otherwise, red is considered as a supplementary font colour: priority = 1 and recorded as an *EditPart* policy. To achieve the display of only 25 characters, an external operation is performed.

A specification example:
FeaturePortrayal: "identification1" linked to "MD_Identification" (instance of *FeatureType*)
PortrayalRule "pr0" without link
priority = 0 // default portrayal
portrayalAction = (" MD_Identification.setFigureType("box") ") // figure type
portrayalAction = (" MD_Identification.abstract.setFigureType("label") ")
portrayalAction = (" MD_Identification.purpose.setFigureType("label") ")
portrayalAction = (" MD_Identification.setColour(255,255,128) ") // yellow for the box background
PortrayalRule "pr1" linked to "abstract" and "purpose"
priority = 1
queryStatement = (" if(abstract.getSize()==0 or purpose.getSize()==0) ")
portrayalAction = (" MD_Identification.setColour(255,0,0) ") // red
PortrayalRule "pr2" linked to "abstract"
priority = 1
queryStatement = (" if(abstract.getSize()>25) ")
portrayalAction = (" abstract.subString(25,0) ") // subString method

External functions that are used:
getSize(String text) : int
setFigureType(String type)
setColour(int R, int G, int B)
subString(int size, int start) : String

PortrayalCatalogue: "PC1" contains {identification1, getSize, setFigureType…)

Output of the source code generation process
The result of the generation is a set of Eclipse plug-ins. They could be reused in another Eclipse workbench, e.g.: a GIS tool that is based on Eclipse technology.
We can see (Figure 4) the three parts of the user interface: the graphical editor area, the property view to fill the parameters and the "metadata model" outline that provides a tree view of the model. The property shows the full content of the attribute abstract, and the graphical view displays only 25 characters. A metadata specification allows several definitions of MD_Identfication, and the graphical editor may create several identification boxes. Let's notice that, the second box is red, because its attributes are not filled.

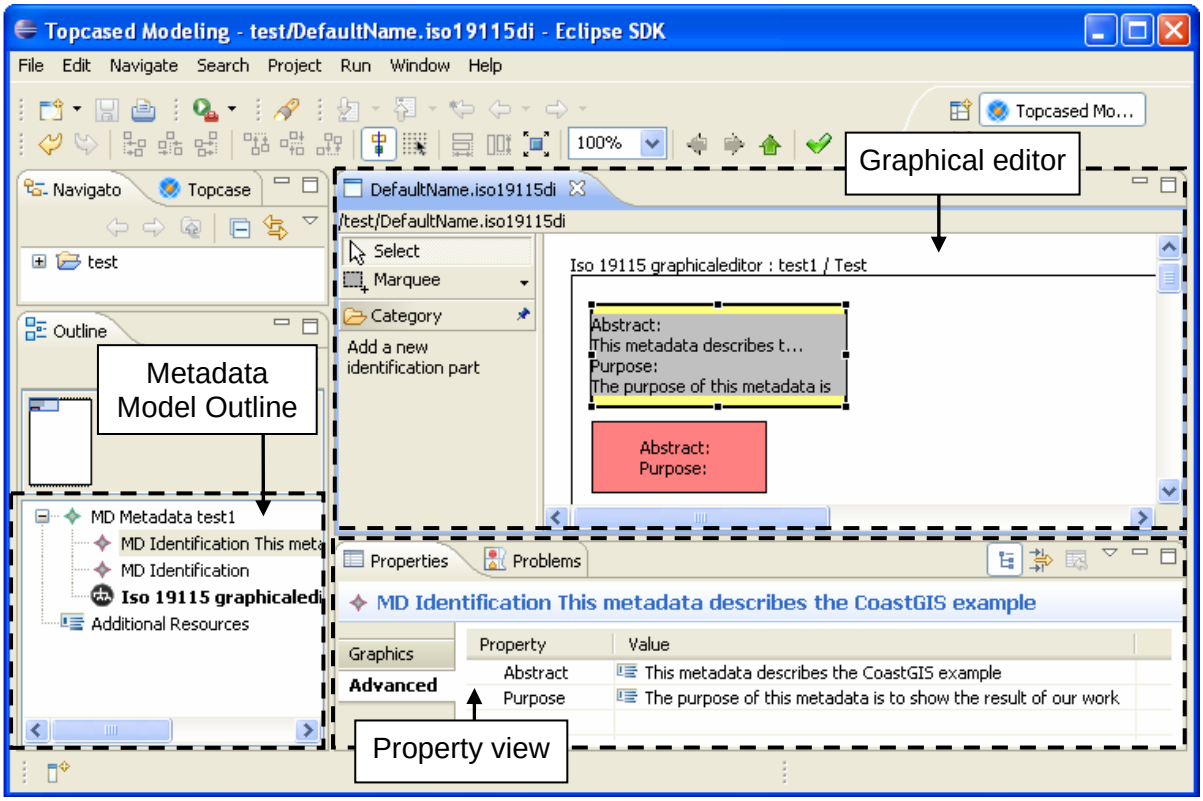

Figure 4. Screenshot of the generated end-user application

CONCLUSION AND FUTURE WORKS:

In this paper we investigated various ways to use an abstract data portrayal for generating a graphical editor. We intended to show the interest of Model-Driven Engineering (MDE) for implementing parts of the ISO 19100 series of International Standards. We insisted on prototyping abilities and on the generic feature of our approach. In particular, we presented with an example how to use the portrayal part of ISO 19100, i.e.: ISO 19117. ISO 19117 model properties are used to configure the display of metadata attributes *via* the generation of several configuration models. The result is a graphical editor that allows to populate metadata models instances from customised ISO 19115 model and that in an interoperable and friendly way. The final application is designed to fulfil the requirements of the end-user.

Presently, our work aims at defining generic portrayal rules that could be reused in other contexts. We extend the metadata application schema in order to take attributes that are dedicated to research activities into account. Thus we are targeting step by step the complete environmental memory concept. This extension should be a subset of a metadata profile in order to continue ensuring interoperability that facilitates the connection with other existent tools.

REFERENCES:

Bézivin, J. (2005), “On The Unification Power of Models”, *Software and System Modeling*, vol. 4(2), pp. 171–188.

Bézivin, J., Dupé, G., Jouault, F., Pitette, G. and Rougui, J. (2003), "First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery", *Proc. OOPSLA 2003 Workshop, Anaheim, USA.*
Dirckze, R. (2002), Java Metadata Interface (JMI) Specification, version 1.0, JSR 040.
Guarnieri, F., Garbolino, E., Houllier, F., Cuq, F., Lévêque, C., Weill, A. and Matarasso, P. (2003), Contribution à la définition opérationnelle et à la modélisation de la mémoire environnementale des zones ateliers, in: Lévêque, C. and Leeuw, S. (eds), *Quelles natures voulons-nous ? Pour une approche socio-écologique du champ de l'environnement*, Elsevier, Paris, France, pp. 296-307.
Einspanier, U. (2004), "Enhancing GI Discovery with ISO Feature Type Catalogues – A Metamodelling Approach", *Proc. 15th International Workshop on Database and Expert Systems Applications (DEXA'04)*, vol. 1529-4188/04, IEEE.
ISO/TC-211 (2002), 19101 Geographic information – Reference model, International Organization for Standardization.
ISO/TC-211 (2003), 19115 Geographic information – Metadata, International Organization for Standardization & Open-GIS Consortium.
ISO/TC-211 (2005a), 19109 Geographic information – Rules for application schema, International Organization for Standardization.
ISO/TC-211 (2005b), 19110 Geographic information – Methodology for feature cataloguing, International Organization for Standardization.
ISO/TC-211 (2005c), 19117 Geographic information – Portrayal, International Organization for Standardization.
Miller, J., and Mukerji, J. (2003), MDA Guide, version 1.0.1.
Muller, P.A., Fleurey, F. and Jézéquel, J.M. (2005), "Weaving executabillty into object-oriented meta-languages", in: Kent, S. and Briand, L. (eds), *Proc. MODELS/UML'200*5, *Montego Bay, Jamaica*, vol. 3713 of *LNCS*, Springer-Verlag, pp. 264-278.
OMG (2003), Common Warehouse Metamodel (CWM) Specification, version 1.1.
OMG (2004), Unified Modeling Language: Superstructure, version 2.0.
OMG (2005a), MOF 2.0/XMI Mapping Specification, version 2.1.
OMG (2005b), MOF QVT Final Adopted Specification.
OMG (2006), Meta Object Facility (MOF) Core Specification, version 2.0.

ACKNOWLEDGEMENTS:
These works are developed within the framework of the *Emios* open-source project. We thank the Geomer laboratory's members for their feedbacks on *Emios*.