MTC Flow : A tool to design, develop and deploy model transformation chains *

Camilo Alvarez
Universidad de Los Andes
Bogotá D.C., Colombia
c.alvarez956@uniandes.edu.co

Rubby Casallas
Universidad de Los Andes
Bogotá D.C., Colombia
rcasalla@uniandes.edu.co

ABSTRACT

This paper presents a tool called MTC Flow, which allows model-driven developers to design, develop, test and deploy Model Transformation Chains (MTCs). The tool offers a graphical DSL for defining MTC workflow models independently of the technologies that support the transformations. Using basic concepts such as metamodels, models and transformations (M2M, M2T and T2M) the user defines, executes and tests his MTC easily in the same development environment. MTC Flow has an abstraction layer to implement technology support. It facilitates the interoperability of model transformation and validation using the existing technologies without changes. Additionally, once the MTC is finished, MTC Flow offers an option to deploy it in any environment that support JAVA technology. The tool supports modularity and alternative execution paths of the MTCs. It was built on top of the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF). The tool offers a development environment using the extending capabilities of the Eclipse platform. We illustrate how MTC Flow supports MTCs development process using an example.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environments; D.2.12 [**Software Engineering**]: Interoperability

General Terms

Model Transformation Chain, Interoperability

1. INTRODUCTION

Model Transformations Chains (MTCs) are crucial artifacts to put Model Driven Development (MDD)[3] approach in practice. MTCs are composed of different types of transformations designed to achieve specific tasks such as Model-to-Model, Model-to-Text or Text-to-Model generation. To implement these transformation types there are many different technologies. For example for Model-To-Model transformation we have ATL[1] [2], ETL [5] [2], QVTO[3] and for Model-To-Text exists ACCELEO[4], EGL[5], XPand[6] among others. Each of these technologies uses its own approach for model management and for transformation execution. Furthermore, generally they do not provide mechanisms to compose and interoperate with other transformation technologies or have a limited support.

Currently, there are some tools such as [10], [4], [8], [2] that partially support the required process to engineer a solution using the MDE approach. These tools dont offer an easy way to integrate diverse transformation technologies, neither a complete development environment in which common tasks such as project management, compilation and execution are already solved. This lack of support for the entire MTCs development process affects, in some extension, the adoption of the MDE approach and is the main motivation of our work.

This paper presents a tool called *MTC Flow* that allows model-driven developers to design, develop, test and deploy Model Transformation Chains (MTCs). The tool provides a graphical DSL to specify the MTC workflow model independently of the transformation technology. To achieve this technology independence, the DSL offers basic concepts of MDE such as Model, Metamodel and Transformation. MTC flow provides an execution engine that uses the model generated from the graphical DSL as an input and, using a Technology Manager Registry (TMR), delegates to the actual implementation the transformation execution. In addition to the standard transformations, MTC Flow proposes a transformation type called MTC that allow the composition of other MTCs in the workflow. This composition of MTCs brings modularity to MDE and offers the opportunity to reuse MTCs in multiple projects. Another important feature is the management of alternative execution paths using a Tag System to mark the elements in the MTCs design. These marks later will guide the MTC execution engine.

MTC flow is a tool set that extends the Eclipse platform.

*Instructions to install and use the tool are available at `http://www.mtcflow.org`

ACME '13, Montpellier, France

[1] http://www.eclipse.org/atl/
[2] http://www.eclipse.org/epsilon/doc/etl/
[3] http://wiki.eclipse.org/QVTo
[4] http://www.eclipse.org/acceleo/
[5] http://www.eclipse.org/edt/
[6] http://wiki.eclipse.org/Xpand

It offers support to MDE developers for the creation of MTC projects, a graphical editor based on GMF for the MTCs definition and Eclipse Extensions Points to include new transformation and validation technologies. There is also a facility for the developer to test completely or partially the MTC during development.

The rest of the paper is structured as follows. Section 2 presents a case study to illustrate the functionalities that MTC flow offers. Section 3 describes in detail the design of MTC Flow and section 4 presents its implementation. Next, section 5 presents a detailed comparison with related work and section 6 evaluates the tool using two sample projects. Finally, Section 7 concludes and discusses some future work.

2. USING MTC FLOW IN A CASE STUDY

The purpose of this section is to show, through a case study named WebMTC, the main services that MTC Flow tool offers to an MTC developer. The purpose of WebMTC is to generate automatically transactional web applications, implementing basic CRUD requirements, that run in the JEE Platform. To create a concrete application, the application designer creates a model (written in a textual DSL) specifying the business entities of the application. Then, WebMTC receives it as input and generates the source code and configuration files for the presentation, logic and persistence tiers of the corresponding application. To implement WebMTC, the developer has to create many metamodels and diverse transformations M2M, M2T, etc. An additional requirement is that WebMTC supports the generation of presentation and persistence tiers using diverse technologies; the application designer should be able to select the technology of the final components, for example, he could choose between Java Server Pages (JSP) or Java Server Faces (JSF) technology for the presentation tier.

2.1 Describing at a high-level the WebMTC

MTC Flow offers services to facilitate the creation of a new MTC Project in which all the artifacts; models, metamodels, transformation and the MTC definition will be contained. The MDE developer can prototype and validate their solution at high level. He can focus his efforts to meet the MTC objectives, not in finding how to connect or execute a particular transformation. Using the MTC Flow graphical editor the MTC developer designs his MTC workflow independently of any particular technology.

In our case study, the whole WebMTC is composed by three sub-MTCs one per tier. Figure 1 shows the workflow model of the WebMTC in a diagram created using the DSL editor. At the top of the diagram a business model is the input to the chain; then a Text-to-Model transformation generates a Business Model that conforms to the Business Metamodel. Using the MTC composition mechanism the main workflow splits into three different MTCs. Each MTC is responsible for the generation process of an application tier. At the bottom in the diagram is indicated the output files that each MTC generates. With this composition mechanism the MTC developers have the possibility to handle complex MTCs and the creation of reusable artifacts.

Figure 2 shows the WebMTC definition for the presentation tier. This MTC begins using a Model-to-Model transformation to generate the presentation model that conforms to the presentation metamodel. After the presentation model

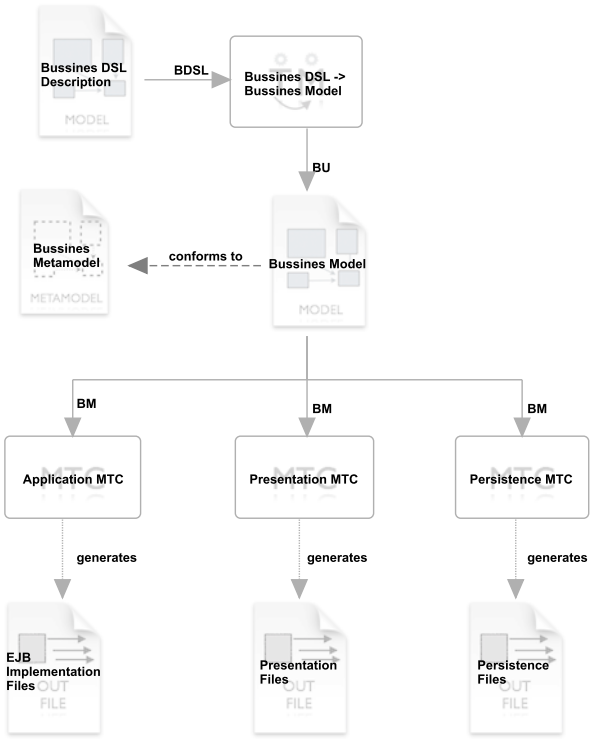

Figure 1: WebMTC General Diagram

is generated it is used as an input to another Model-to-Model transformation.

2.2 Defining alternative execution paths

One of the requirements for WebMTC is to offer multiple implementation technologies for the presentation tier. In this tier, WebMTC has two Text-to-Model transformations included. The first offers support for the generation of web interfaces using JSF technology while the second one generates JSP pages.

The tool uses the concept of *tags* to define alternative execution paths. At design level, the tags work as labels that could be applied to the artifacts of the MTC. In this case, the MDE developer creates a JSF and JSP tags and applies them to the JSF and JSP transformations respectively. These marks will allow the identification of the elements related to a specific technology. Section 3 presents a deeper explanation of tags.

2.3 Choosing technologies

MTC flow provides a development environment shown in figure 3. In the left, there is the WebMTC project structure that contains all the artifacts of the project. This includes transformations, validations, libraries, models, metamodels and the MTCs definitions. Thanks to the technology interoperability, it is not required to have a specific project for each transformation technology. MTC Flow is able to handle technologies that require to be compile like ATL; the tool executes the compilation process automatically to generate compiled versions. Another feature that the development and execution environment provides is the automatic registration of metamodels in the EMF Registry. In the center

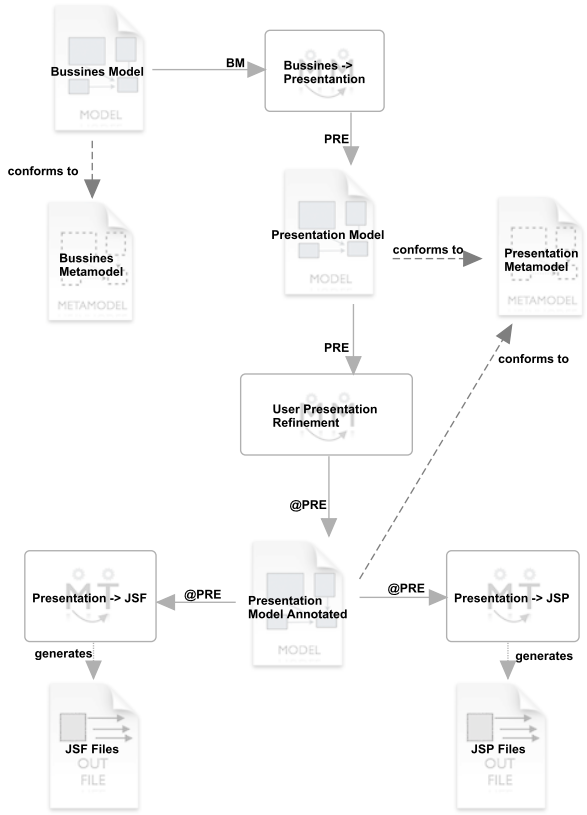

Figure 2: WebMTC Presentation Tier MTC

of the figure, the MTC Editor appears and at the bottom a console that shows information about the MTC execution.

Once the MTC developer has designed the WebMTC, he has to choose the most suitable transformation technology for each task. In the case study, the selected technologies were: 1) Xtext[7] for Text-to-Model transformation because it offers integrated support for eclipse and an easy way to transform the DSL text file to model. 2) ATL[2] for Model-to-Model transformations for the generation of the specific tier models. But we also use EOL[8] [6], because it is based in an imperative approach and allow the invocation of JAVA methods. 3) ACCELEO for Model-to-Text transformations because it offers code hints in the editor based on the metamodel elements. Regardless the decisions, the main idea is that MTC developers can use and integrate diverse transformation technologies. Now that the technologies have been selected, and the metamodels and model are available, the developer could start to create the required transformations.

2.4 Testing WebMTC

Developers can execute the entire MTC or by parts during the development process. First the developer has to relate each transformation in the diagram to a specific file that implements the transformation. This link will allow the execution engine to select and execute the concrete transformation. Then, the user can select any MTC Design File and using a contextual dialog, start the execution of the entire MTC or from a specific transformation. When an error occurs during the execution, the console will show the information that the transformation technology offers. Developers get instantly transformation testing and an overall overview of the development process using the partial execution functionality that the tool offers.

In this case study the MTC developer wants to offer to the application developer the possibility of configuring some aspects of the generation process of the tiers. He can do this because he can use any JAVA class and invoke it from any transformation technology that supports it. The execution engine allows the use of these classes in the development environment without starting a new virtual machine.

2.5 Deploying WebMTC

When an MTC is finished, in most cases, the target environment is not the same of development. This is the case for WebMTC where the developed MTC should be deployed to a standalone environment for working as a web service. The deployment of MTCs is a main concern in the MTC Flow tool. The developer just needs to package the project in a JAR file and then he can use the standalone implementation of the tool to execute it. The execution of MTCs outside the development environment is only limited by the support that the selected transformation technologies offer.

3. MTC FLOW: DESIGN

The Figure 4 presents a general overview of our approach. We identified two main actors in the process. The first is the *MTC developer*, which is responsible of the MTC specification and transformations implementation. The second actor is the application developer whose responsibility is to execute the MTC using the implemented artifacts.

[7]http://www.eclipse.org/Xtext/
[8]http://www.eclipse.org/epsilon/doc/eol/

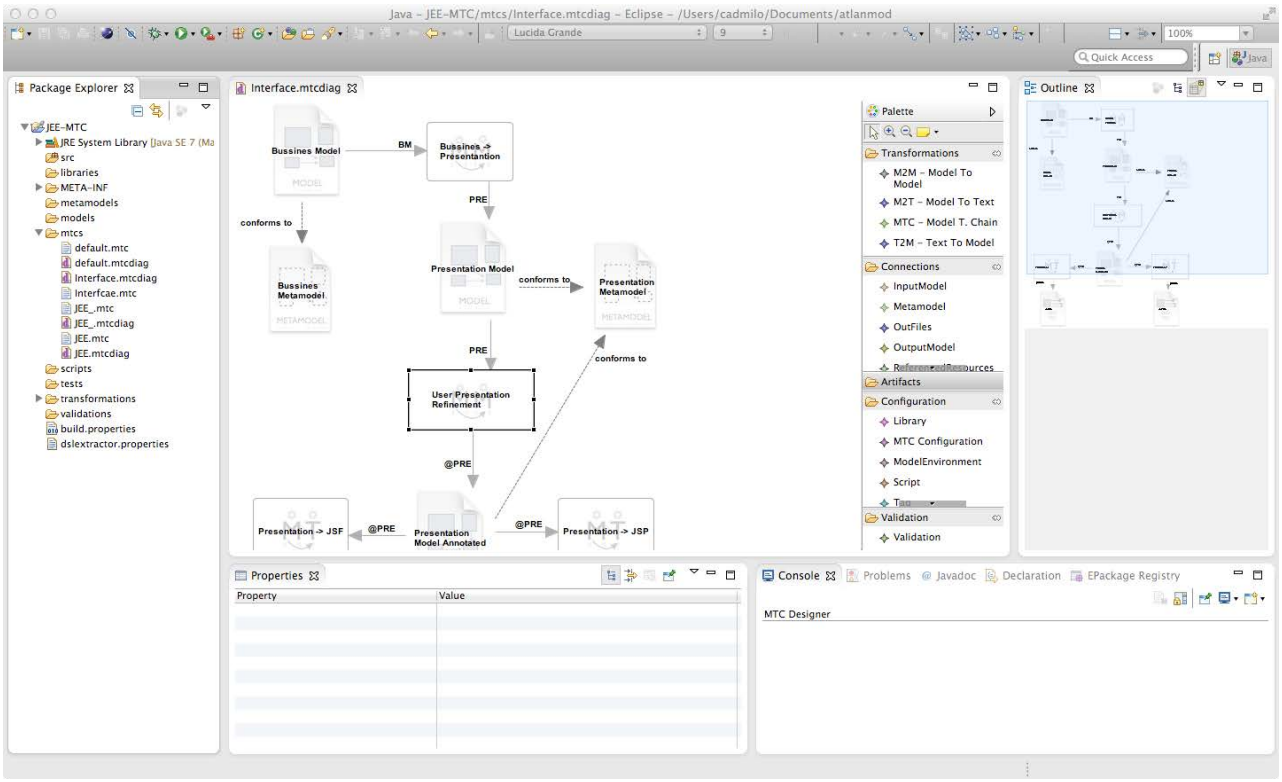

Figure 3: MTC Flow GUI

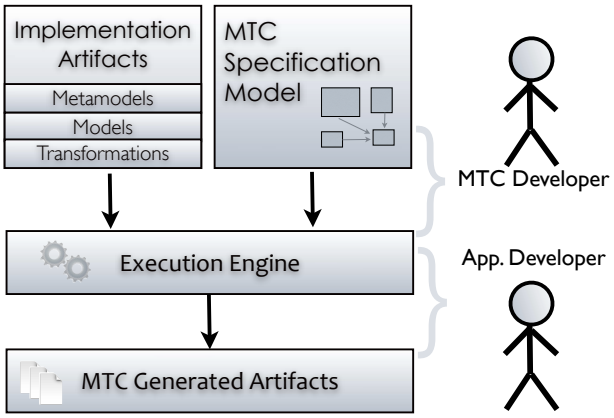

Figure 4: MTC Flow Overview

The development process starts when the MCT developer creates an MTC Specification Model that is independent of the transformation technologies. This model describes the execution workflow and it is created using the graphical DSL. After this, the MTC developer could create and test the artifacts that actually implement the task in the workflow; we call these artifacts *Implementation Artifacts*. These artifacts are metamodels, models and the implementation of each transformation in their specific technology. MTC Flow has an *Execution Engine* responsible of executing the MTC using the Implementation Artifacts and the workflow. Once the development process is completed, the application developer could use the engine to execute the MTC and obtain the outputs of the transformation processes. In the next paragraphs we detail the design of the main features that the tool offers.

3.1 Transformation Chains Specification

We propose a model-based approach to achieve the specification of MTCs independent of the transformation technologies. The main challenge to define the metamodel was to select the right level of abstraction. It must not only allow the definition of any MTC, but also have the important details to be used as an input for the execution engine. Our metamodel was inspired by many metamodels of management systems such as [9], [1]. As a result, we use a transformation classification based on the type of input and output handled by each transformation type. This classification allows the definition of Text-to-Model, Model-to-Model or Model-to-Text transformations.

Figure 5 shows an extract of the MTC Flow metamodel.

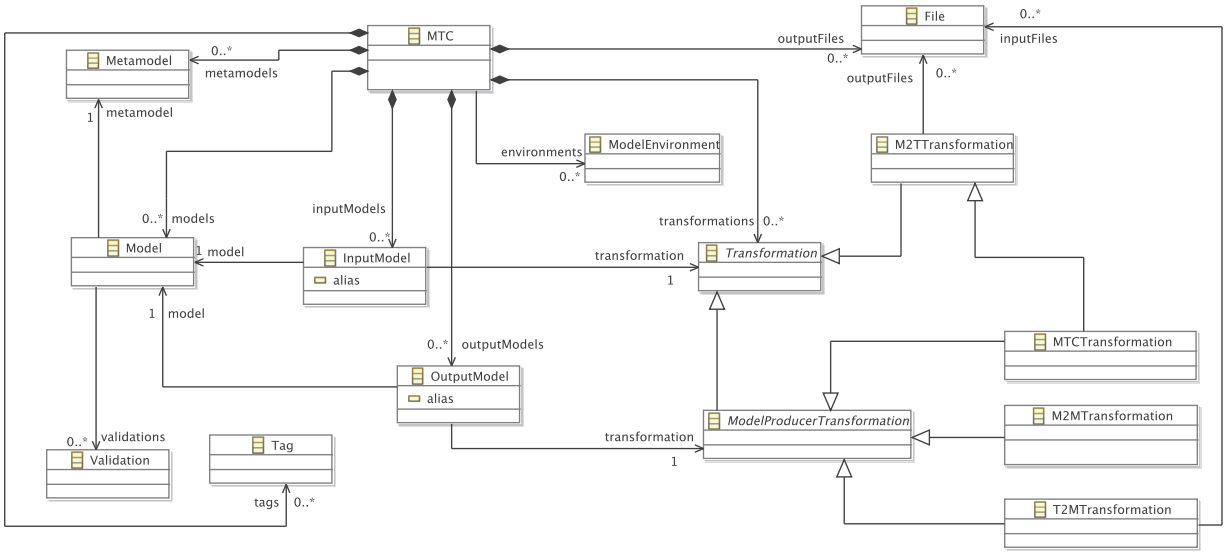

Figure 5: Extract of the MTC Flow Metamodel

The *MTC* concept works as a container for all the elements of any MTC. The concept *Model* has a reference to a *Metamodel* to which it conforms to. It also has a reference to the *Validation* concept. This relationship associates the model with validations that it should satisfy. The *Tag* concept could be associated with any element of the MTC to mark alternative execution paths. The structural elements of the MTC are represented in the *InputModel* and *OutputModel* concepts. Both have a reference to a *Model* and a *Transformation*. It also holds the alias attribute for the model. The *InputModel* concept indicates that the associate *Model* is an input for the referenced *Transformation* and the *OutputModel* indicates that the associated *Model* is generated by the *Transformation*.

In the metamodel, we have four different types of transformations: The first is *M2MTransformation* which represents all the transformations that receives models as input and generates models as output. The second is *T2MTransformation* that represents the transformations that handle textual files as input and generates models as output. *T2MTransformation* concept has a reference to *File* which represents the fileset that the transformation uses as input. The third is the *M2TTransformation* that represents the transformations that handle models as input and textual files as output. The last type is *MTCTransformation* which represents a reference to another MTC specification and is able to handle inputs and outputs of any type. This concept is very important because it allows the modularization and reuse of MTCs.

3.2 Workflow definition

The graphical language that MTC Flow offers to define the workflow of the MTC is pretty simple. It allows the designer to define transformation's sequences. Each activity in the diagram corresponds to a transformation implementation. For each activity, the designer can define the input and output models as well as their corresponding metamodels.

3.3 Alternative execution paths

MTC Flow offers alternative execution paths for the MTCs. During the development of complex MTCs is important to be able to handle multiple alternatives in the execution process. For example diversity in the target platforms or even at the architectural level are usual requirements. We propose a mechanism based on tags to handle different execution paths. This system is used in the design and execution of the MTCs. In the specification process the developer will create a set of *Tags* and then he will label the MTC elements with them. The developer should mark all the transformations related to a specific path. Later, These marks will guide the execution process. There is no restriction in the quantity of *Tags* or their combination.

At runtime a set of tags will be included as an input for the MTC. The execution engine will use these tags as follows: Before the execution of a transformation step, the engine validates that at least one of the specified tags is present in the transformation. If the tag is not present, the transformation will not be executed and as a consequence the subsequent steps either. In the other case, the engine will execute the transformation and continue the normal execution of the MTC. It is important to note that these tags could be modified at runtime by the transformations.

3.4 Extensibility and technology interoperability

MTC Flow allows the integration of multiple technologies using the *Technology Manager Registry* (TMR). The TMR holds the information about all the available technologies. It has a reference to a manager implementation for each technology. This manager works as a factory for the creation of transformations and validations executors, these are able to handle any transformation or validation of the technology. For example, the ATL Manager could execute any ATL transformation using their compiled file. Furthermore, when a technology, like ACCELEO, requires a compilation step before execution, it is able to handle it. The TMR is a key

service for the MTC Flow allowing a complete independence of the definition and implementation of transformations.

Besides the technology interoperability the TMR also offers a standard mechanism to extend the tool. When a transformation provider wants to add support for a new transformation technology it has to complete two steps. First, he must provide an implementation of the technology manager JAVA interface and second, he must register his implementation as a technology handler.

Sometimes the existing transformation technologies do not fit the developer requirements. In this case, the tool offers an option to specify a custom implementation of the transformations. This option allows the developer to integrate any existing library in the chain workflow. The developer only needs to create the implementation in the MTC project and ship it inside the MTC JAR. The execution engine will delegate the execution to the customized implementation.

3.5 Execution engine and standalone execution

As we said before, MTC Flow has an engine for the execution of the MTCs. This engine has two main objectives. First, it must allow the execution of any MTC and second it should support standalone execution. To accomplish these objectives the engine offers a complete abstraction of 3 key components. The TMR is the first component of the engine. It hides the technology interoperability problem and eases integration of new technologies. The second component is responsible for resource localization. For example, when the execution is made inside the Eclipse environment, the component can find the resources using an Eclipse Project but if the execution is done standalone, the component can find the resources in a folder structure. The last component is responsible for the Model Environment management. In MTC Flow a Model Environment refers to any environment that requires a configuration process before the MTC execution can proceed, for such as UML, EMF or a specific DSL implementation. This component is responsible for the setup of the environment as well as the required steps to handle additional resources. For example there is UML models that depends on UML Profiles; in this case the required profiles need to be loaded before the model could be use. The components implementation of the engine could be changed. This allows their usage in any JAVA environment.

4. MTC FLOW: IMPLEMENTATION

In this section, we present the main components of the MTC Flow tool and how we integrate it in the Eclipse platform. The figure 6 presents a component diagram of the MTC Flow architecture. In this image we show the principal components of the tool, which are described in detail in the next subsections.

4.1 The Core components

The first component is the *Core*; it is in charge of the model management and definition of the core interfaces. We use the EMF framework to implement MTC Flow metamodel. The *Model* subcomponent of the Core offers the required JAVA classes to load and manipulate the MTC models using the EMF APIs. The *Core Interfaces* subcomponent is responsible for the definition of the contracts that the components of the tool should implement. All the communication between components is based on the interfaces

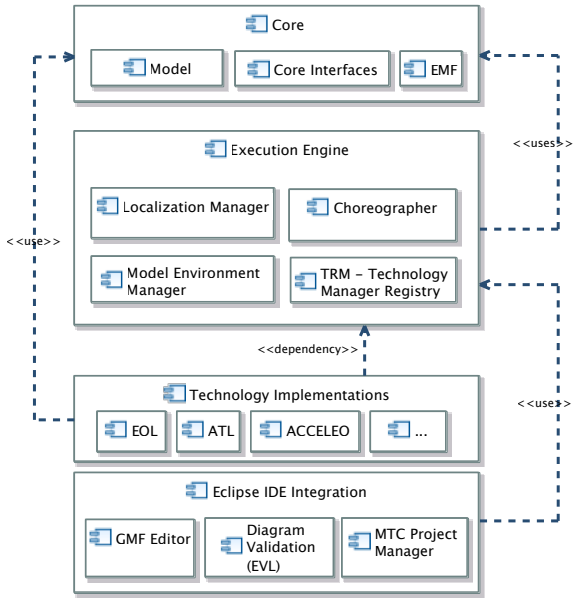

Figure 6: MTC Flow Architecture

defined by the *Core Interfaces*. This decoupling allows additions of new technologies for validation and transformation. Furthermore, it allows having multiple implementations of the components.

4.2 Execution Engine

The second component is the *Execution Engine*. It is responsible for the execution of the MTCs using the MTC Model and the implemented transformations. It has four subcomponents. The *Resource Locator Manager* is responsible to locate any resource (Metamodel, Model, Transformation, Validation) file of an MTC. Currently, we provide two implementations of this component. One could be used inside Eclipse and it is based in the *MTC Project* structure. The other locates the resources using the MTC JAR file. The second component is the *Model Environment Manager*. This component allows the management of the setup process that the modeling technologies require. We provide implementations to support EMF, UML and Xtext DSLs. Inside the Eclipse platform this component is not used, because the tools already handle this process in their implementation plugins.

Technology Manager Registry (TMR).

This is one of the most important components in the tool. It allows the integration of any transformation or validation technology without modification of MTC Flow or the existing technologies. The EMF Package Registry inspires the TMR implementation. In practice, it relates each technology file extension to a concrete Technology Manager Implementation. This implementation works as a factory to create new executors and to compile files related to the technology. When the engine is used in standalone mode the technologies should be registered invoking the TMR API. On the other hand, when the engine is used inside Eclipse we provide an Extension Point that allows the integration of technologies using Eclipse Plugins. This extension point

T2M	M2M	M2T	Validation
XTEXT	ATL	ACCELEO	OCL
	QVTO	EGL	EVL
	ETL		
	EOL		
	ECL		
	EML		

Table 1: MTC Flow Technology Support

brings technologies interoperability to the development environment and eliminates the registration process. Using this extension mechanism we provide support for the major technologies that are shown in table 1.

The fourth subcomponent is the *Choreographer*, it is responsible for the management of the execution process. In general, the choreographer will start a transformation when all the needed models for execution are ready. During development, the component helps the developer to test, a partial sequence of the MTC, in the development environment. The *Choreographer* creates all the transformation and validation executors using the TMR. When the *Choreographer* receives an event that indicates its input models are ready, it searches all the transformation executors that have these models as an input and notify them that the model is ready. After this, it will ask to the transformation executor if it could star. If it is ready, the engine will execute the transformation in a new thread. The transformation executors are responsible for notifying the engine when a new model is generated.

4.3 Eclipse IDE integration

The Eclipse IDE Integration component offers three main subcomponents that are created using the extension mechanism of the Eclipse platform. Using EuGENia[9] [7] we generate a graphical editor for the GMF framework. This editor allows the developer to create models that conform to MTC Flow Metamodel. In addition to the basic model creation we add features to locate the resources from the diagram. For example, when the user makes double-click inside a transformation element in the diagram it automatically open the linked transformation file with their corresponding editor. Using the extension point that EVL [10] offers, we create a validation component that verifies the correctness of the model directly in the graphical editor. This validation displays feedback to the user during the MTC definition process.

MTC Project Manager.

Currently, most of the transformation technologies require the creation of a project to work with its own transformations. This fact creates a problem for the developer who has to handle many different types of projects. To solve this problem, we create a new Project Nature called *MTC Nature* that provides wizards for the creation of the project and, by default, offers a structure to organize the MTC Artifacts. This project nature has associated a project builder, which helps the developer to achieve two tasks. First, it is able to automatically register ECORE metamodels and second, using the TMR, it is able to compile the transformation in the project. Another important feature of the Eclipse support is the integration of a contextual menu that allows the developer to execute any MTC Model or Diagram File.

[9]http://www.eclipse.org/epsilon/doc/eugenia/

[10]http://www.eclipse.org/epsilon/doc/evl/

5. RELATED WORK

There are different tools that support a wide variety of requirements such as technology interoperability, composition, and execution. Vanhooff et al[10] propose a model-based solution and implement it in a tool called UniTI. The main concern of this approach is to offer technology interoperability using a clear separation of transformation specification and execution. Kleppe[4] presents a model transformation environment focusing in the integration of transformations tools. This environment is implemented in a tool called MDA Control Center (MCC). MCC also defines a language-based taxonomy for model transformation applications. Rivera et al[8] proposed and implement an engine for the design of MTCs limited to the execution of ATL[2] transformations. They present a graphical language for the workflow definition and execution in a tool called Wire*. Another approach for modeling workflow is the Modeling Workflow Engine (MWE)[11]. MWE is an eclipse project that uses an xml-based language to support workflow definition using a dependency injection framework. This engine offers a tool set that supports the edition, execution and debugging of workflows.

Table 2 summarizes a comparison of the main characteristics of these tools and MTC Flow. We present the comparison in some detail in the next items.

- Definition language and design support: All of the tools offer a language to define the MTC workflow; however MTC Flow language is graphical and it is completely independent of any transformation technology. UniTI, Wires and MTC flow provides an editor to design the workflow.
- Technology extensibility support: UniTI, MWE and MTC Flow were design extensible, i.e., new technologies can be integrated in the environments. To extend MCC a development process is required and Wires only accept ATL transformations.
- MTC Modularization and reuse support: Regarding the tools in the table, MTC Flow is the only one to support the modularization and reuse of complete MTCs.
- Alternative execution path support: This characteristic refers to the possibility of configuring diverse execution paths of the workflow. MTC flow facilitates this by means of tags that the developer can define and configure.
- Standalone environment support: Only MWE and MTC Flow support this feature. It means, we can execute outside the development environment the created MTC.
- Validation support: Only MTC Flow provides this feature. We can add to the workflow tasks to execute validations on the resulting models in the MTC.

Regarding usability and completeness of our tool, MTC Flow provides *Project development support* and *IDE integration support*. The MTC developer can create MTC projects,

[11]http://projects.eclipse.org/projects/modeling.emf.mwe

Table 2: Related Work Comparison

	UniTI	MWE	MCC	Wires*	MTC Flow
Definition Language and design support	Textual	Textual	Textual	Graphical	Graphical
Technology extensibility support	X	X	manual	Limited to ATL	X
MTC Modularization and reuse support	-	-	-	-	X
Alternative execution path support	-	-	-	-	X
Standalone environment support	-	X	-	X	X
Validation Support	-	-	-	-	X
Project development support	-	-	-	-	X
IDE integration support	-	X	-	-	X

just one project per MTC instead of one project for each technology. MTC Flow does registration of metamodels automatically. Furthermore, it facilitates the integration with the IDE providing extension points to add support for transformation technologies.

6. EVALUATION

In this section, we evaluate MTC Flow using two final projects of the Model Driven Development course[12]. This course prepares the students to apply the MDE approach; they learn concepts such as Models, Metamodels and Transformations and develop skills to implement them. To achieve the evaluation, the students implement their final projects using the MTC Flow tool. We apply a qualitative survey focused in the experience during the development process.

The first project called LINPRO allow users to express linear programming mathematical models using a textual DSL and then it transforms the models to executable code in the AMPL[13] and GAMS[14] formats. The second one called MMUSIC offers an editor to express music tracks using a simplified version of the English Musical Notation system, then it generates the correspondent scores in MusicXML[15] format, tablature in HTML format and a playable file in MIDI[16] format. Both projects required different types of transformations: Text-to-Model, Mode-to-Model and Model-to-Text and integrate multiple technologies specifically XTEXT, ATL and ACCELEO.

The students received a basic introduction to the tool. After this, they implemented their projects using MTC Flow. Once the implementation was completed we ask them to complete a survey about their experience. The topics of this survey were:

- The installation process
- The graphical editor usage
- The expressive power of the graphical DSL
- The development experience in general
- Troubles using the tool
- Improvement opportunities

In the results of the survey the users describe the tool as easy to use and helpful in the development process. In both cases the users could complete the installation process normally. They report that the graphical editor was easy to use and intuitive. Both projects could describe their MTC Workflows without problems and the users acknowledge the utility of executing parts of the MTC during the development. They remark, the elimination of the required code to connect the transformations. In general they report a faster and more productive development process compared to their previous experience with the course projects during the semester.

The users report two main issues about troubles and improvement opportunities. First, all agree that there is a requirement for a better documentation about the tool. We plan to include screencasts for common scenarios and detailed documentation in the project website. The second issue is the properties view of the graphical editor. The flat-list that is presented with all of the properties that an element can have, confuse some users. This view is the default implementation generated by the GMF project. To tackle this issue, we plan to use the recently accepted project by the eclipse community called Extended Editing Framework (EEF)[17] to redesign the properties view in a more appealing way.

7. CONCLUSIONS AND FUTURE WORK

MTC Flow is a tool intended to be use by MTC developers. These developers have to create metamodels, diverse types of transformations, models and other artifacts to complete an MTC that actually implements a model driven solution. Usually, these solutions require many and diverse technologies that have its own model management and execution process, and they are not expected to interoperate with other technologies. The lack of development environments that support the whole model driven development process, and the complexity to integrate and use the technologies make difficult the adoption of the MDE approach.

The main contributions of MTC Flow are: 1) *Technology independence*: a high-level graphical language to describe the workflow of the MTC. 2) *Technology interoperability*: a Technology Manager Registry (TMR) to integrate any transformation and validation technology. Together with the *Technology independence* it allows the interoperability of multiple technologies. 3) *Alternative execution paths*: a mechanism based on tags makes possible to specify and decide a path of execution in the MTC. This feature is important because usually an MTC developer wants to offer variability in the generation process. 4) *Stand-alone execution*: MTC flow is an extension of Eclipse; however, a concrete model-driven solution can be deploy outside Eclipse in any java environment. This facilitates the portability of the

[12]http://sistemas.uniandes.edu.co/isis4718
[13]http://www.ampl.com/
[14]http://www.gams.com/
[15]http://www.musicxml.com/
[16]http://www.gams.com/
[17]http://www.eclipse.org/modeling/emft/?project=eef

developed solutions.

In future work we plan to include flow control elements into the MTC Workflow, like loops and conditionals. Currently, we are working on a Cross-Technology Debugger based on the existing debuggers of each technology, and similar, to improve performance of the MTCs, the developer needs a Cross-Technology Profiler that allows him to identify shortcomings during the process. Cache Support will be also a useful feature to improve model management.

8. REFERENCES

[1] J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. Modeling in the large and modeling in the small. In *Model Driven Architecture*, volume 3599 of *Lecture Notes in Computer Science*, pages 33–46. Springer Berlin Heidelberg, 2005.

[2] F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1-2):31 – 39, 2008.

[3] S. Kent. Model driven engineering. In M. Butler, L. Petre, and K. Sere, editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin Heidelberg, 2002.

[4] A. Kleppe. Mcc: a model transformation environment. In *Proceedings of the Second European conference on Model Driven Architecture: foundations and Applications*, ECMDA-FA'06, pages 173–187, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] D. S. Kolovos, R. F. Paige, and F. Polack. The epsilon transformation language. In *ICMT*, pages 46–60, 2008.

[6] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. The epsilon object language (eol. In *In: Proceedings European Conference in Model Driven Architecture (EC-MDA) 2006*, pages 128–142. Springer, 2006.

[7] D. S. Kolovos, L. M. Rose, S. B. Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. Taming emf and gmf using model transformation. In *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part I*, MODELS'10, pages 211–225, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] J. E. Rivera, D. Ruiz-Gonzalez, F. Lopez-Romero, J. Bautista, and A. Vallecillo. Orchestrating ATL model transformations. In *Proc. of MtATL 2009*, pages 34–46, Nantes, France, July 2009.

[9] R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh, and P. Viriyakattiyaporn. An eclipse-based tool framework for software model management. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, eclipse '07, pages 55–59, New York, NY, USA, 2007. ACM.

[10] B. Vanhooff, D. Ayed, S. Van Baelen, W. Joosen, and Y. Berbers. Uniti: a unified transformation infrastructure. In *Proceedings of the 10th international conference on Model Driven Engineering Languages and Systems*, MODELS'07, pages 31–45, Berlin, Heidelberg, 2007. Springer-Verlag.