Migration to Model Driven Engineering in the Development Process of Distributed Scientific Application Software

Raphael Gayno
IFP Energies nouvelles
Rueil-Malmaison
France
raphael.gayno@ifpen.fr

Jean Marc Gratien
IFP Energies nouvelles
Rueil-Malmaison
France
j-marc.gratien@ifpen.fr

Goulwen Le Fur
OBEO
Nantes
France
goulwen.lefur@obeo.fr

Daniel Rahon
IFP Energies nouvelles
Lyon site
France
daniel.rahon@ifpen.fr

Sébastien Schneider
IFP Energies nouvelles
Lyon site
France
sebastien.schneider@ifpen.fr

Abstract

For several years now the IFP Energies nouvelles (IFPEN) group has been developing the OpenFlowSuite, a software suite in the oil and gas domain based on Eclipse RCP, incorporating graphical components and parallel calculators. These calculators are themselves developed in Fortran or C/C++. The processing chain “data entry", “database persistence", “calculator input", "execution” and “result processing” entails mapping between models and requires the development and maintenance of complex communication code. The progress made in recent years in the field of model driven engineering, and the accompanying Eclipse tools, led us to consider introducing these solutions in management of the communication code. In this article we describe the introduction and use of Model Driven Engineering (MDE) Eclipse tools in this context of industrial development of distributed scientific applications.

Categories and Subject Descriptors D.2.11 [**Software Engineering**]: Software Architectures – data abstraction, domain-specific architectures

General Terms Performance, Design, Reliability.

Keywords Modeling; code generation; Eclipse; numerical simulation.

SPLASH'12 October 19–26, 2012, Tucson, Arizona, USA.

1. Introduction

The IFPEN group is a major actor in the software market of oil & gas reservoir simulation and characterization and basin modelling with its OpenFlowSuite software. The applications in OpenFlowSuite incorporate the results of scientific research in the form of complex and high performance computation models held in the OpenFlow platform in 3D graphical applications based on Eclipse [1].

OpenFlow is a platform composed of a business data model. It is used to create, import, modify and store the different models of reservoir and basins studied and all the information necessary for the simulation of physical phenomena via the calculators. The computations may be distributed on a local network and the results are available in real-time for analysis and post-processing by oil reservoir engineers or geologists. The integration and execution of Fortran or C/C++ scientific calculators requires the definition of protocols for communication and exchange with the OpenFlow platform. These exchanges are accompanied by model transformations to map the OpenFlow “user” oriented model to the calculators' "simulation” oriented structures. All this technical code, initially handled manually, is complex to implement, develop, optimize to avoid impacting computation performance and maintain, especially in terms of integrating any upgrades and modifications to the OpenFlow platform.

This situation led us to consider the introduction of solutions to automate - as far as possible - development of this technical code for mapping and communication. The sever-

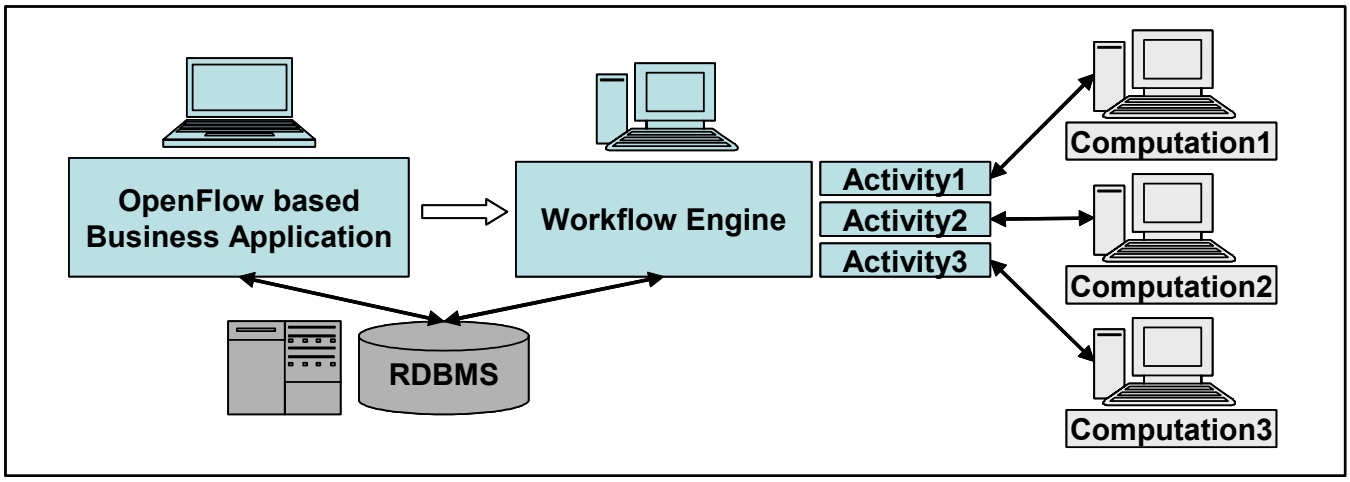

Figure 1. OFS outline diagram

al work produced in MDE [2, 3] implementation in the development of business application in the recent years [4, 5, 6, 7]. Our application case is complementary, since it involves a context of integration between a Java client application, where MDE most commonly applies, and Fortran/C/C++ calculators, for which this approach is more original.

In addition, to facilitate testing and calculator integration, data entry editor generation solutions have been studied and proposed. This article therefore describes a complete approach to the use of MDE and code generation, with the aim of simplifying the integration and use of calculators from a graphical client.

2. The Situation

2.1 OpenFlow/Calculators communication

OpenFlow consists of a graphical application based on Eclipse-RCP, with plugins for data management, data edition, multiple scientific plots and graphics, data analysis, etc. OpenFlow includes a concept of batch executable scientific business activities which can be grouped together to form workflows in the field of exploration and characterization of oil reservoir and basin modeling. These business activities may have an embedded simulator or scientific calculator which is executed when the workflow execution is requested by the user. Simulations are executed on a network of machines and integration takes place via the database, centralizing data, and the workflow engine chaining activities.

The **Figure 1** shows the main components of the OpenFlowSuite including the graphical client (Business Application) used to manage data, interactive treatments and workflows definition. The Workflow Engine handles activities execution on a local network.

Both data and activity management system are written in Java. The calculators are complex scientific components developed in Fortran or C/C++.

To input data to a calculator, manage its remote launch, retrieve its results and monitor its execution, the platform uses a dedicated component, OpenDataServer (ODS). As illustrated on **Figure 2** the ODS main tasks are to start the calculator, open a network communication channel with it and respond to its requests for transmission of data and reception of results.

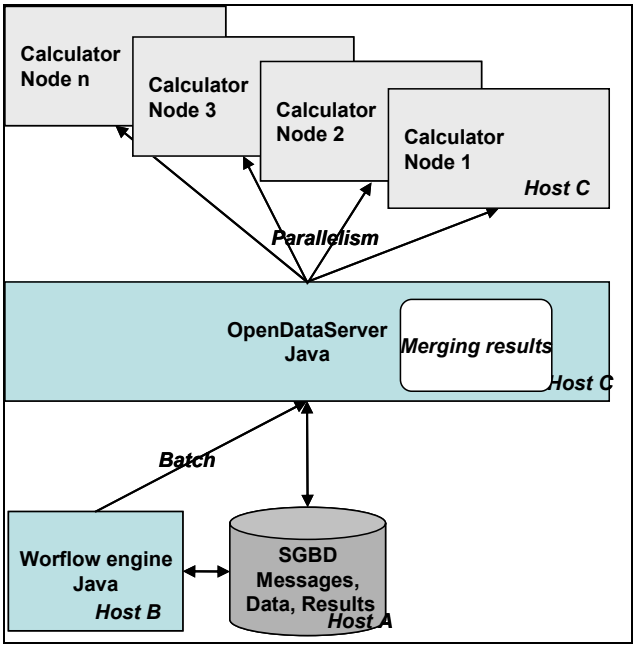

Figure 2. OpenDataServer component

In addition, ODS had to meet some design goals and constraints:

1. To provide an efficient communication solution to avoid creating a bottleneck for the calculator and keep network transfers to a minimum;
2. To process clients for parallel executions. This constraint entails the ability to adapt the data sent/received depending on the number of processors;
3. To monitor execution: computation client logs, errors and exceptions report accessible by the graphical client and the ability to stop, pause or restart the computation.

Finally the solution should be based on an open, standardized, scalable architecture to facilitate the connection of new calculators even if they use a different communication system.

This set of constraints led to the construction of a fairly complex technical component written in Java to manipulate the OpenFlow data model, using a low level communication socket to connect to Fortran or C/C++ tools. It is based on a XML dictionary defining the client/server communication protocol. Also, mapping between the OpenFlow model and the calculator models is sometimes performed by server side services, other times by client side services when processing messages are received.

2.2 A simple example

To better illustrate this problem of model transformation and system maintainability, we consider a simple example modeling the surface facilities used in reservoir simulation to compute properties of oil and gas phases at surface condition. This example is based on:

- The Java class *Separator* in the OpenFlow data model with two attributes modeling the temperature and pressure parameter of the business concept of separator;

```
public final class Separator {
   // Temperature in separator
   private double temperature = 0.0;
   // Pressure in separator
   private double pressure = 0.0;
}
```

- A C++ class *Process* that store in the reservoir calculator the number of separators and their parameters and a C++ function that load the Separator parameters to instantiate that class;

```
class Process {
 private:
   int m_nb_of_separator;
   double* m_temperature;
   double* m_pressure;
};

private void loadProcess() {
 // Get DataServer
 ods = OpenDataServer::GetOpenDataServer();
 // Send message to load data
 ods->Write("LoadAllSeparatorData");
 // Mapping starts here
 int nbSep = 0;
 ods->Read(&nbSep );
 double* p = new double(nbSep);
 double* t = new double(nbSep);
 for(int i=0; i< nbSep ; i++){
   ods->Read(p);
   ods->Read(t);
   p++; t++;
 }
 Process proc = new Process(nbSep, p, t);
}
```

- The XML file that drives the services enabling the mapping between the *Separator* model in the database and the *Process* structure use in the calculator.

```
<services>
<!-- All methods in the internal section are
only used by the server side -->
  <internal>
   <method name="getTemperature"
    class="data.model.Separator"
    type="double">
   </method>
   <method name="getPressure"
    class="data.model.Separator"
    type="double">
   </method>
   <macroSequence id="SendSeparator" >
    <alias tar-
get="data.model.Separator.getTemperature "/>
    <alias tar-
get="data.model.Separator.getPressure"/>
   </macroSequence>
  </internal>

<!-- All this services can be directly called
by the client side -->
  <external>
   <macroSequence id="LoadAllSeparatorData"
    collection="GetAllSeparator">
    <alias target="SendSeparator"/>
   </macroSequence>
  </external>
</services>
```

This simple example illustrates the different components involved in the data transfer from the Java based persistent data model to the C++ calculators. C++ clients have only access to the <external> services declared in the XML dictionary. Internal mechanisms of the OpenDataServer transform external requests in access to the Java data model.

2.3 The current state

The ODS component is developed and maintained entirely manually, giving rise to problems. System complexity was a major difficulty, which has made fixing bugs harder. Maintenance of dictionaries has also been hard, especially because the data model has been changing frequently to satisfy new needs. It has also been difficult to transfer certain business knowledge from the calculators to the server services to facilitate mapping.

In our simple study case, if we enriched the surface facility model and added a new parameter `maxRate` we need to update manually:

- The *Separator* Java class to add the new attribute;

```
public final class Separator {
   // Temperature in separator
   private double temperature = 0.0;
   // Pressure in separator
   private double pressure = 0.0;
   // Max allowed rate in separator
   private double maxRate = 0.0;
}
```

- The C++ *Process* class and the load function;

```
class Process{
 private:
   int m_nb_of_separator ;
   double* m_temperature;
   double* m_pressure;
   double* m_max_rate;
};
private void loadProcess() {
 // Get DataServer
 ods = OpenDataServer::GetOpenDataServer();
 // Send message to load data
 ods->Write("LoadAllSeparatorData");
 // Mapping starts here
 int nbSep = 0;
 ods->Read(&nbSep );
 double* p = new double(nbSep);
 double* t = new double(nbSep);
 double* r = new double(nbSep);
 for(int i=0; i< nbSep ; i++){
   ods->Read(p);
   ods->Read(t);
   ods->Read(r);
   p++; t++; r++;
 }
 Process proc = new Process(nbSep, p, t, r);
}
```

- The XML dictionary file to ensure the mapping of the transformed model;

```
<services>
<!-- All methods in the internal section are
only used by the server side -->
  <internal>
      <method name="getTemperature"
    class="data.model.Separator"
    type="double">
   </method>
   <method name="getPressure"
    class="data.model.Separator"
    type="double">
   </method>
   <method name="getMaxRate"
    class="data.model.Separator"
    type="double">
   </method>
   <macroSequence id="SensSeparator" >
   <alias tar-
get="data.model.Separator.getTemperature "/>
  <alias tar-
get="data.model.Separator.getPressure"/>
  <alias tar-
get="data.model.Separator.getMaxRate"/>
   </macroSequence>
  </internal>
  .....
</services>
```

These observations and the development of MDE tools around the Eclipse platform led us to consider a code generation strategy for part of the server functions. The introduction of MDE is also accompanied by a stricter definition of the server role, particularly in terms of model transformation.

3. Tools

The tools used to introduce MDE in the development of OpenFlow elements are Eclipse project tools. Eclipse RCP is our application support platform and development tool. It is therefore logical for us to use tools integrated into our environment: Eclipse EMF, EcoreTools for modelling, Acceleo for code generation and EEF for SWT editor generation.

3.1 EMF

Eclipse Modeling Framework (EMF [8]) is a set of modeling tools used to simplify code generation for building tools based on data models. These tools are widely used to develop applications for developers and are now increasingly used in business applications (TopCased, for example [9]).

3.2 Ecore Tools

Use of the EMF project entails definition of the data model to be manipulated by business applications. This definition is based on the Ecore formalism provided by the EMF project.

Ecore Tools offers components for management of Ecore models: graphical construction, editing, maintenance and validation.

3.3 Acceleo

Acceleo [10] is an open source code generator from the Eclipse Foundation, used to implement the model driven approach for building applications from models. It is an implementation of the standard from the Object Management Group (OMG) [11]for model to text transformation.

The language used by Acceleo is an implementation of the standard MOF MTL (Model to Text Language). This code generation language uses a template based approach. With this approach, a template contains static and dynamic text fields to define the code to be generated. The dynamic text fields allow information to be extracted from the model by defining expressions navigating the entities and properties of this model. Within Acceleo, these expressions are based on OCL language.

Acceleo contains a code generation modules editor with syntax highlighting, completion, error detection and refactoring.

4. Integration of the New Solution

The main structural change is replacing part of the code produced and processed manually in OpenDataServer and client-side (**Figure 3**) by generated components based on a communication model. The **Figure 3** details in the internal square the different elements on both server and client side that compose the communication solution: the XML dictionaries expose available messages which are processed by OpenDataServer to access persistent data model. Then a first model transformation is done to send data as messages to the client where messages are decoded to recreate data structures used by the simulator. All bricks are manually developed and maintained.

The solution proposed for the new architecture is based on MDE EMF concepts and on the Eclipse tools used to implement them: EcoreTools and Acceleo.

The target architecture is therefore based on a model of communication between the data server and the client calculators. This model allows use of code generation tools and a much better separation between the communication layer and upgrades that may occur on the OpenFlow platform and its persistent model.

The principles applied in the target solution and illustrated **Figure 4** are as follows:

1. Definition of a communication data model and implementation as an Ecore model;

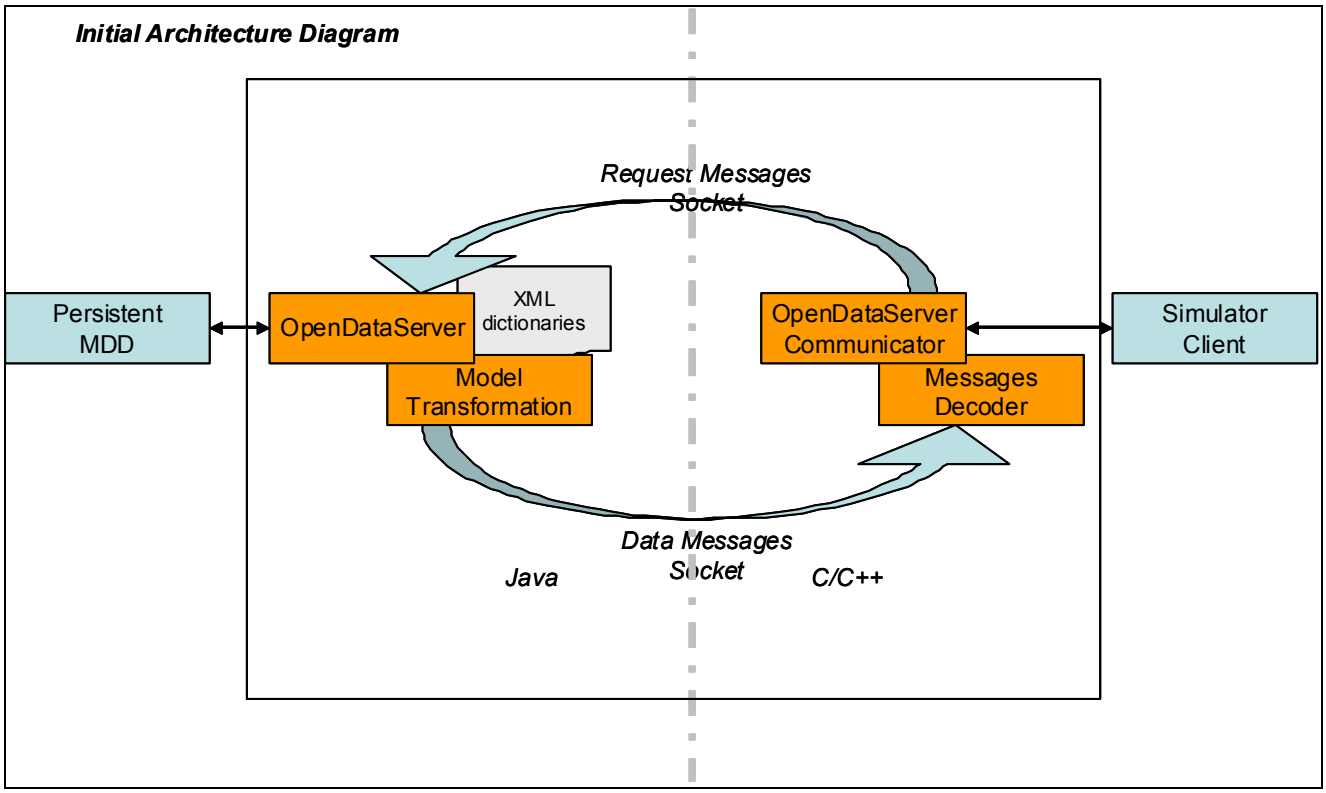

Figure 3. Existing architecture

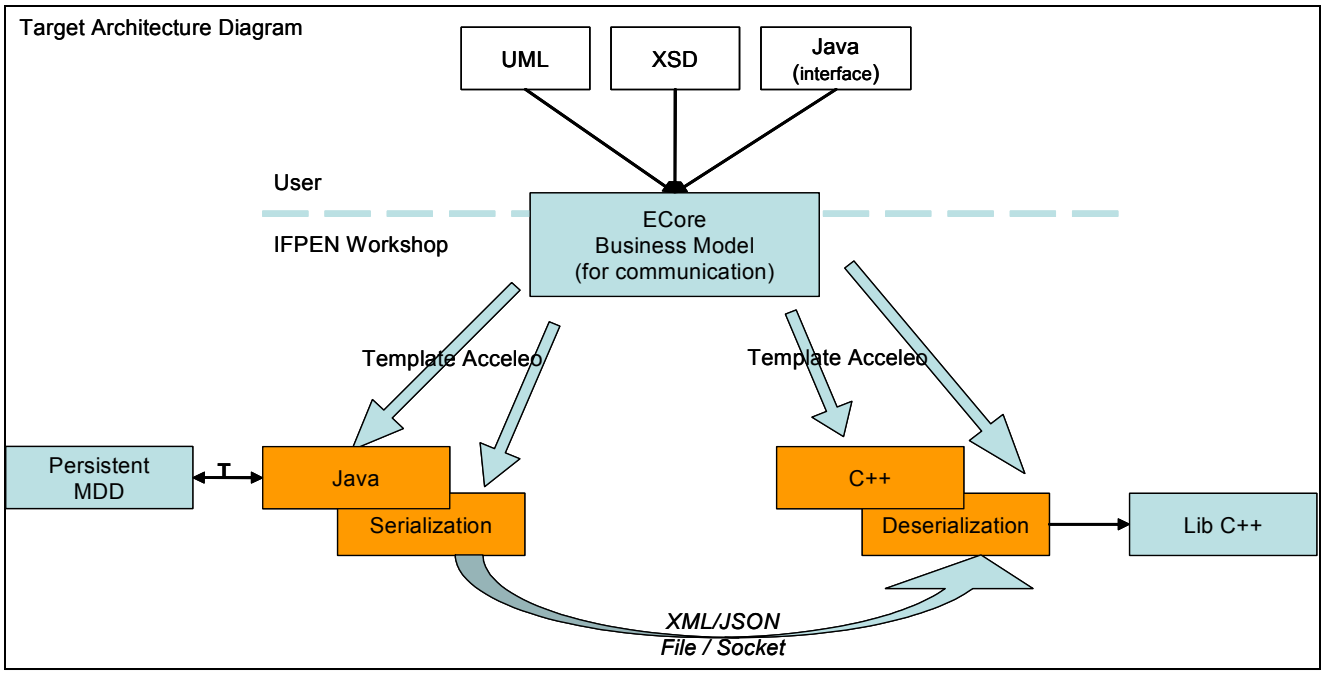

Figure 4. Target architecture

2. Generation of the Java code of this communication model with an Acceleo based generator;
3. Serialization of the communication model with a class also generated with Acceleo.
   Deserialization of the communication into a C++ model. This is done by a C++ class generation with Acceleo;
4. From the calculator, access to data via the C++ model getters. The XML dictionary is no longer necessary;
5. No business knowledge will be put in the communication section. Optimization and transformation for adaptation to the calculator can be done via construction of an appropriate communication model.

For example, with our small case study, we work with a EMF model of *Separator*. With our new solution, the Java class is generated and there is no more XML dictionary. There is a new generated C++ *Separator* class used by the `loadProcess` function as in the following listing.

```
private void loadProcess() {
 // Get DataServer
 ods = OpenDataServer::GetOpenDataServer();
 // Send message to get data
 ods->Write("LoadAllSeparatorData");
 std::vector<Seperator*> separators;
 // Creates C++ objects
 ods->load(separators);
 // Do complete mapping in C++ on client side
Process proc = createProcessFromSepara-
tors(separators);
}
```

Any modification in the persistent data model leads to the generation of new version of the Java and C++ classes. At this stage, the only manual edition should be realized in the mapping function on the client side.

5. Implementation – validation

5.1 Organization in projects – plugins

Remember that the final goal is to provide the developers with an IFPEN work environment so that they can develop efficiently an OpenFlow – Calculators data exchange.

In order to have generation functions in the work environment we therefore need:

- An Eclipse version with environment for EMF modeling and Acceleo 3;
- An Eclipse plugin with Java code generation templates and serializer;
- An Eclipse plugin with C++ code generation templates and deserializer;
- An Eclipse plugin with the generation actions and wizard for selection of elements to be generated.

Based on this work environment, the actions to be performed by the developer to develop the OpenFlow - Calculators exchange are as follows:

- Define a new plugin project;
- Construct an Ecore communication model. This model corresponds to the data that will actually be communicated to the calculator;
- Apply Java, C++ generation;

- Integrate the Java plugin generated in OpenDataServer via the extension point;
- Link the C++ generated libraries including the model and the deserializer with the calculator;
- Complete the calculator code to request the data and perform mapping to its own model.

5.2 Implementation

This involves implementing the design elements listed above for an existing calculator operating on the initial architecture. This implementation applies to new business entities to be communicated to the calculator. It is therefore a matter of getting the two architectures to co-exist in order to gradually upgrade the initial solution to the target solution.

The following assumptions are made to ensure this co-existence:

- In the first steps, to validate operation, we assume that the communication model and the persistence model are similar. To manage the differences we semantically annotating the data model. Annotations are taken into account during the generation process;
- The co-existence in the data server of two modes "with dictionary" and "without dictionary" will be handled by a special branch. The idea is to configure the project with a key helping the server to choose which branch to use.

Based on these assumptions, the Acceleo templates for generation from the Ecore model are developed together with the GUI action for generation. Taking the assumptions into account, only the templates for generation of the Java serializer in XML, the C++ model and the C++ deserializer are developed. A Java-Json serializer is also tested for reasons of compactness. These plugins are deployed as an IFPEN toolkit and integrated in Eclipse for tests.

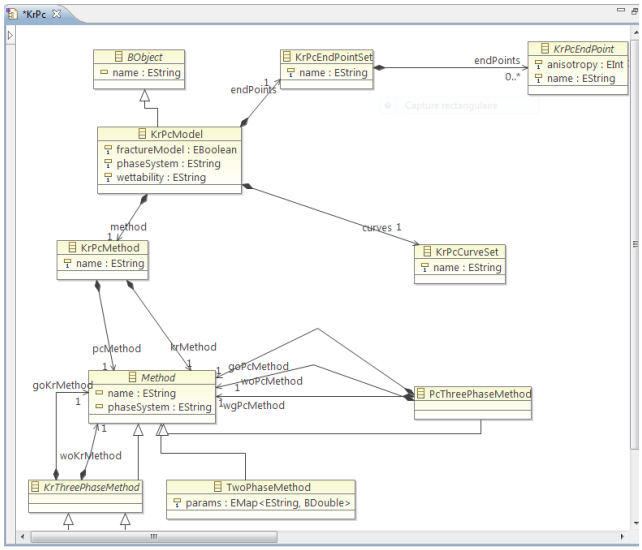

Figure 5. Ecore model for validation

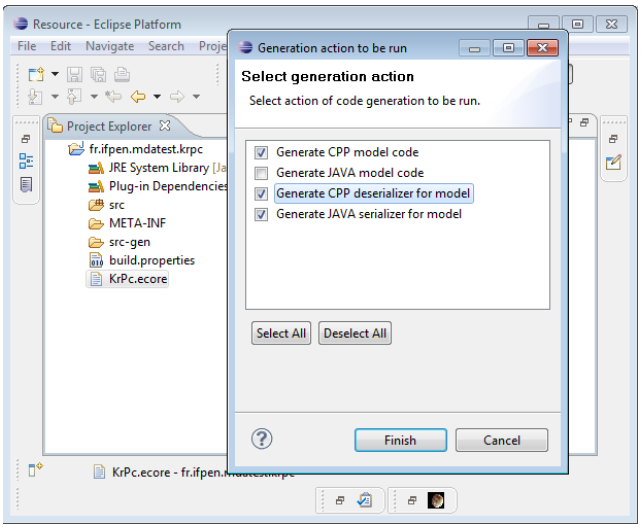

Figure 6. Code generation action

5.3 Test on an actual case

The elements developed are validated using an existing data model already processed with the old architecture in order to compare the results. The data model is based on the persistent model of KrPc (rock type, relative permeability and capillary pressure), some of which is reproduced in **Figure 5**.

The generation action for the different source codes is available on selection of the Ecore model (**Figure 6**). The output code is then compiled and integrated in OpenDataServer for Java and in the client for C++.

5.4 Execution

Once the elements have been generated and integrated in the different entities, the next step is the execution sequence (see **Figure 7**).

1. The C++ client sends a request for data to OpenDataServer. This operation is based on a protocol which is also generated with Acceleo templates;
2. The Java server ODS processes the request, loading the data via the persistent model;
3. The server then calls data adaptors if necessary;
4. The Java instances are serialized, then transferred to the client;
5. The client calls the deserialization tool and retrieves the C++ instances from the transferred data;
6. The client can then use the data.

As mentioned in paragraph 5.2 old and new architectures coexist in a validation version and therefore simulations can be run with both solutions. Results are compared to validate the new architecture. The simulated physical phenomena, oil and gas production at wells in our case, should be strictly identical and it is in this way that we can validate our architecture.

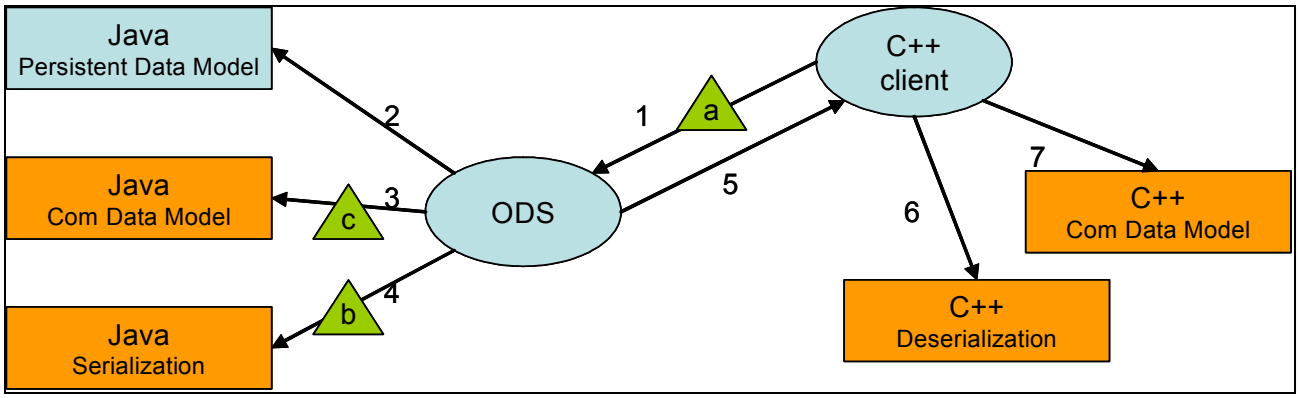

Figure 7. Runtime sequence

The meanings of the small triangles in the **Figure 7** are:

a. Requires a communication protocol between client and server;

b. Uses an Eclipse extension point to register communication data model and serialization tool;

c. Defines mapping between two data models and applies M2M transformation if communication data model and persistent data model differ.

6. Some Metrics

In order to better illustrate the impact of the introduction of MDE in the development process we can quantify the part of technical code that can be generated.

These two tables show the reduction of technical code to maintain when introducing the MDE based solution. The first table counts technical code written once and used as framework in each solution.

Table 1: Generic code metrics

What	Number of lines of code in initial solution	Replace by in IFPEN Workbench
XML for communication	10k	
Java for communication	30k	Acceleo Java template (<1k)
C++ for communication	30k	Acceleo C++ template (<1k)

The second table counts technical code written for each business model. Considering that we have at least 20 different business models to communicate between OpenFlow and a simulator and about 5 to 10 different simulators to manage, more or less 1 million lines of manually written and maintained code can be replaced by Ecore models and generated Java and C++.

Table 2: Per model metrics

What	Number of lines of code in initial solution for KrPc model	Replaced by in IFPEN Workbench
XML for communication	2k	An Ecore model (30 classes)
C++ for model	5k	

The key point is that Acceleo generator is written only once. Then only different data models evolve during the application lifecycle. The maintenance action is then reduced to the maintenance of the workbench (the Acceleo generator) and all generated technical lines of code are not modified any more.

7. Lesson Learned

The objective of this study was to propose a solution to simplify the global mechanism of data transfer from a Java application managing information structured with a data model to C++ or Fortran scientific simulators using this information but through different structures.

The analysis of our existing architecture allowed us in a first time to question the technical choices by returning in particular on the initial constraints, dating almost 10 years, which are not necessarily relevant today. In particular the notion of open services described in a dictionary, if it allows to connect simulators without intervention in the data server, turns out too heavy to maintain in our context. Actually we have the full control of the simulators to be connected and can intervene to manage evolutions of the services.

Besides, the dispersion of the mapping of the data models, partly in the data server, partly on client side turns out to be a source of permanent problems in a model in evolution.

All this analysis and the study of the solutions based on MDE and code generation brought to us to propose a more rigorous solution leaning on a unique reference: the data model. The first step was to build a persistent Java data

model that use database. But in the future, the data model will evolve to be based on communication rather than persistence. The existence of the data model allows us to build an efficient and easy-to-use development environment based on the powerful EMF tools.

8. Conclusion

This study demonstrated the advantages of introducing MDE for implementation of a communication system between a graphical environment, a database and distributed numerical simulators.

The main advantage is, of course, to reduce the amount of technical code manually managed and therefore drastically decrease the development time required for the integration of new simulator and the update effort necessary for each evolution of the data model to be transferred.

Another significant advantage is the ability to regulate and harmonize operating modes. In particular, working on a pure communication model ensures that business processing always takes place in the same location and that the role of OpenDataServer is exclusively to send data to the calculators. Within the framework of manual management, it is tempting to perform business processing in this Java section before providing data to the simulators. But scattering business processing throughout OpenDataServer will create upgrade problems. For example, replacing one simulator with another will become more difficult.

The IFPEN workbench therefore also allows developers to work in a better defined framework, and to switch more easily from one simulator to another while concentrating on the connection aspects.

This solution also has limitations, particularly in terms of parallelism and high performance. In the context of a parallel simulator, it may be necessary to supply it with data corresponding to a partitioning. Within this framework the use of MDE can be very difficult. The solution implemented works well for a transfer of a cluster of objects with a parent node and a moderate total volume. In some cases the volume of data to be exchanged has to be finely tuned, and this can only be done manually.

The prospects for industrial implementation are however very interesting for a high number of information elements and simulators that require distribution but not necessarily parallelism. The aim is therefore to put the solution into practice for new data and future simulators every time the impact on performance is negligible.

References

[1] Eclipse Foundation: What is Eclipse and the Eclipse Foundation?, http://www.eclipse.org/org/ (2012)

[2] Schmidt, D.C.: Model-Driven Engineering, IEEE Computer (2006)

[3] Object Management Group: MDA Spécifications, http://www.omg.org/mda/specs.htm

[4] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd Edition, Addison-Wesley Professional, (2008)

[5] Baker, P., Loh, S., Weil, F.: Model-driven engineering in a large industrial context - Motorola case study. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, pp. 476–491. Springer, Heidelberg (2005)

[6] Staron, M.: Adopting model driven software development in industry - a case study at two companies. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 57–72. Springer, Heidelberg (2006)

[7] Fleurey, F. et al.: Model-Driven Engineering for Software Migration in a Large Industrial Context, MoDELS 2007, LNCS 4735, pp. 482–497. Springer, Heidelberg (2007)

[8] Eclipse Foundation: Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/ (2012)

[9] TOPCASED: The Open-Source Toolkit for Critical System, http://www.topcased.org/ (2011)

[10] Eclipse Foundation: Acceleo, http://www.eclipse.org/acceleo/ (2012)

[11] Object Management Group: Object Constraint Language Spécifications, http://www.omg.org/spec/OCL/ (2012)