A tool for the automatic generation of multimodel editors

David Blanes
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
dblanes@dsic.upv.es

Javier Gonzalez-Huerta
LATECE Group, Université de Québec à Montréal
Avé President Kennedy, PK4938, H2X 3Y7, Montreal, QC
gonzalez_huerta.javier@uqam.ca

Emilio Insfran
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
einsfran@dsic.upv.es

Abstract— **In a software development process, normally different stakeholders have different system views at different abstraction levels. This heterogeneity when applying the Model-Driven Software Development approach requires the use of multiple views to represent a software system. However, EMF tools are focused on the generation of editors for one unique view. This fact affects negatively the maintainability, e.g. one editor must be modified when a new view is added in the system. In this demo, we present an infrastructure where the user can create metamodels for representing multiple views, and their relationships. This infrastructure allows creating automatically editors for any multimodel that represents these multiple views by means of the EMF code generation facilities. This solution provides a flexible way for generating automatically multimodel editors. Additionally, we show an example of a multimodel editor that illustrates the feasibility of the infrastructure. A video with a demo can be found at: http://goo.gl/rLmmZK.**

Index Terms— **Multi-view modelling, EMF, model editors, editor generation.**

I. Introduction and Motivation

Usually multiple artifacts, roles, and phases are involved in a software development process. When following a Model-Driven Software Development approach, a properly software system representation is usually composed by multiple views. These views are not isolated, and thus, it is necessary to establish relationships among them. In a possible scenario, we could have a system to represent a web page initially formed by two views: UML class diagrams to specify the structure, and Domain Specific Language (DSL) to represent the interface. Since the UML has no models to represent graphical interfaces, we should establish a relationship among each UML class from the class diagram, and the interface represented in our DSL.

A suitable solution to this problem is the use of a multimodel, which it is composed of multiple views, represented as viewpoint models, and the relationships among them. An editor for this kind of representations needs to support not only the addition and removal of views, but also the establishment of relationships among these views and in a flexible way. Furthermore, it is required also to be able to import models (and their metamodels) created by using third party tools.

Eclipse Modelling framework (EMF) is a framework, and code generation facility, for building tools and other applications, e.g. model editors, based on a structured data model [1]. EMF allows defining metamodels, and generating automatically the model editors for these metamodels. However, for creating multimodel editors this solution is not enough due the high coupling between the model and the multimodel editor (i.e., each time that a new view is created in the multimodel, it is necessary to change the graphical editor).

In this demo we propose an infrastructure to address these problems. This infrastructure allows creating editors for multimodels adapted to the specific models in an automatic way, by using the EMF code generation facilities. Additionally, we show an example of a multimodel editor that illustrates the feasibility of this infrastructure for creating multimodel editors. Section II describes the designed multimodel architecture used to give infrastructure support. Section III describes the infrastructure to generate automatically editors and the implemented multimodel editors. Section IV, discusses similar approaches. Finally, Section V summarizes the demonstration.

II. The multimodel architecture

A two-layer architecture is proposed to support our infrastructure. Firstly, a generic metamodel, called Core metamodel, is proposed for containing the necessary elements to represent a multimodel. A multimodel is defined as a set of interrelated models that represents different viewpoints of a particular system. A viewpoint is an abstraction that yields a specification of the whole system restricted to a particular set of concerns. In any given viewpoint it is possible to define a model of the system that contains only the objects that are visible from that viewpoint. Such a model is known as a viewpoint model, or a view of the system from that viewpoint [2]. Secondly, the user creates custom metamodels for specific problems that extends the meta-classes of the Core metamodel.

The Core metamodel is based on the metamodel proposed at [3] for supporting multimodels. Fig. 1 presents the structure of the multimodel metamodel following an UML-like notation, where the dotted lines represent metaclasses that are contained on external metamodels. Any viewpoint model (*ViewPointModel*) contains the necessary entities (*Entity*) and relationships (*Relationship*) in the viewpoint model. An interesting application of metamodel structure is that we can load models that are supported by third party tools in our multimodel in a flexible and easy way.

However, the multimodel is not limited to contain viewpoint models. The multimodel also contains additional entities (*MultimodelEntity*) that extend entities, on the proxy-pattern [3], defined on the different viewpoint models defined so that they hold inter-viewpoint relationships (*MultimodelRelationship*). These relationships involve two multimodel entities extending model entities on different viewpoints. Finally, the multimodel entities and relationships could contain additional attributes (Multimodel Attribute). The attributes allows enriching a multimodel entity with additional information, e.g. add OCL constraints [3].

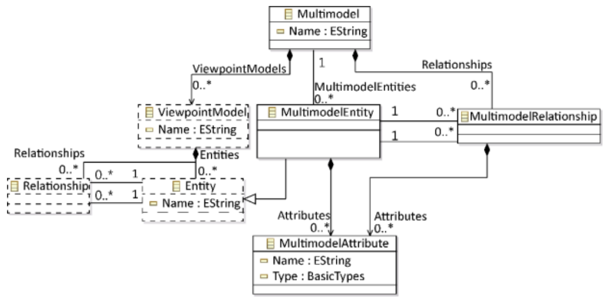


Fig. 1. Core Multimodel[1]

III. An infrastructure for generating automatically editors

To support the multimodel definition with two or more viewpoints, and the establishment of relations among elements of these viewpoints, it has been defined an infrastructure for editing and managing multimodels that implements the multimodel generic structure. In this section we describe how was carried out the implementation of this infrastructure. First, the user creates a custom metamodel to support a multi-model for a specific domain by extending the Core metamodel. Second, the user generates automatically and editor by using the EMF code generation facilities. We show an example of a multimodel editor that illustrates the feasibility of this infrastructure to create multimodel editors. The custom metamodel is used for representing a software product line from the automotive domain which comprises the safety-critical embedded software systems responsible for controlling a car [4]. This example defines a multimodel[2] used for deriving product architectures with the required quality attributes from the product line architecture. This editor allows editing viewpoints, entities, and relationships. The multimodel used to describe SPLs is composed of three interrelated viewpoints:

[1] We use an UML-like notation to illustrate the multimodel structure. However, it only expresses the generic structure of a multimodel.

[2] The example material is available at: http://users.dsic.upv.es/~jagonzalez/CarCarSPL/

- The *variability viewpoint* expresses the commonalities and variability within the product line. It is represented with the notation proposed in [5]. The main element is the feature, which represents a system functionality.
- The *architectural viewpoint* represents the architectural variability of the Product Line architecture that realizes the external variability of the SPL expressed in the variability viewpoint. It is expressed by means of the Common Variability Language (CVL) [6] and its main element is the Variability Specification (*VSpec*).
- The *quality viewpoint* represents the hierarchical decomposition of quality into sub-characteristics, quality attributes, metrics and the impacts and constraints among quality attributes. It extends the ISO/IEC 2500 (SQuaRE) standard [7], thus providing the quality assurance and evaluation activities in SPL development with support.

A. Generation of multimodel editors

The user can register metamodels that represent viewpoints on a custom meta-model. The user can include as a view any EMF-compliant metamodel that represents a views point. The user loads the necessary viewpoint models, entities, and establishes relationships adapted to the problem domain. Once the custom metamodel is created, then the user generates the editor code by using the EMF code generation facilities [1]. Fig. 2 shows the obtained automotive multimodel generator. It is possible creating instances from this metamodel, and editing these instances by a standard tree-editor by means of the generated editor code.

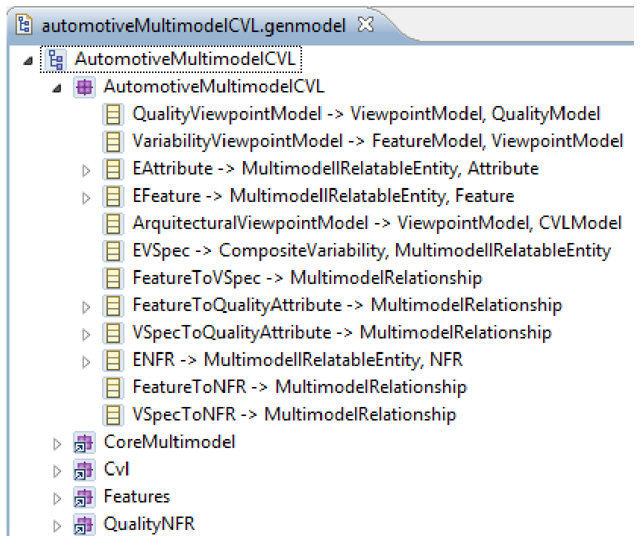
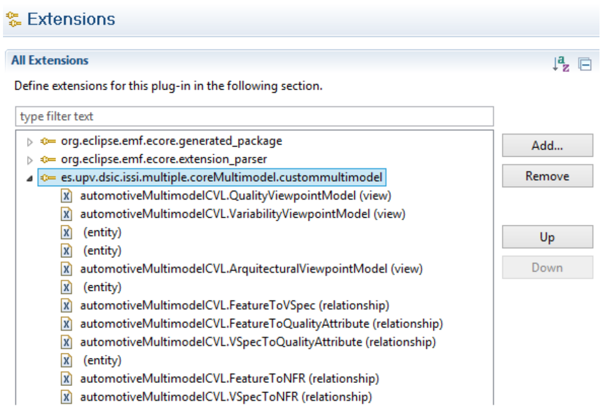

```
automotiveMultimodelCVL.genmodel ✕
◢ AutomotiveMultimodelCVL
  ◢ AutomotiveMultimodelCVL
      QualityViewpointModel -> ViewpointModel, QualityModel
      VariabilityViewpointModel -> FeatureModel, ViewpointModel
    ▷ EAttribute -> MultimodelIRelatableEntity, Attribute
    ▷ EFeature -> MultimodelIRelatableEntity, Feature
      ArquitecturalViewpointModel -> ViewpointModel, CVLModel
      EVSpec -> CompositeVariability, MultimodelIRelatableEntity
      FeatureToVSpec -> MultimodelRelationship
    ▷ FeatureToQualityAttribute -> MultimodelRelationship
    ▷ VSpecToQualityAttribute -> MultimodelRelationship
    ▷ ENFR -> MultimodelIRelatableEntity, NFR
      FeatureToNFR -> MultimodelRelationship
      VSpecToNFR -> MultimodelRelationship
  ▷ CoreMultimodel
  ▷ Cvl
  ▷ Features
  ▷ QualityNFR
```

Fig. 2. Custom multimodel generator

Once the editor code is generated, the views are registered in the plugin.xml file by using a wizard, which it is included in our tool. The mechanism is based on the eclipse extension points [6]. Fig. 3 shows the obtained extension point for the automotive multimodel. This registered information is used by the implement multimodel editor described above.



```
Extensions

All Extensions
Define extensions for this plug-in in the following section.
type filter text
▷ org.eclipse.emf.ecore.generated_package                      Add...
▷ org.eclipse.emf.ecore.extension_parser                       Remove
◢ es.upv.dsic.issi.multiple.coreMultimodel.custommultimodel
    automotiveMultimodelCVL.QualityViewpointModel (view)
    automotiveMultimodelCVL.VariabilityViewpointModel (view)
    (entity)                                                    Up
    (entity)                                                    Down
    automotiveMultimodelCVL.ArquitecturalViewpointModel (view)
    (entity)
    automotiveMultimodelCVL.FeatureToVSpec (relationship)
    automotiveMultimodelCVL.FeatureToQualityAttribute (relationship)
    automotiveMultimodelCVL.VSpecToQualityAttribute (relationship)
    (entity)
    automotiveMultimodelCVL.FeatureToNFR (relationship)
    automotiveMultimodelCVL.VSpecToNFR (relationship)
```

Fig. 3. Generated extension point for registering the multimodel

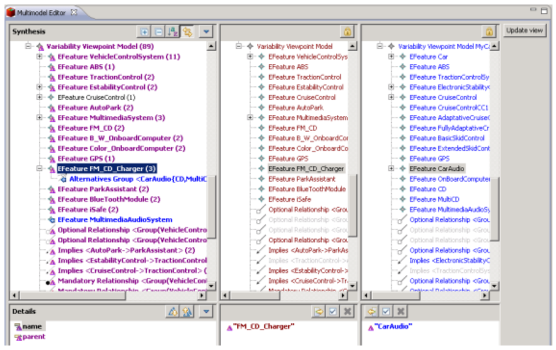
B. *Multimodel edition*

In this sub-section, we describe the main functionalities and features of an editor created by using the infrastructure as example to illustrate the infrastructure for generating multimodel editors. The editor has been implemented with a multi-tab design. Each tab contains an editor dedicated to a specific functionality: edit views, import models, update models, edit elements, and edit relationships. The editor is able to create a specific view from scratch. Hence, the multi-model view is empty and then the user is responsible for manually add the entities and relationships. The editor is able to recognize the allowed views, entities, and relationships in the multimodel by using the information registered in the plugin.xml file. This solution allows decoupling among the editor interface and the models.

If we have a model created using an EMF-based third party tool, we can import it to the multimodel by using the import view. We select the model to import in the editor, and then the model is loaded into memory by means of dynamic EMF code. If there are entities in the model that had been extended at design time in the meta-model, then the tool performs a type conversion (casting) of that entity. Thus, the original entity (*Entity*) is casted to an extended entity type (*MultimodelEntity*). This representation is based on inclusion polymorphism (also called sub-typing or redefinition).

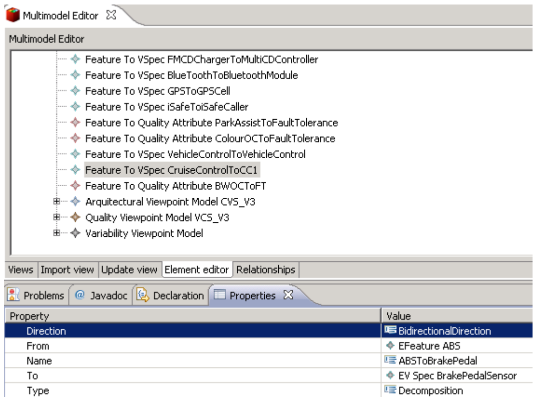
In the case that we would update an existing view, the editor also allows to update an existing viewpoint. For this purpose we have integrated a tool called EMFCompare [8], which provides a utility to compare models. When it is necessary to update a viewpoint, the viewpoint have to be loaded first by using the wizard, and then the editor shows the model differences among the loaded model and the corresponding viewpoint of the metamodel. This allows edit external tools with models that have been imported multimodel, even if their bodies are part of such relationships multi-model. Fig. 4 shows the editor integrating a new version of the variability viewpoint model.

The editor also integrated a tree-editor (see Fig. 5) for the multimodel entities. It supports the addition, modification, and removal of any multimodel element. Finally, the multi-model includes a relationship editor among the different viewpoints.



```
Multimodel Editor

Synthesis
⊟ Variability Viewpoint Model (89)
   ⊞ EFeature VehicleControlSystem (11)
     EFeature ABS (1)
     EFeature TractionControl (2)
   ⊞ EFeature EstabilityControl (2)
   ⊞ EFeature CruiseControl (1)
     EFeature AutoPark (2)
   ⊞ EFeature MultimediaSystem (3)
     EFeature FM_CD (2)
     EFeature B_W_OnboardComputer (2)
     EFeature Color_OnboardComputer (2)
     EFeature GPS (1)
   ⊟ EFeature FM_CD_Charger (3)
       Alternatives Group <CarAudio{CD,MultiC
     EFeature ParkAssistant (2)
     EFeature BlueToothModule (2)
     EFeature iSafe (2)
     EFeature MultimediaAudioSystem
     Optional Relationship <Group(VehicleContro
     Optional Relationship <Group(VehicleContro
     Implies <AutoPark->ParkAssistant> (2)
     Implies <EstabilityControl->TractionControl
     Implies <CruiseControl->TractionControl> (
     Mandatory Relationship <Group(VehicleCont

⊟ Variability Viewpoint Model
   ⊞ EFeature VehicleControlSys
     EFeature ABS
     EFeature TractionControl
     EFeature EstabilityControl
     EFeature CruiseControl
     EFeature AutoPark
   ⊞ EFeature MultimediaSystem
     EFeature FM_CD
     EFeature B_W_OnboardCo
     EFeature Color_OnboardCo
     EFeature GPS
     EFeature FM_CD_Charger
     EFeature ParkAssistant
     EFeature BlueToothModule
     EFeature iSafe
     Optional Relationship <Grou
     Optional Relationship <Grou
     Optional Relationship <Grou
     Optional Relationship <Grou
     Implies <AutoPark->ParkAs
     Implies <TractionControl->
     Implies <EstabilityControl->
     Implies <CruiseControl->Tr

⊟ Variability Viewpoint Model MyC                         Update view
   ⊞ EFeature Car
     EFeature ABS
     EFeature TractionControlSy
   ⊞ EFeature ElectronicStability
   ⊞ EFeature CruiseControl
     EFeature CruiseControlCC1
     EFeature AdaptativeCruise
     EFeature FullyAdaptativeCr
     EFeature BasicSkidControl
     EFeature ExtendedSkidCont
     EFeature GPS
   ⊞ EFeature CarAudio
     EFeature OnBoardComputer
     EFeature CD
     EFeature MultiCD
     EFeature MultimediaAudioSy
     Optional Relationship <Grou
     Optional Relationship <Grou
     Optional Relationship <Grou
     Optional Relationship <Grou
     Implies <ElectronicStabilityC
     Implies <TractionControlSys
     Optional Relationship <Grou

Details
  name          "FM_CD_Charger"          "CarAudio"
  parent
```

Fig. 4. Editor for updating the viewpoint models



```
Multimodel Editor
Multimodel Editor
     Feature To VSpec FMCDChargerToMultiCDController
     Feature To VSpec BlueToothToBluetoothModule
     Feature To VSpec GPSToGPSCell
     Feature To VSpec iSafeToiSafeCaller
     Feature To Quality Attribute ParkAssistToFaultTolerance
     Feature To Quality Attribute ColourOCToFaultTolerance
     Feature To VSpec VehicleControlToVehicleControl
     Feature To VSpec CruiseControlToCC1
     Feature To Quality Attribute BWOCToFT
  ⊞ Arquitectural Viewpoint Model CVS_V3
  ⊞ Quality Viewpoint Model VCS_V3
  ⊞ Variability Viewpoint Model
Views | Import view | Update view | Element editor | Relationships

Problems | Javadoc | Declaration | Properties
```

Property	Value
Direction	BidirectionalDirection
From	EFeature ABS
Name	ABSToBrakePedal
To	EV Spec BrakePedalSensor
Type	Decomposition

Fig. 5. Multimodel tree editor

IV. Related Work

In this section, we analyze approaches that represent the system with multiple views and build editors following a model-driven strategy. Cicchetti [9] use a base multimodel for representing the system, and the views as subportions of the original base metamodel. A difference metamodel is derived from the original and views metamodels though ATL model-to-model transformations. A bidirectional model-to-model transformation is executed in order to convert model differences between the view models and the original model. An editor for modifying the view models is automatically generated by a QVT model-to-model transformation, which takes as input the created view metamodel. In our approach, we use a multimodel that integrates several views and the relationships among them in the multimodel. This approach differs with ours primarily in the views goal; the hybrid approach creates a view from a single-based meta-model with synchronization, while our proposal uses a multimodel that connects several views and its relations among them.

EMF Views [10] proposes a generic approach allowing to build views on any set of interrelated models that conform to potentially different metamodels. It provides a two-step approach that explicitly separates the specification of viewpoints from the realization and handling of corresponding views. A virtual metamodel, which is a metamodel whose (virtual) elements are just proxies to actual elements contained in other models, connects each viewpoint. The virtual metamodel is defined by using a Domain Specific Language (DSL) to create the proxies. Our infrastructure uses the multimodel metamodel in a similar way since it contains the proxies to the viewpoint model entities. However, the definition and edition of the multimodel metamodel in our proposal is based on the EMF standard tools instead of using a DSL language for editing the virtual metamodel.

MuVieMoT [11] proposes a set of domain-specific modelling languages for the specification and model-driven development of multi-view modelling tools. The starting point for the transformation is the modelling scenario model, which includes the specification of the meta models and the viewpoints. The Modelling Scenario is trans-formed into ALL code, which allows the immediate construction of an initial model-ling tool. MuVieMoT is focused on the generation of graphical editors for multi-view modeling tools. Our infrastructure has a different focus since it is oriented on the generation of general multimodel editors integrated in the EMF framework.

We analyzed several proposals that deal with systems with multiple views and provide editors for these systems. Our proposal differs with the mentioned proposals since it is totally integrated into the EMF-platform without the need to define any DSL language. In our infrastructure, the multimodel integrates the multiple views and its relations by using the EMF framework. Additionally, the editor code is generated by using the standard EMF generator and allows using our multimodel editor. This integration potentiates the interoperability with other EMF tools.

V. Conclusions & Further Work

In this demo, we provide an infrastructure with the aim to generate automatically multimodel editors. This infrastructure is based on an architecture that contains a metamodel for multimodels, which is composed of viewpoint models to represents the views, and its relationships. If the user creates a custom multimodel for a specific domain based on the proposed architecture, then infrastructure allows obtaining a multimodel editor through the EMF code generation facilities. In order to illustrate the applicability of the infrastructure, we implemented a multimodel editor that is able to use this code to edit instances based on this custom multimodel independently of the used views in the multimodel. Furthermore, it is possible to populate a viewpoint model from external models generated with external tools in a flexible way; since it is possible to include external metamodels that represents external views in the multi-model metamodel. Additionally, the tool allows establishing relationships among elements from the viewpoints. We can also support updating a view from an external tool with the integration of the *EMFCompare* plug-in in the editor. The proposed infrastructure gives a solution to the problem of generating multimodel editors in a flexible way with a low coupling between the model and the editor interface.

As future work, we intended to perform experimental validations applying our multimodel infrastructure in other domains to get feedback to improve the infrastructure. Finally, we are currently working in the definition of model transformations on ATL language to import the views in the multimodel editor, instead of transformations based on dynamic EMF.

Acknowledgment

This research was supported by the FPU program (AP2009-4635) from the Spanish Ministry of Education and Science, the Value@Cloud project (MINECO TIN2013-46300-R), and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008).
2. Barkmeyer, E.J., Feeney, A.B., Denno, P., Flater, D.W., Libes, D.E., Steves, M.P., Wallace, E.K.: Concepts for Automating Systems Integration. , Technical Report. National Institute of Standards and Technology. Gaithersburg, USA (2003).
3. Javier González-Huerta: Quality Derivation, Evaluation and Improvement of Software Arquitectures on Software Product Lines Development (In Spanish). (2014).
4. González-Huerta, J., Abrahão, S., Insfran, E.: Automatic Derivation of AADL Product Architectures in Software Product Line Development. Proceeding of the 1st International Workshop on Architecture Centric Virtual Integration (ACVI) Collocated with MODELS. pp. 69–78. Valencia, Spain (2014).
5. Gómez, A., Gómez Llana, A.: Model Driven Software Product Line Engineering: System Variability View and Process, (2012).
6. Object Management Group: Common Variability Language (CVL) OMG Revised Submission. (2012).
7. ISO/IEC: ISO/IEC 25000:2005 Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE, (2005).
8. Zhang, X., Haugen, Ø., Moller-Pedersen, B.: Model Comparison to Synthesize a Model-Driven Software Product Line. Proceeding of the 15th International Software Product Line Conference (SPLC). pp. 90–99. IEEE (2011).
9. Cicchetti, A., Ciccozzi, F., Leveque, T.: A hybrid approach for multi-view modeling. Electron. Commun. EASST. 50, (2012).
10. Bruneliere, H., Perez, J.G., Wimmer, M., Cabot, J.: EMF Views: A View Mechanism for Integrating Heterogeneous Models. Proceeding of the 34th International Conference on Conceptual Modeling (ER). pp. 19–22. , Stockholm, Sweden (2015).
11. Bork, D., Karagiannis, D.I., Fill, H.-G.I.: Model-Driven Development of Multi-View Modelling Tools: The MuVieMoT Approach. Proceedings of the 9th International Joint Conference on Software Paradigm Trends (ICSOFT-PT). pp. 11–23 (2014).