MoDisco, a Model-Driven Platform to Support Real Legacy Modernization Use Cases

Gabriel Barbier, Hugo Bruneliere, Frédéric Jouault, Yves Lennon, Frédéric Madiot

MoDisco, a Model Driven Platform to Support Real Legacy Modernization Use Cases

Gabriel Barbier[1], Hugo Bruneliere[3], Frédéric Jouault[3], Yves Lennon[2], Frédéric Madiot[1]

[1] MIA-SOFTWARE
4, rue du chateau de l'Eraudière BP 72 438
F-44324 Nantes Cedex 3 - France
gbarbier@mia-software.com
fmadiot@mia-software.com

[2] SODIFRANCE
4, rue du chateau de l'Eraudière BP 72 438
F-44324 Nantes Cedex 3 - France
ylennon@sodifrance.fr

[3] ATLANMOD Team (INRIA & EMN)
4 rue Alfred Kastler
F-44307 Nantes Cedex 3 - France
hugo.bruneliere@inria.fr
frederic.jouault@inria.fr

Abstract. Presentation of a model-driven migration chain used by Sodifrance on its projects and of Eclipse/Modisco platform a new model-driven framework to develop legacy modernization tools.

Keywords: Knowledge Discovery, KDM, MDE, Eclipse, EMF

1. Introduction

Legacy systems embrace a large number of technologies: making the development of tools to cope with legacy systems evolution a tedious and time consuming task. To deal with the myriad of technological combinations found in modernization roadmaps, model-driven approaches and tools offer the requisite abstraction level to build up mature and flexible modernization solutions. This case study presents a two-phase discussion of model driven modernization.

In Section 2 of this paper we present the initial collaboration between AtlanMod and Sodifrance which led to a model-driven legacy modernization approach. Then, in Section 3, we describe the current process and tools used by Sodifrance on its modernization projects. An illustration drawn from a real migration project, from VB6 to JEE, carried out for Amadeus Hospitality, is included in that section.

Finally, in Section 4, we present [MODISCO], a new Eclipse initiative capitalizing on this experience to deliver a model-driven platform for the development of legacy modernization tools. An illustration of a knowledge discovery tool developed for WesternGeco to understand an existing application containing both Java and C# code is included in that section.

2. Genesis of a Model-Driven Legacy Modernization Approach

In 1993, Sodifrance and Nantes University established a collaboration to work on semantic knowledge extraction from legacy systems. The goal of this collaboration was to transfer innovative work and ideas from Jean Bezivin's research team to a company able to industrialize the prototypes and solve industrial modernization problems encountered by their customers.

The research work of Jean Bezivin's team was to enable the representation of existing applications. When dealing with all the syntactical categories found in an existing program, they came to the conclusion that the precise, complete and evolutionary expression of these categories, and of the various relations between them, needed a robust ontological framework. In 1993, they developed the sNets [SNETS] technology, a first implementation of typed, reflective and modular semantic networks. It was a minimalist representation for dealing with all kinds of models, metamodels and meta-metamodels.

The sNets technology was immediately used by Sodifrance to develop a semantic discovery tool, named Semantor, to analyse any COBOL program and provide a fine-grained level of information about its internal structure and data. The representation of this information, with typed nodes and links, could then be processed to discover control and data flows, represented in the same way. To help the user understand the programs being analyzed,, a view of the program, synchronized with the source code, was provided to facilitate the understanding of the relationship between an element of the semantic network and its initial textual representation. In addition to this, graphical representations were automatically produced to facilitate navigation through this semantic network.

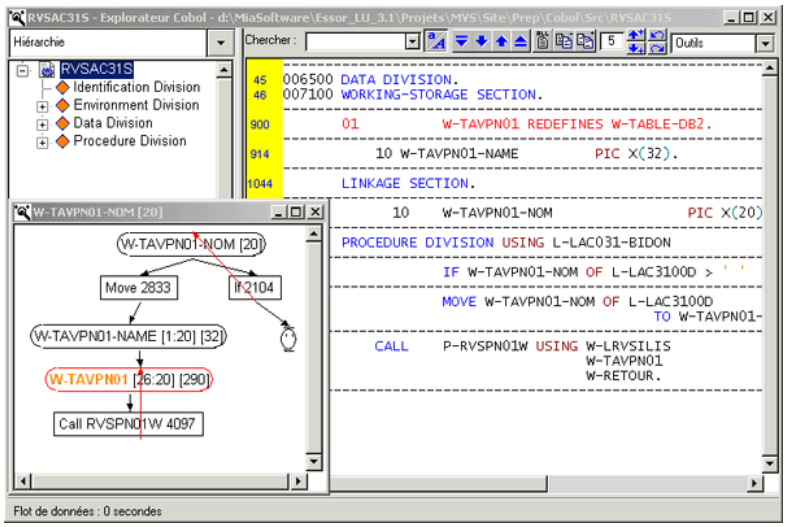


Figure 1 : COBOL source code and control flow displayed in Semantor

When faced with the need for evolution of a real program whose documentation is obsolete or missing, Semantor helps with the analysis of the impacts engendered by a modification, especially on critical points or variables of the program, allowing the user to:

- Take into account the secondary impacts via parent zones or subsidiary zones
- Take into account the impacts caused by redefined structures
- Select impact types: storage, test, computation, etc.
- Choose the number of propagation levels
- Select a propagation direction : forward, backward or both
- Activate the interpretation of subprograms calls
- Indicate specific zones to be excluded or "mark" them as propagation break points

This tool, now included in the Mia-Insight [MIA-INSIGHT] suite, has been used in numerous modernization projects, particularly for the two following project types: Year 2000 and Euro conversion. Semantor has also been used to migrate COBOL programs from Bull's GCOS8 operating system to IBM's MVS in France. During this project, which ran 100.000 man-days, Semantor helped expose dependencies among the components, facilitating the step-by-step migration of the whole system.

In parallel, at the end of 1998, based on the experience gained in the rebuilding of another insurance company's contract management system, where modelling and tailored code-generators were successfully used, Sodifrance started developing its own model transformation technology. From this work, in association with Jean Bezivin who brought his knowledge about early work at OMG on MOF, was born two tools. The first was Scriptor-Generation, a template-based code generator for models, and the second was Scriptor-Transformation, a rule-based model-to-model transformation tool.

These tools, now grouped in the Mia-Studio suite, have been used by major companies to build tailored model-driven tools for producing their application systems. The type of applications produced with Mia-Studio vary from flight management systems written in C and ADA for avionics companies, to information systems based on JEE or .Net frameworks for companies in the banking, insurance, transportation and administration sectors.

In 2002, Sodifrance was asked by one of its banking customers to migrate its client/server system written with the obsolete Cool:Gen technology to the JEE framework used to develop new applications. As this client had already adopted a Model Driven Architecture (MDA) process to produce these new applications, Sodifrance had the idea to link its legacy analysis tools with its model-driven transformation tools to build a migration chain capable of automating the migration process.

On the basis of this successful experience, Sodifrance has now successfully carried out more than ten similar migration projects using this approach, capitalizing on a dedicated process and modular tools capable of confronting a wide range of source and target technologies.

3. Process and Tools for Migrating Legacy Applications

3.1 The challenge

Many of the migrations performed by Sodifrance consist of transforming existing Client/Server applications to JEE or .Net environment. The technology of the source application is frequently based on a proprietary language which is no longer supported by its author. Examples of these languages, mainly born in the nineties, are VB6, OracleForms, Forte, Cool:Gen, NSDK, Natstar or PowerBuilder. Most of these languages are procedural as opposed to object-oriented.

The applications written in these languages are mostly based on a two-tiered architecture. The first tier (the server) is frequently a database containing stored procedures, or a mainframe application. The second tier (the client) is a collection of fat windows providing rich graphical components and embedding business processing and direct access to the server tier.

The challenge is to transform, simultaneously, the architecture of the application (from Client/Server to Multi-Tier), the paradigm used to structure the code (from procedural to object-oriented), and the implementation syntax(es). In addition, the target applications have to rely on patterns and frameworks which are generally different from one customer to another.

3.2 The approach

To solve the complexity of such transformations and facilitate reusability between similar projects, Sodifrance has progressively built a semi-automated migration chain based on model-driven engineering. The approach follows the Horseshoe Model as formalized by the OMG/ADM Task Force [OMG, 2007]. This model defines three synchronized steps which have to be combined to constitute a transformational path:

1. Knowledge discovery of the existing solution. This occurs at many levels of abstraction across varying degrees of scope as appropriate to the projects involved.
2. Target architecture definition. In order to create a transformation approach, analysts must create a target solution that serves as a framework into which existing solutions are mapped or transformed.
3. Transformative steps that move the as-is state to the to-be state. The approach ranges from the physical (e.g. a language migration) to the more abstract (e.g. business rule mapping to a rules-based environment).

Depending on business and IT requirements, the ADM Horseshoe Model includes a wide range of possible journeys from technical to business architecture-driven modernizations.

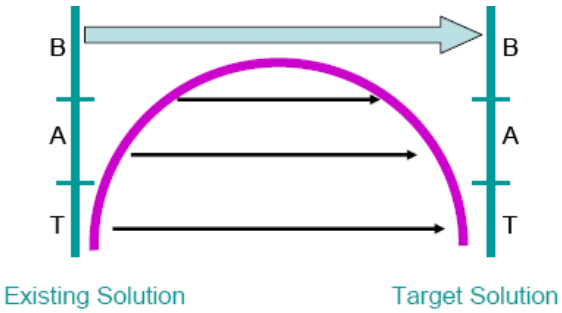


Figure 2 : ADM Horseshoe Model

3.3 The migration chain

3.3.1 Overview

Sodifrance's current migration chain is based on models created by discoverers dedicated to each source technology. These models are transformed to produce the code of the target application. The transformations are performed by model-to-model and model-to-text engines driven by rules adapted to the particular context of each migration.

The migration chain is composed of three main steps:

1. Extraction of a comprehensive model (the initial model) of the existing application from its assets (source code, configuration files, development repositories, etc).
2. Transformation of the model of the existing application into a comprehensive model of the target application (the target model).
3. Generation of the source code of the target application from the model of the target application.

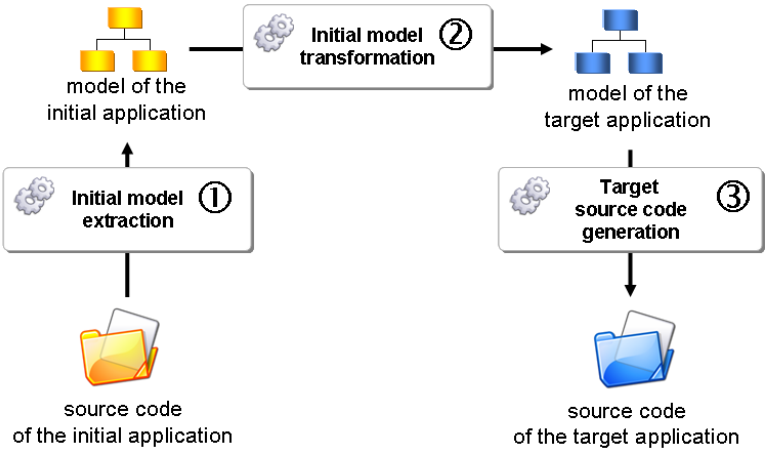


Figure 3 : Overview of project migration approach

In the case of migrations from Client/Server applications to JEE or .Net technologies, the initial and target models conform to two metamodels dedicated to this type of migration. To maximize reusability of the transformation between the two models, the Client/Server and Multi-Tier metamodels have been designed to be language-neutral and independent of any specific framework.

To avoid losing information during the transformation and to guarantee the generation of a complete application, the metamodels define both the concepts needed to fully describe the algorithms and the graphical user interfaces. In addition, the Client/Server metamodel defines concepts such as fat window, where the Multi-Tier metamodel defines concepts such as class, web page, MVC pattern, etc.

3.3.2 Initial model extraction

To facilitate the creation of the initial model, this step has been divided into three sub-steps:

1.1. A discoverer extracts information out of artefacts constituting the existing application so as to create an initial model conforming to the concepts of the implementation language.

1.2. A transformation translates this language-specific model into a language-independent model (the Client/Server model) to eliminate those concepts specific to the initial source language.

1.3. Implicit knowledge is deduced to enrich this model with information such as GUI navigation logic, control and data flows, business versus technical data and other useful information about the application environment.

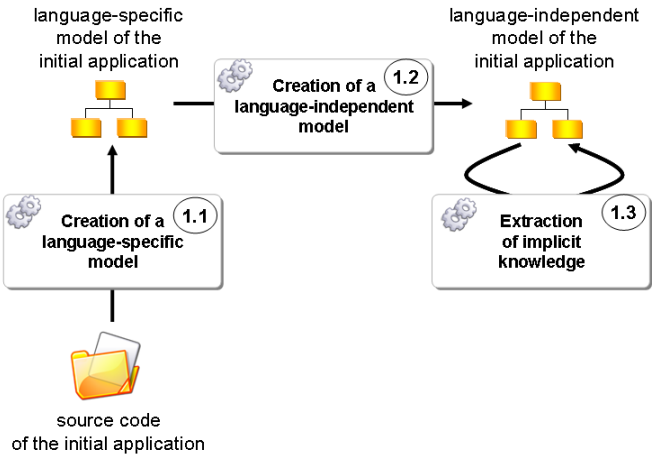


Figure 4 : Extraction of initial model

3.3.3 Initial model transformation

The second main step consists in transforming the Client/Server concepts into Multi-Tier concepts. It is during this step that the application is redesigned to identify business objects, data access components and graphical user interface navigation logic.

Several strategies involving cumulative transformations have been developed by Sodifrance to solve the principal cases encountered. When the transformation is too complex, or needs a human expertise, tags are placed on the corresponding model elements in order to resolve them manually at the end of the transformation.

The result is a comprehensive model describing the complete target application with its presentation details, its business logic and its access to databases and hosted services.

3.3.4 Target source code generation

The third step consists of generating the application source code from the target application model respecting the customer's technical choices (language, frameworks, design patterns and coding rules). Apart from those cases where manual modifications are needed, because the target model is comprehensive, the regeneration is complete.

One of the main challenges in this generation is to manage the differences in the technical services provided by the source and target platform. There are two cases to be considered:

1 A technical component has no equivalent on the target platform, or existing components are too different, making the transformation too complex. The solution is to develop an equivalent component with the same level of service. An example of this would be a printing component which is often very specific to the initial environment.

2 The best practices of the target platform require a component to be used which has no equivalent on the initial platform (or it exists, but has not been used). For example, replacing direct accesses

to a database by Object/Relational mapping such as Hibernate. This can imply a very complex transformation so as to fully respect the target component. Depending on the degree of this complexity, the number of occurrences and the level of conformity to the best practices required by the customer, this transformation can be partially automated.

3.3.5 MDA migration

In most cases, the customers want to take advantage of the migration to industrialize the maintenance and future evolutions of the migrated application. Where they have already put in place an MDA process with [UML] modelling rules (or a DSL) and code generators tailored to their technical choices, the migration chain can be adapted and enhanced to integrate the production of UML models compliant with these code generators.

These UML models are not complete representations of the application, but they contain the minimum information needed for the code generators to work. For example, most UML models contain neither specific business rules algorithms, nor the specific graphical user interface design (position, size and colour of the widgets). It is for this reason that most generators exploiting UML models only generate file skeletons which have to be completed manually by the developers. To allow iterative work on models and files, these generators produce tags which identify portions where the developer can add his/her code. These fragments of code are preserved when the code is regenerated.

Rather than generating all of the code from the target model, this model is used to produce a UML model compliant with the modelling rules expected by the customer's code generator. All the code which cannot be produced by the UML generator (non-generated files and code fragments to insert into the file skeletons) is produced by a second generator which uses the comprehensive model. This code corresponds to the code which has to be produced manually in the customer's MDA process.

The last step consists in automatically integrating the code fragments into the file skeletons.

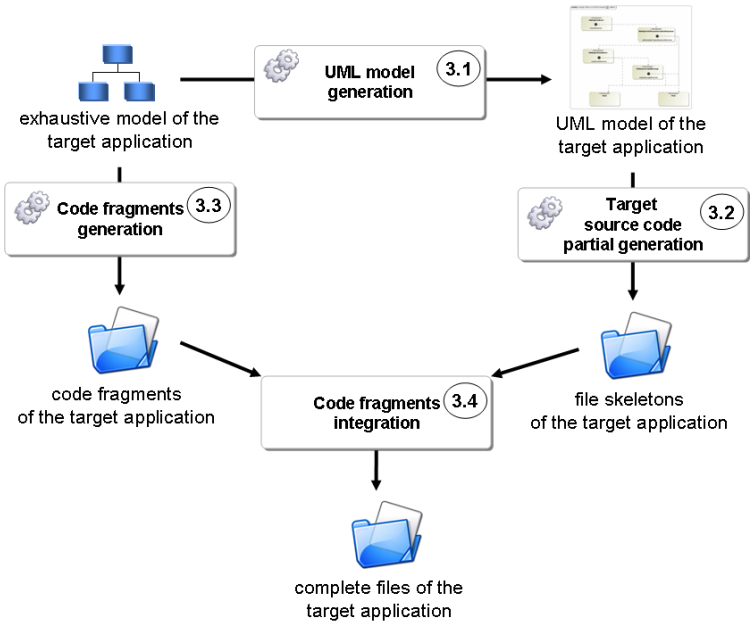


Figure 5 : Generation of target application in an MDA migration

Once migrated, the application can evolve by modifying the model (for functional or design evolutions) or the code generator (for technical evolutions) and regenerating the code.

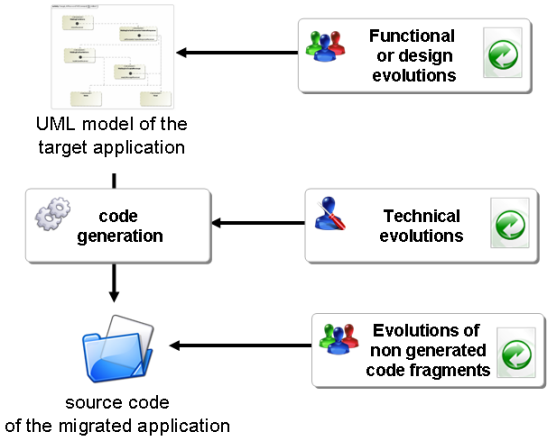


Figure 6 : Maintenance of target application after an MDA migration

The benefit of this approach is that, if the customer has already adopted an MDA process to develop new applications, the same process can be used to maintain the migrated applications.

3.4 The tools

For a given project, Sodifrance's migration chain is automated with three tools: a model discoverer, a model-to-model transformation engine and a model-to-code generation engine.

3.4.1 The model discoverer

A model discoverer is a tool which creates a model out of artefacts of the existing system. Most of these artefacts are files containing source code. These files are analysed by a parser developed from the BNF grammar of the language used to develop the program. Sometimes models can be created by querying a database from which the table structures are extracted. In other cases, the model is built by using APIs to connect to a development environment from which the project structure is extracted.

In each case the result is a model conforming to a metamodel containing all the concepts needed to describe the existing application. In Sodifrance's migration chain, this metamodel is defined with MOF. Thus, the models provided by the discoverers can be serialized with XMI and exported for the use by the transformation engines.

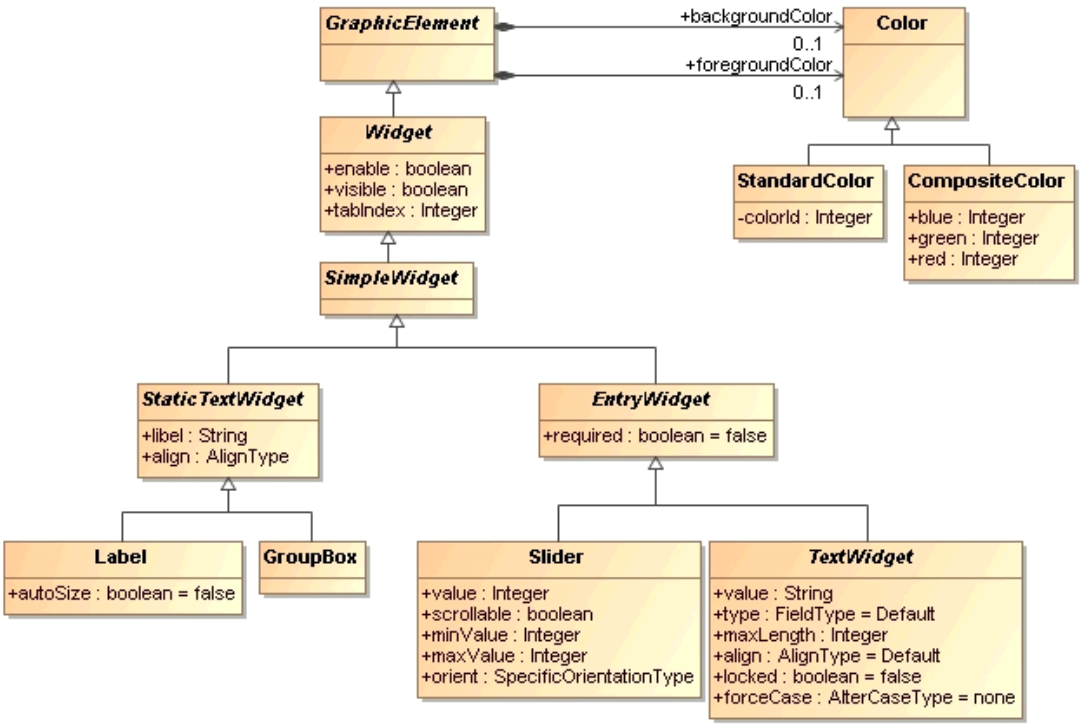


Figure 7 : Subset of Sodifrance's GUI metamodel

3.4.2 The model-to-model transformation engine

To perform the required model transformation from the language-specific model to the target application and the UML models, Sodifrance uses Mia-Transformation, the model-to-model technology provided by Mia-Software in its Mia-Studio tool suite.

Mia-Transformation is a rule-based transformation engine, and its dedicated editor, for defining and maintaining model(s)-to-model(s) transformation rules. Each rule is composed of a query, an action, a context and sub-rules sharing the rule's context. The query and the action can be written using an inference syntax (similar to Prolog) or directly in Java.

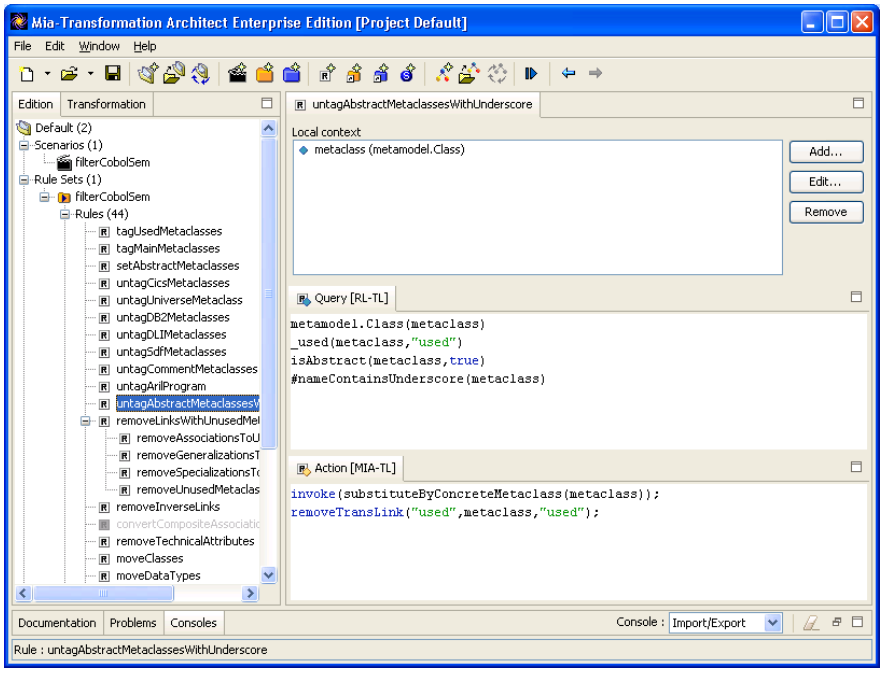


Figure 8 : Edition of transformation rules with Mia-Transformation

In the utilized migration chain, all the transformations are executed by the same tool with its own rule-set organized in several reusable packages. The decomposition of the transformational path into several steps based on intermediate, language-neutral metamodels facilitates the reuse of transformation packages between migration chains dedicated to different source and/or target technologies.

3.4.3 The model-to-code generation engine

The generation of target application artefacts was performed with Mia-Generation, the model-to-text technology provided by Mia-Software in its Mia-Studio tool suite.

Mia-Generation is a template-based generation engine, and its dedicated editor, for defining and maintaining model-to-text transformation rules for any metamodel defined with MOF. Each template can define tags to separate generated code from code added by the developer. These fragments of code are preserved when the file is regenerated. To write complex generation rules, Mia-Generation provides Java APIs to navigate through the model.

To be able to start code generation directly from Eclipse, VisualStudio or MagicDraw, Mia-Software provides plug-ins for each of these environments.

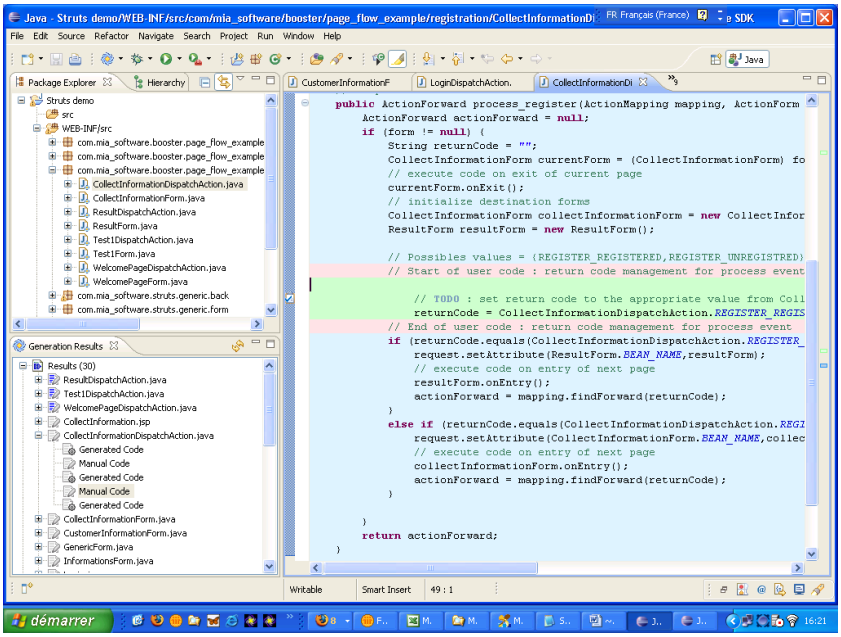


Figure 9 : Mia-Generation plugin for Eclipse

A set of reusable templates based on the Multi-Tier metamodel has been developed by Sodifrance to regenerate Java or C# algorithms from a model statement or web pages (JSP or ASP.Net) from the model of the pages. Most of the other templates are written according to the customer's technical choices.

3.5 The migration process

Each migration project follows a strict process organized in two main phases. The first one is the preparation, during which the migration strategy is defined and the tools are adapted to the specific context of the project. The second one is the industrial phase, during which the application is transformed and tested. This main phase is divided into several phases, one for each migration batch, executed in parallel.

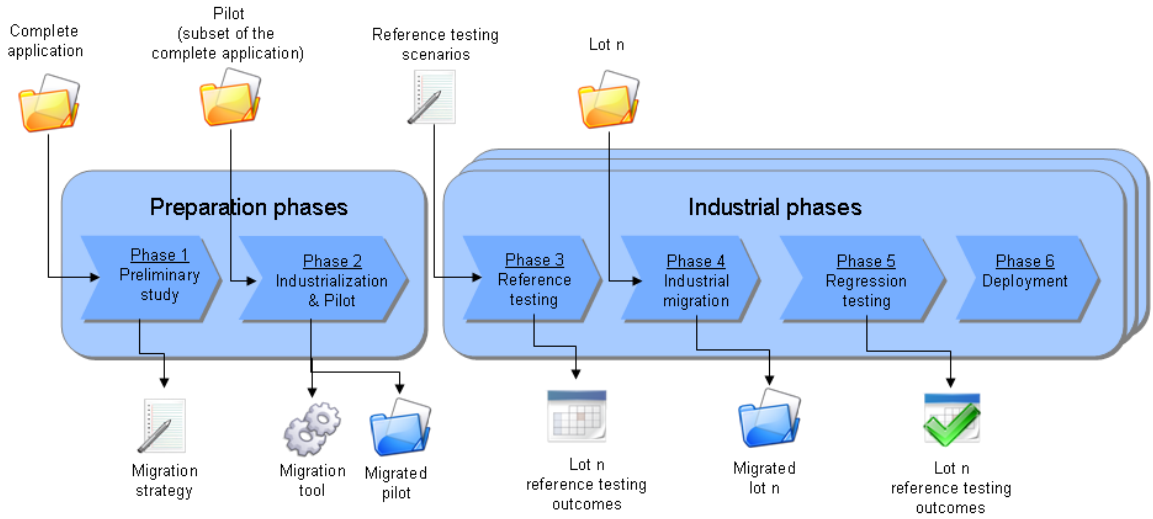


Figure 10 : Sodifrance's migration process

3.5.1 Preliminary study

This phase consists of analysing the entire source application to be migrated, in order to define the migration strategy. During this phase, the existing application's architecture is identified, the design patterns and the technical components used to implement the code are listed and the dependencies between the main components are identified.

Based on the knowledge of the source application acquired, the target architecture is then defined in accordance with the customer's technical requirements. In the majority of cases, the customer has already defined an architecture and a framework for developing new applications in JEE or .Net environments. In such a case, the supplied framework is enhanced to support any new features needed by the application being migrated. The main extensions concern graphical user components not available in the new framework and needed to reproduce rich functionalities of the existing application.

Finally, a migration strategy is defined to establish the way each concept of the initial application will be transformed. This strategy usually comes with technical prototypes validating the technical choices.

3.5.2 Industrialization and Pilot Migration

This phase consists of adapting the tools which make up Sodifrance's migration chain:

- Discoverers
- Intermediate metamodels
- Model-to-model transformation rules
- Code generation templates
- Target framework

As the migration chain is modular and based on language-neutral, intermediate metamodels, the effort mainly concerns the parsers (which have to be developed from scratch when it is the first migration of a given source language) and the target framework (each customer has his own technical requirements).

Each time that a new case is discovered which is not handled by the existing migration chain, a decision is taken whether to automate the transformation of this case or not. The two criteria used are the complexity of the case and the number of its occurrences. Whenever automating the transformation has no evident benefit to the project, the case is not integrated in the migration chain and instructions are given to the migration project team indicating how to manually modify the corresponding code fragment.

The new migration chain is then validated on a pilot application which is a subset of the real application to be migrated. This phase is iterative, until the pilot is correctly transformed and all risks have been identified. Finally, once the pilot has been validated by the customer, a break-down of the whole migration into migration batches is proposed in order to parallelize the migration process and facilitate the validation of each migration batch.

The following four phases, which included reference testing, industrial migration, regression testing, and deployment, were executed for each migration batch.

3.5.3 Reference testing

The reference tests are the result of test scenarios performed on the application before its transformation. The reference tests are later compared to the result of the same scenarios performed on the migrated application.

3.5.4 Industrial migration

This phase consists of transforming the source code and configuration files of a migration batch.

It begins by transforming manually the source application to handle any specific cases not covered automatically by the migration chain. Then the chain transforms the code. Finally, manual adaptations are made to the result to handle any other cases which have not been automated. For example, adapting the layout of the web pages obtained from windows screens.

3.5.5 Regression testing

This phase consists of executing the reference tests on the migrated application and comparing the results with those executed on the application before its transformation. The aim is to verify that the transformation has not caused unintended effects and that the new system still behaves like the initial one.

3.5.6 Deployment

After the results of the reference tests have been validated by the customer, the application is deployed in the new environment. This phase is generally performed by the customer.

3.6 Amadeus Hospitality: VB6 to JEE Migration

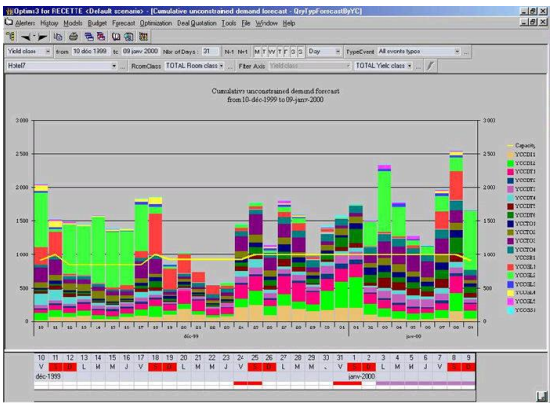
In 2008, Amadeus Hospitality, a subsidiary of Amadeus with 8,300 employees, the leader in IT solutions provided for the tourism and travel industry, asked Sodifrance to migrate one of their applications for hotel management from VB6 to Java Enterprise Edition (JEE).

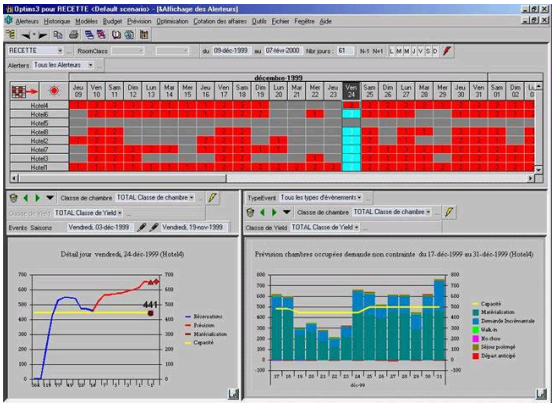
3.6.1 The initial application

The Amadeus Hospitality revenue management system (RMS) is a yield management application providing a fuller understanding of hotel customers' behaviour to help hoteliers in optimizing the profitability of their assets. It is deployed in more than 1,500 properties from 54 hotel chains. In 2000, a specific branch of RMS had been developed for Accor, one of their customers, European leader in hotels and tourism.

The initial RMS version was developed in VB6, and was performing queries on an Oracle database. It was composed of 300 screens, 200 of them displaying charts (pie charts, bar charts or line graphs). The VB6 code was composed of 306,000 source lines of code, VB6 code in 216 classes and 261 modules.

The Accor branch was composed of 100 additional screens - 60 of them displaying charts - and an Informix database.





Figures 11 & 12: Screenshots of initial RMS application

Amadeus Hospitality wished to migrate RMS to a new technical environment for two main reasons. The first reason was the end of support for VB6 by Microsoft, announced for 2008. The second one was the possibility of opening up its information system and facilitating the integration of its partners. With this migration, it was also an opportunity for Amadeus Hospitality to merge the specific Accor branch into the main RMS branch.

3.6.2 The transformation

As JEE technology had been recently chosen at a corporate level, Amadeus Hospitality decided to select this technology for a new lightweight version of RMS based on a classic three-tier format organised in the following manner:

- Presentation tier: screen-handling and user interaction management
- Business Logic tier: business logic and application management
- Data tier: data persistence and access to external components

The new architecture respects the following principles:

- Application of the Model-View-Controller pattern
- Externalisation of accesses to data (database or external components) in the data access objects (DAO pattern)

The new application has been implemented with four Java frameworks: JFreeChart (charts generation), Webwork (presentation layer), Spring (business and data access) and FOP (printing). The database, based on Oracle, remained unchanged for RMS, while the Accor branch database, based on Informix, merged into the Oracle RMS database.

To transform the existing application to this new architecture and technology, Sodifrance followed the model-driven approach, with the tools and the process described in the previous section (apart from the production of a UML model, as Amadeus had not chosen maintain the new application using an MDA process).

The transformation of the VB6 classes to Java classes has been relatively straightforward for the structure and the statements (see the sample VB6 translation below).

```
Dim oDim As cBaseDim
Dim oDefDim As cAnalysisAxis
   For Each oDefDim In AnalysisAxes
      If oDefDim.AxisType = AnaAxisTypStdAxis Or oDefDim.AxisType = AnaAxisTypSubInventory Then
         Set oDim = New cBaseDim
         Set oDim.AnaAxis = oDefDim
         mcolDims.Add oDim, CStr(oDefDim.Id)
         Set oDim = Nothing
      End If

   Next
   Set oDim = New cBaseDim
   Set oDefDim = New cAnalysisAxis
   oDefDim.AxisType = AnaAxisTypStdAxis
   oDefDim.Id = AnaAxisRoomTypeBar
   Set oDim.AnaAxis = oDefDim
   mcolDims.Add oDim, CStr(oDefDim.Id)
```

Figure 13 : VB6 sample

```
BaseDim oDim;
AnalysisAxis oDefDim;
for(int i=0;i<GlobalContext.getGestionCondVals_analysisAxes().size();i++)
{
    oDefDim = GlobalContext.getGestionCondVals_analysisAxes().get(i);

    if (oDefDim.getAxisType() == AnaAxisTypeEnum.ANA_AXIS_TYP_STD_AXIS ||
            oDefDim.getAxisType() == AnaAxisTypeEnum.ANA_AXIS_TYP_SUB_INVENTORY) {

        oDim = new BaseDim();
        oDim.anaAxis = oDefDim;
        mcolDims.add(oDim, StringUtil.parseString(oDefDim.getId()));
        oDim = null;
    }
}
oDim = new BaseDim();
oDefDim = new AnalysisAxis();
oDefDim.setAxisType(AnaAxisTypeEnum.ANA_AXIS_TYP_STD_AXIS);
oDefDim.setId(MConstants.ANA_AXIS_ROOM_TYPE_BAR);
oDim.anaAxis = oDefDim;
mcolDims.add(oDim, StringUtil.parseString(oDefDim.getId()));
```

Figure 14 : Translation of the VB6 sample into Java

Nevertheless, the transformation required the adaptation of the existing Sodifrance rules to take into account the specific Amadeus constraints. Below are listed some of the points for which adaptations have been made:

- **User interface**:
  The translation of the graphical user interfaces required a mapping between VB6 components and the equivalent Java components (including the mapping of corresponding APIs). Most of the target Java components have been selected from WebWork, the framework chosen by Amadeus.

Library	Component	Number of utilisations	Technical solution
VB 6	CheckBox	117	webwork checkbox
VB 6	ComboBox	51	webwork combobox
VB 6	CommandButton	250	webwork Input
VB 6	Frame	162	webwork div
VB 6	Image	9	webwork image
VB 6	Label	399	webwork label
VB 6	Line	21	Html hr
VB 6	ListBox	21	webwork select
VB 6	Menu	2	architecture complements
VB 6	OptionButton	41	webwork radio button
VB 6	PictureBox	73	Html image
VB 6	Shape	39	case by case
VB 6	TextBox	115	webwork textfield
ComCtl32	ImageList	20	webwork div
ComCtl32	ListView	8	Display of images in an html table
ComCtl32	ProgressBar	3	Animated Gif
ComCtl32	Slider	4	Architecture complements
ComCtl32	TabStrip	1	webwork tabbedPane
Threed20.ocx	SSFrame	19	webwork panel
Threed20.ocx	SSCheck	14	webwork checkbox
Threed20.ocx	SSOption	7	webwork radio button
Threed20.ocx	SSPanel	26	webwork div
Timocx.ocx	DBList	70	Table
Timocx.ocx	DBCombo	153	webwork combobox
Timocx.ocx	DBEdit	332	webwork textfield
MSVBCldr.ocx	MSVBCalendar	4	webwork datepicker
InputDate.ocx	InputDate	43	webwork datepicker
Crystal	CrystalReport	4	FOP

Figure 15 : Subset of the mapping from VB6 graphical components to Java

- **Navigation** :
The navigation management has been implemented with WebWorks, which implements the MVC pattern. The logic of the navigation has been specified with XML files.

```
<xwork>
    <include file="webwork-default.xml"/>
    <include file="config-browser.xml"/>

    <package name="default" extends="webwork-default">
              <action name="connect" method="connect" class="connect">
           <result>Login.jsp</result>
                    <result name="SUCCESS">index.jsp</result>
        </action>
    </package>

    <package name="login" extends="webwork-default">
    </package>

    <include file="xwork-optools.xml"/>
    <include file="xwork-optims.xml"/>
</xwork>
```

Figure 16 : Sample of navigation logic with WebWorks

- **Events management**
In Java web applications, there are 3 technologies available for the handling of user events:

- **Form submission**: the whole form is submitted. A new page is returned to the user. There is a latent period between actions.
- **Javascript**: the event is dealt with on the client side.
- **Ajax**: a flow is exchanged with the server in a way that is transparent to the user. This means that the user stays on the current page.

To maintain the user-friendliness of the original client/server application, it was not advisable to use systematically the submission of pages function. Thus, to respect the general ergonomics of the new application, the migration of events has been systematised in the following manner:

- A click on a button, on a menu item: **Form submission**
- Other events (value changes, enter/exit a field etc) when the process connected to the event necessitates access to data on the server: **Ajax**
- Otherwise, if the process is limited to the handling of graphic elements: **Javascript**.

- **Internationalization :**
  Amadeus wished to maintain its existing method of internationalization management, which consists of displaying the different labels defined in the database depending on the user's language and the choice of the company to which he or she belongs. A cache system for labels has been developed in order to improve performance.

- **Multi-database access**
  In the Accor version, a user with the necessary authorization can select the database from which he wishes to run the application. A mechanism for handling data sources has been put in place with the Spring framework to make the use of one database or another transparent to the user.

- **Running SQL queries**
  A generic mechanism to access data had been developed for the original application. This mechanism is based on SQL queries stored in '.sql' extension files. The same mechanism has been implemented with the help of a data access object responsible for:
  - Reading the desired query from the '.sql' file
  - Interpreting it (analysis of any parameterized portions) in order to constitute the final flow
  - Executing the query
  - Recovering the final result and copying it into a transfer object

- **Graph Printing**

The default browser printing function can not be used as the landscape orientation is not possible. To get around this problem, Sodifrance used the PDF format generated from the free library Java iText. For each graph, the user has the option to generate a PDF file in-line.

- **Accessing distant components**

Access to distant components (programs) was managed with a single OCX component written in C++. This component was used via a dedicated function (RemoteExecute) and by events raised depending on the context informing the caller (the VB6 program) with the result of its call. Rather than integrating the OCX component (or a recompiled version accessible with JNI), and given that no equivalent Java component exists with all the required mechanisms, the same component has been rewritten in Java.

- **Data import/export**

Data import and export were managed by a specific component developed by Amadeus to allow the user to upload the database by copying a bundle of data from an Excel or text file. To reproduce this functionality, Sodifrance explored the following two possible solutions:

- Using a html/javascript table
- Using an Office Web Component

The table below shows the advantages/drawbacks of each option:

Solution	Functionalities	Portability
html/javascript	Partial	Yes
Office Web Component	Complete	Partial (needs Windows with Internet Explorer on the client workstation)

The decision was made to use **Office Web Component**

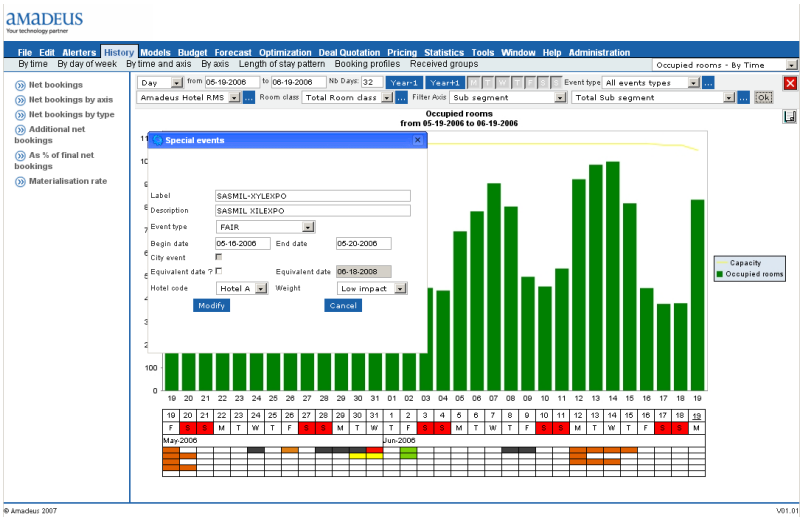
3.6.3 The migration project

The migration project was completed by Sodifrance in 1,600 man-days with ten engineers over a year.

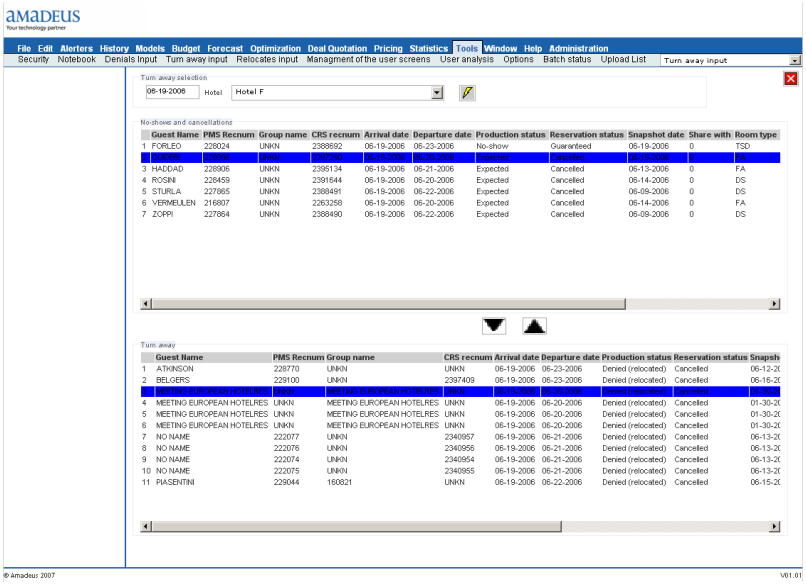
The preparation phase, composed of the preliminary study (study of the existing, validation of the target architecture and specification of the transformation) and the industrialization (adaptation of the transformation chain), was carried out in three months.

The industrial phase was divided into three migration batches: two for the main RMS branch and one for the Accor branch. The reference tests were provided by Amadeus Hospitality as scenarios containing a total of about 3,000 unit tests which have been replayed on the JEE version of applications to validate the transformation. The transformation of all the VB6 code (access to data, business rules and interface) was 80% automated, while the definition of the screens (Forms) was only 50% automated, due to the necessity to redesign them for a web mode.

The new version of RMS is now composed of about 300,000 lines of code in 1,000 Java classes and 310 Jsp (Java Server Pages).



Figures 17: Screenshot of migrated RMS application



Figures 18: Screenshot of migrated RMS application

As required by Amadeus Hospitality, the new application has been tested with 100 simultaneous users and the performance is unchanged.

The deployment of the final application has been carried out by Amadeus Hospitality.

4. MoDisco, a New Eclipse Platform to Support Model Driven Legacy Modernization

In 2007, the European Community funded [MODELPLEX], a new research project whose objective is to deliver open, model-driven solutions for complex systems engineering. As developing complex systems requires integrating or transforming existing systems, Modelplex has planned to deliver solutions for knowledge discovery: how to create models out of existing heterogeneous systems to handle their complexity. Sodifrance and AtlanMod, the new INRIA laboratory created in Nantes by Jean Bezivin, were selected to lead this subject.

AtlanMod, already involved in a lot of Eclipse modelling projects such as ATL (Atlas Transformation Language, the model-to-model transformation language) and AMW (Atlas Model Weaving, a model weaving technology), proposed to launch MoDisco, a new Eclipse project dedicated to reverse-engineering. Sodifrance decided to contribute to MoDisco to bring its experience of successful modernization projects to this new platform. The objective reached by AtlanMod and Sodifrance is to make MoDisco a reference platform for legacy modernization tools based on Eclipse.

4.1 General Presentation

MoDisco (Model Discovery) is an Eclipse GMT (Generative Modelling Technologies) component for model-driven reverse engineering. The objective is to facilitate the development of tools to extract models from legacy systems and use them on modernization use cases.

Because of the widely different nature and technological heterogeneity of legacy systems, there are several different ways to extract models from such systems. MoDisco proposes a generic and extensible metamodel-driven approach to model discovery. A basic framework, including implementations of OMG standards such as KDM or SMM, and a set of guidelines are provided to the Eclipse contributors to bring their own solutions to discover models in a variety of legacy systems.

As an Eclipse component, MoDisco tools can integrate with plug-ins or technologies available in the Eclipse environment, especially those of the Eclipse Modelling Project EMF (Eclipse Modeling Framework), M2M (Model to Model transformations), GMF (Graphical Modelling Framework), TMF (Textual Modelling Framework), etc.

4.2 Discovery Principles

Principles of model discovery are based on a metamodel-driven approach. It means that every step is guided by a metamodel. Thus, the very first step of a model discovery process is always to define the metamodel corresponding to the models to be discovered. This step is common to all types of systems.

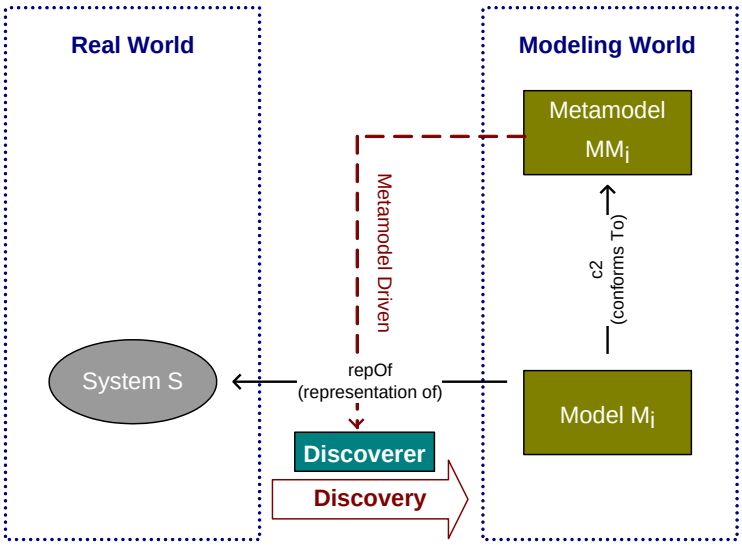


Figure 19: Metamodel driven discovery

Then, the second step is the creation of one or many discoverer tools, referred to as "discoverers" in this document. These tools extract the required information from the system in order to build a model conforming to the previously defined metamodel. In some cases these discoverers can be partially generated from a description of the syntax of the files containing the information. In other cases, the discoverer reuses an existing component able to extract the information and translates this information into a model. The output of a discoverer is a model, in XMI format for instance.

4.3 Architecture

MoDisco aims at providing a platform supporting various legacy modernization use cases for various kinds of existing technologies. To facilitate reuse of components between several use cases, MoDisco has been organized in three layers:

- The **Use Cases** layer containing components providing a solution for a specific modernization use case.
- The **Technology** layer containing components dedicated to one legacy technology but independent from the modernization use case.
- The **Infrastructure** layer containing generic components independent of any legacy technology.

Figure 20 shows each of these layers within the overal MoDisco architecture.

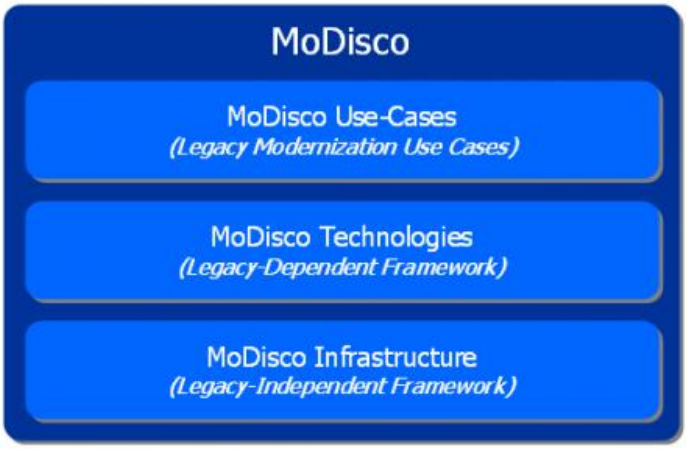


Figure 20: Overall architecture of MoDisco

4.3.1 Use Cases Layer

This layer contains components supporting legacy modernization use cases. The kind of use cases MoDisco could support is theoretically infinite. Nevertheless, the main ones are well identified:

- **Quality Assurance**: verifying whether an existing system meets the required qualities (detection of anti-patterns in existing code and computation of metrics.)
- **Understanding**: extraction of information from an existing system to help understanding one aspect of this system (structure, behaviour, persistence, data-flow, change impact , etc).
- **Refactoring**: improvement of an existing system to integrate better coding norms or design patterns.
- **Migration**: transformation of an existing system to change the framework, the language, or its architecture.

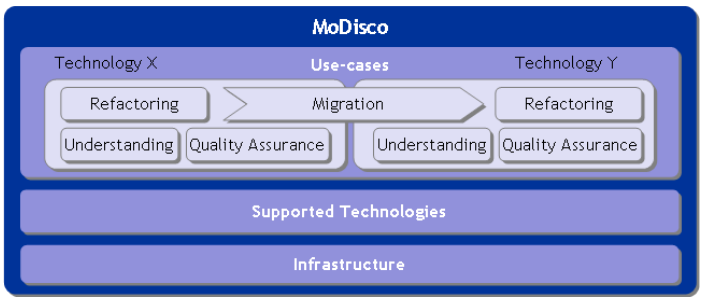


Figure 21: Use cases layer of MoDisco architecture

Other more specific use cases may be supported. For example, the extraction of business rules from programs to populate a business-rules engine, or the modification of an existing system to better integrate with another system, etc.

4.3.2 Technology Layer

The technology layer aims at containing components dedicated to one legacy technology.

These components can be reused between several use case components involving the same legacy technology. For example, a use case computing metrics on Java source code and another providing refactoring for Java applications could reuse the same component.

Use cases generally involve only one legacy technology (the one used to implement the existing system). Nevertheless, some use cases can involve several legacy technologies. It is the case when the existing system is heterogeneous, built with several languages. For example, when a system, implemented with Java, stores data into a relational database using JDBC, the use case may need MoDisco components able to analyse Java and SQL source code.

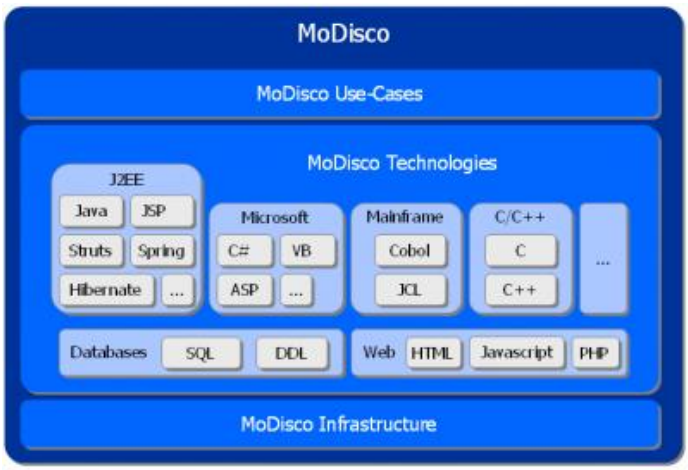


Figure 22: Technologies layer of MoDisco architecture

Each technology component is composed of, at least, a metamodel of the dedicated technology. This metamodel describes the elements required to support modernization use cases for the corresponding technology. Depending on the kind of use case, the metamodel can be complete (for refactoring or migration) or partial (for cartography or some quality analysis scenarii).

Ideally, the metamodel comes with a discoverer, a component which aims at building models conforming to the metamodel from artefacts of an existing system. Artefacts analyzed by discoverers are not necessarily source code files. There exists several other ways for a discoverer to find the information needed to create a model from an existing system:

- Analyzing parameter files
- Analyzing execution logs
- Unzipping an archive to access its contents
- Querying a database
- Using APIs to access a tool with which the system has been designed
- Translating data provided by a reverse-engineering tool into a model
- Transforming models provided by another discoverer

A *MoDisco Technology* component can come with utilities dedicated to its metamodel:

- Browsers to navigate the models more easily
- Viewers to represent the models graphically
- Computation of standard metrics
- Transformation to standard metamodels (ASTM, KDM, UML …)
- Generator to regenerate the initial artefact from its model

4.3.3 Infrastructure layer

The Infrastructure layer aims at providing components independent from use cases and legacy technologies. There will be two kinds of components in this layer:

- **MoDisco Knowledge components**

These components provide metamodels describing legacy systems independently from their technology. Like components of the Technology layer, these components can come with discoverers or utilities.

Examples of MoDisco Knowledge components are the metamodels from OMG/ADM :

- KDM (Knowledge Discovery Metamodel)
- ASTM (Abstract Syntax Tree Metamodel)
- SMM (Structured Metrics Metamodel).

- **MoDisco Technical components**

These components are utilities to build or facilitate the use of all the other components.

Examples of MoDisco Technical components are:

- Abstract discoverers from which concrete discoverers can be derived
- A file-system metamodel describing the organization of files and directories
- A model browser facilitating the visualization of MoDisco models.

4.4 Existing components

MoDisco already contains several components for each layer.

4.4.1 KDM, the OMG Knowledge Discovery Metamodel

MoDisco contains a reference EMF implementation of the Knowledge Discovery Metamodel, the OMG standard which aims at ensuring interoperability and exchange of data between legacy modernization tools provided by different vendors.

This component belongs to the *Infrastructure* layer of MoDisco.

4.4.2 J2SE5 to discover Java applications

The J2SE5 metamodel is a reflection of the Java language, as it has been defined in version 3 of "Java Language Specification" from Sun Microsystems ("JLS3" corresponds to JDK 5). It contains 101 metaclasses which allow describing the complete abstract syntax graph of a Java file to be described: the structure of a class (declarations of variables and methods), the body of each method (blocks, statements and expressions) and the link between elements' usage and their declaration (superclass definition and the corresponding class declaration, variable setting and the declaration of the variable, method invocation and the declaration of the method, etc).

This metamodel comes with two additional components:

- A discoverer which populates a J2SE5 model from a Java project. This discoverer is based on the Eclipse JDT (Java Development Toolkit) component which provides an abstract syntax tree (AST) from a Java project. The role of the MoDisco discoverer is to translate this AST into a model conforming to J2SE5.
- A transformation to translate the J2SE5 model into a KDM model. This transformation supports the KDM Code package and part of the Action package: the structure of the Java classes and the signature of the methods are created in the KDM model and some of the statements contained within the methods (method invocations for example).

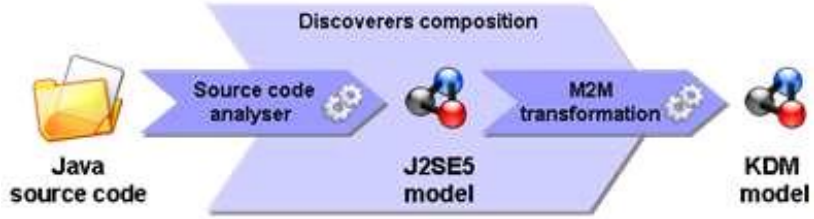


Figure 23: J2SE5 discoverers

Combined with model-to-model and model-to-text transformation tools, the discoverer has already been used on several use cases such as:

- Quality Assurance: control of specific coding rules
- Documentation: generation of a specific UML model and HTML documentation (javadoc) taking into account a proprietary framework used to develop a JEE application
- Model Filter: generation of partial UML models containing only typed dependencies between classes
- Migration from EJB2 to EJB3: modification of the J2SE5 model to integrate the EJB3 concepts and regeneration of the source code
- Migration from Swing to GWT: conversion of Swing APIs to GWT APIs (Google Web Toolkit, a Java framework to develop rich graphical user interface for web applications) in the model generation of GWT files.

The Model Filter use case, developed for the Western-Geco company, is described in section 4.5.

This component belongs to the *Technologies* layer of MoDisco.

4.4.3 CSharp to discover C# applications

The CSharp metamodel is the reflection of the C# language, as it has been defined in version 2.0 of "C# Language Specification" from Microsoft Corporation. It contains 81 metaclasses. Like J2SE5, this metamodel describes up to the contents of the methods.

This metamodel also comes with a transformation to KDM. Work is in progress to contribute an open discoverer able to populate a CSharp model.

Combined with a proprietary discoverer (written in C) and ATL (the model-to-model transformation tool provided by INRIA/AtlanMod) this component has been used on the Western-Geco use case described in section 4.5.

This component belongs to the *Technologies* layer of MoDisco.

4.4.4 ModelBrowser to discover model contents

Each discoverer provides one or several models created by analysing the artefacts of an existing system. These models typically contain a great number of elements, as modernization use cases need a very precise description of the systems they represent. Moreover, contrary to models created manually with a graphical modelling tool, models automatically created out of an existing system are not known beforehand. That is why, it is crucial to provide a tool which helps in exploring this kind of model.

The MoDisco model browser is a feature-rich Ecore model browser. It can be used to browse and edit any Ecore model more easily than with the default Ecore editor.

The surface of the browser is separated into two panes. The left one displays a list of metaclasses, and the right one shows instances of the selected metaclass (that is, model elements). At the top of each pane, a toolbar allows you to quickly change display options relative to that pane.

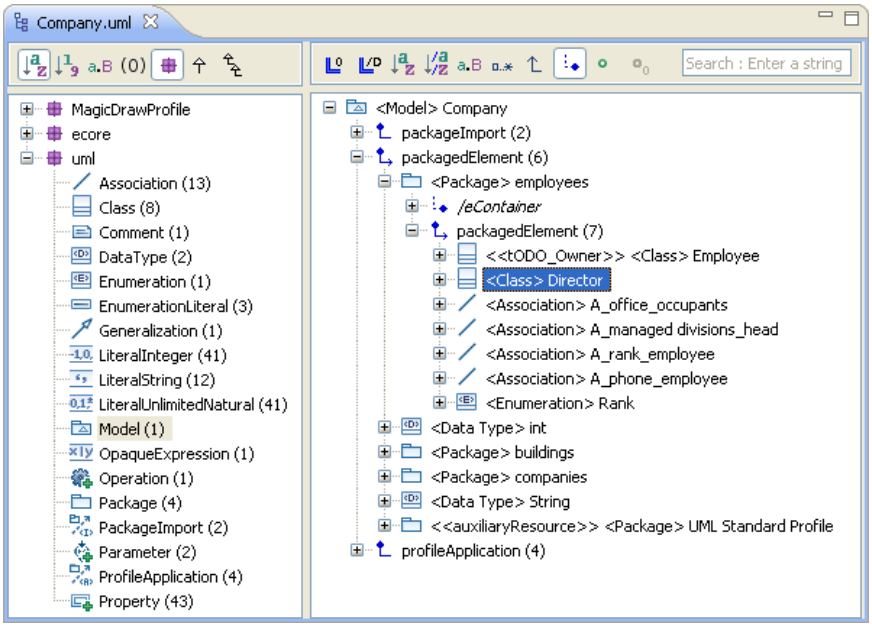


Figure 24: MoDisco Generic Model Browser

Links can be follwed between model elements by expanding the links tree nodes. Links appear for associations, aggregations and the EMF container. The tree representing the model is infinite. Specific icons indicate the type of the links (navigable or not, aggregate or not) and the number of instances attached via this link.

The source of the link is always its parent in the tree, and the targets are its children.

A toolbar allows the display options to be set (show links with no instances, show derived links, show link multiplicities, sort links by name, show a link to the container, show attributes in the tree, show empty attributes, sort instances by name).

The left pane displays all the metaclasses of the opened model, with the number of instances for each metaclass. By right-clicking on a metaclass, its instances are displayed in the right pane. Multi-selection of metaclasses is also supported.

Right-click on a model element and select "Browse" to select this element's metaclass of this element in the left pane and display this model element amongst its siblings of the same type. You can also press <Enter> while an element is selected to trigger this action.

The MoDisco model browser currently provides one extension point for customizing the naming of instances, and another one for specifying icons for model elements. More will come in the future.

This component belongs to the *Infrastructure* layer of MoDisco.

4.5 Analysis of an existing J2EE application, a MoDisco use case on a real application

One of the first industrial use cases using MoDisco has been the understanding of a Large Scale Data Intensive Geological system for WesternGeco, a geophysical services company. As part of [MODELPLEX], a research project funded by the European Community, Mia-Software has developed a tool to help WesternGeco understand this application.

4.5.1 About WesternGeco

WesternGeco works in the area of Oil and Gas Exploration, offering advanced seismic services. The aim of seismic surveying is data acquisition to produce images of geological features and their structure below the

surface of the earth.

The company goal is to provide a full seismic service package to the oil companies, ranging from 3D and 4D (time-lapse) seismic surveys to multi-component and electromagnetic surveys, supplying their clients with accurate measurements of subsurface geology.

In order to increase differentiation, WesternGeco's strategy is to develop proprietary seismic-related software and hardware products to support the seismic service business.

The technology covers the range from high-speed embedded sensors and data-collection networks to advanced instrument control and data processing on super-computers and large clusters. The main features of this type of system are real-time, large data volumes and CPU intensiveness. These systems are also highly technology bound and will therefore face the prospect of being continuously upgraded for the next 5 to 10 years.

4.5.2 Description of the existing system

The system to be focused on in the Modelplex use case, was the onboard *Spread Management System*, which boots up, controls and monitors the spread instrumentation. The spread instrumentation is a very complex network of distributed computers, various devices and sensor nodes (hydrophones).The time-lapse, or 4D, seismic method involves acquisition, processing, and interpretation of repeated seismic surveys over a producing hydrocarbon field. The objective is to determine the changes occurring in the reservoir as a result of hydrocarbon production or injection of water or gas into the reservoir by comparing the repeated datasets.

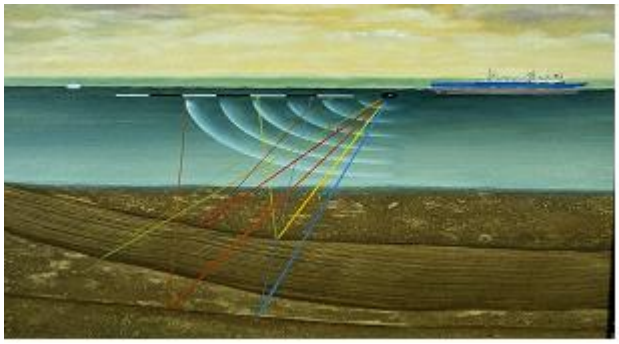


Figure 25: Large Scale Data Intensive Geological system

To attain this goal, seismic acquisition has implied the development of a complex heterogeneous system :

- Data rate > 100MB/s
- Number of sensors > 100,000
- Disk capacity > 60 Tbyte
- Compute power > 1 TFlop
- Network of computers > 500

4.5.3 Complexity of the existing system

The complexity of this system relies on five dimensions:

Size

The size of a deployed seismic spread is large. Each spread typically will be composed of more than 100,000 sensors. The spread will also contain up to 1,000 distributed computers in various devices.

Heterogeneity

In the system as a whole (where Spread Management is a part), different technologies are used both at the hardware and software level. The set of used operating systems includes Embedded Linux, Linux and Unix. Programming languages including Java, C# and C++ and communication mechanisms include CORBA, sockets and Java RMI.

Distribution

The nodes of the spread managed by the Spread Management system are deployed in the sea along cables (streamers) that are towed behind a vessel. There are up to 1,000 distributed computers (nodes) in a spread of 10 cables. Each cable might be up to 12,000 meters long.

Dynamicity

In case of some faults in the equipment, parts of the streamer (streamer section or node) have to be replaced in situ while the operation continues.

Autonomy

Each device is developed as an autonomous computer with no centralized logic. If one device dies it does not influence the other devices or the data traffic. There are multiple routing schemas. Autonomous devices also support better scaling, by transparent supporting adding more devices to the network

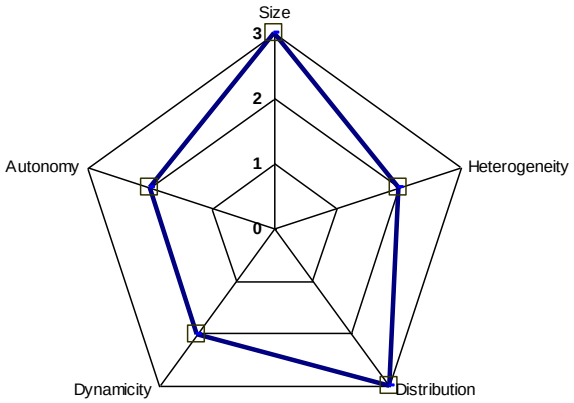


Figure 26: Complexity Dimensions of the application

4.5.4 Problem summary

In the case of legacy applications which are not modellized, with little or no documentation, the initial developers of these applications are no longer available. How can the knowledge of these applications be retrieved to support bug corrections and to develop new functionalities?

Two main constraints had to be taken into account:

- WesternGeco wished to turn the extracted knowledge into UML models
- The knowledge to be retrieved and the way to create a UML model from it could vary depending on specific needs

These constraints led us to design and develop a flexible solution, easily customizable, to extract specific points of view out of the existing system and to create specific UML representations.

4.5.5 Architecture overview

The solution put in place is a sequence of tools. The modularity of the migration chain aims at facilitating the reusability of individual components and thus augments the flexibility of the global migration chain.

The standard migration chain is composed of a discoverer to extract a model from sources, and model transformations to bridge between different metamodels. To facilitate the reusability of components KDM was chosen as the pivot metamodel of the migration chain: rather than developing two transformations to UML (one from Java and another from C#) the KDM to UML transformation can be reused, connected to initial discoverers from Java or C#. In addition, KDM could be used as input to transformations to metamodels other than UML.

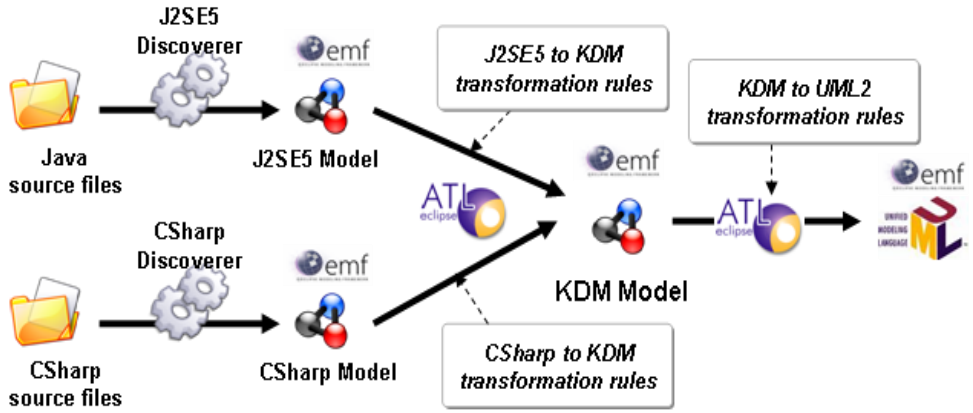


Figure 27: Standard transformation chain from Java and C# to UML

The first version of the solution aims at retrieving different kind of dependencies between existing components.

Thus, to obtain a filtered model with dependencies from a selected element, the Java and C# discoverers were enhanced to obtain a KDM model with Action elements (elements related to the behaviour of applications), and an additional transformation was inserted into the existing transformation chain. This transformation explores the KDM model to identify additional dependencies which are added to the KDM model. Based on KDM, it can be used with models coming from Java or C#.

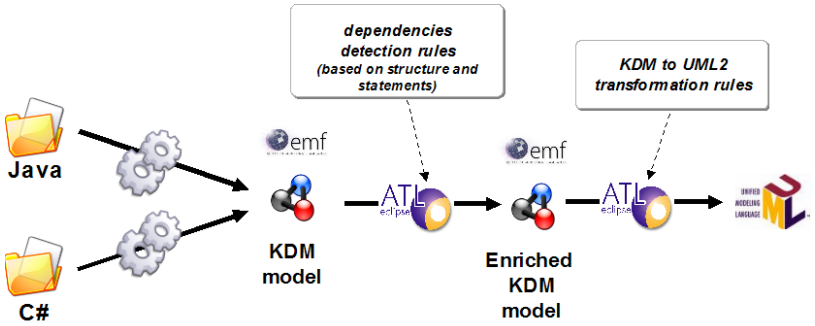


Figure 28: Extended transformation chain from Java and C# to UML

4.5.6 Solution

The first version of the solution provided to WesternGeco supports two points of view both represented with UML models:

1. To have a complete view of a legacy application, a full UML model is provided to show the structure of the legacy application. Modelling tools provide an automatic generation of class diagrams from this model (for example with Eclipse UML2Tool).

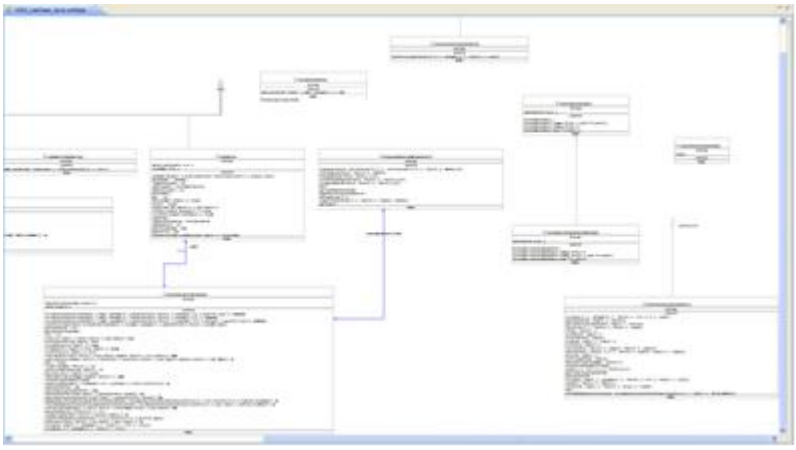


Figure 29: Standard UML model extracted from Java or C# source code

2. To visualize dependencies from an element, a model filter is provided and information extracted from intermediate models such as the KDM model. The main idea is to extract dependencies, of course from the structure, but also from a profound analysis of the application. The KDM model is a detailed representation of the application, so we can explore all references to external elements and represent them as "Dependency" links in UML models. Finally, the UML model is optimal: it contains the strict minimum elements required to describe the dependencies graph of a Java or C# class selected in the development environment.

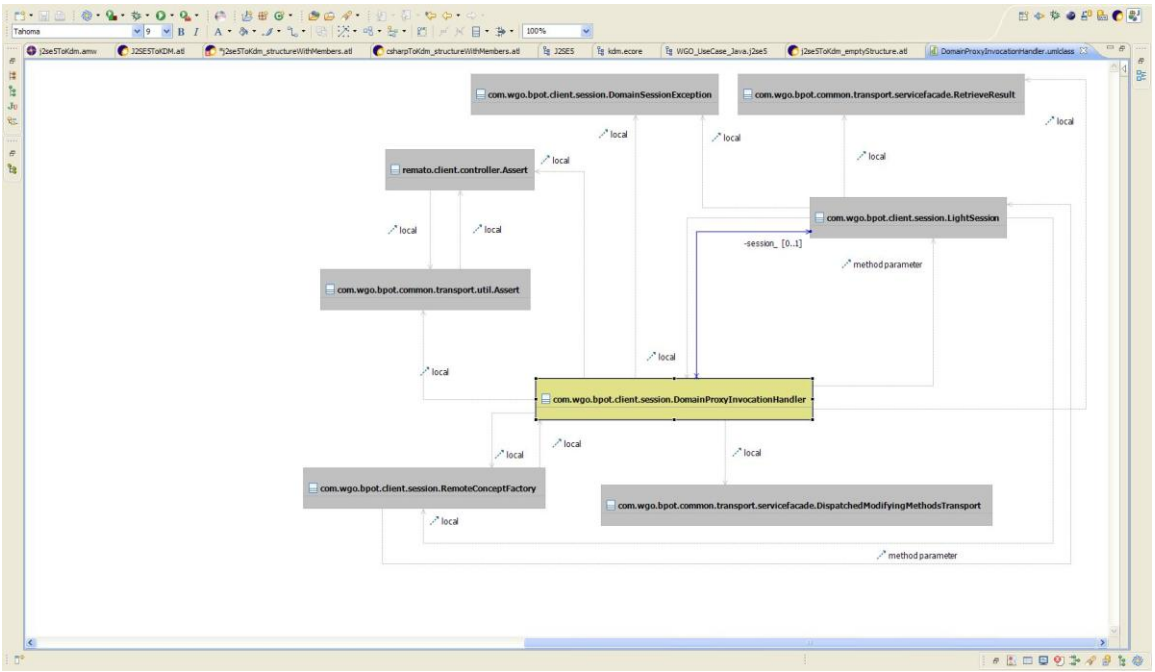


Figure 30: Result of the analysis from a Java class

All dependencies are represented by attributes, associations or UML dependencies named "local" or "method parameter". For example, a UML dependency named "local" represents a dependency in a method block discovered from a local variable or a static method call.

An option allows the defining of recursion level of filter to adapt the UML model to the complexity of the class to represent.

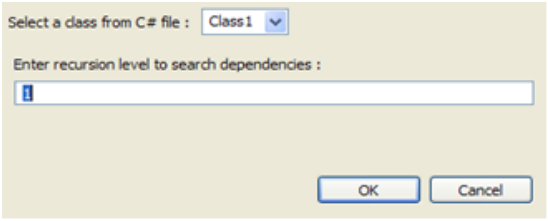


Figure 31: Selection of recursion level

These tools have been delivered to WesternGeco to extract information from two applications:

- An application written in Java: 591 files for a total of 75,638 lines of code
- An application written in C#: 1196 files for a total of 341,326 lines of code

They are also available in the Eclipse MoDisco project as use cases. They aim at demonstrating how to construct a discovery tool from existing discoverers, and to customize it to fit specific needs.

All the sources of the plug-ins are available in the SVN repository of Eclipse MoDisco project.

4.5.7 Roadmap

To support new understanding needs, the standard migration chain can be easily extended with additional transformations or new discoverers. Based on open standards (Java, EMF, KDM and UML) the initial components have been contributed to the MoDisco platform and the development of additional components can be undertaken with any tool compliant with those standards.

With the same approach, new components can be implemented to retrieve real type of attributes when declared type is an interface.

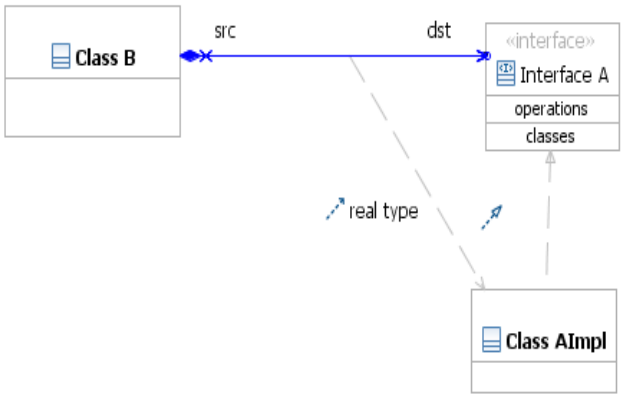


Figure 32: Retrieve real type of attributes

5. Conclusion

The use of model-driven technologies on major migration projects conducted by Sodifrance has proven the benefit of this approach. Based on metamodeling standards, the tools used to extract the knowledge from existing applications, to transform this knowledge into new paradigms and architecture, and to regenerate the application according to specific technical platforms and patterns are more flexible and reusable. The new MoDisco platform will bring all these benefits into an open extensible framework containing model-driven reverse-engineering tools and components. This platform will be use case oriented to deliver concrete tools for real life software modernization scenarios.

6. Acknowledgments

The work presented in this paper has been carried out in the context of the EU FP 6 project Modelplex, funded by the European Commission.

7. References

[SNETS] Jean Bézivin, "sNets: A First Generation Model Engineering Platform" in: Satellite Events at the MoDELS 2005 Conference, LNCS 3844, Springer, 2006, p. 169-181.

[KDM] KDM Final Adopted Specification ptc/06-06-07

[OMG, 2007] Architecture-Driven Modernization: Transforming the Enterprise, by Dr. Vitaly Khusidman and William Ulrich (http://www.omg.org/docs/admtf/07-12-01.pdf)

[UML] Unified Modeling Language (http://www.uml.org)

[MODISCO] Home page of MoDisco project (http://www.eclipse.org/gmt/modisco/)

[MODELPLEX] MODELPLEX – MODELling solution for comPLEX software systems, IST 34081. (www.modelplex-ist.org)

[MIA-STUDIO] Mia-Studio suite (http://www.mia-software.com/en/products/suite-mia-studio/)

[MIA-INSIGHT] Mia-Insight suite (http://www.mia-software.com/en/products/suite-mia-insight/)