Forms2Net - Migrating Oracle Forms to Microsoft .NET

Luis Andrade[1], João Gouveia[1], Miguel Antunes[1],
Mohammad El-Ramly[2], and Georgios Koutsoukos[1]

[1] ATX Software S.A, Rua Saraiva de Carvalho, 207C, 1350-300 Lisbon, Portugal
{luis.andrade, joao.gouveia, miguel.antunes,
georgios.koutsoukos}@atxsoftware.com
[2] Department of Computer Science, University of Leicester,
University Road, Leicester, LE1 7RH, UK
mer14@le.ac.uk
http://www.cs.le.ac.uk/~mer14

Abstract. Forms2Net is an ATX Software commercial reengineering tool that automatically converts Oracle Forms applications to the equivalent .NET (C#) ones, with approximately 75% rate of automatic conversion. From the reengineering and transformation theoretical viewpoint, Forms2Net falls in the general category of language-platform conversion tools. As theory and practice indicate, for such tools to be effective, there are two major issues that must be handled: (a) the resolution of the semantic gap between the pair of source-target languages and (b) the resolution of the dependencies (e.g., API dependencies) on functionalities provided by default by the source platform or on programming idiosyncrasies of the source platform (in this case Oracle Forms). This paper presents the important practical aspects of Forms2Net and the underlying technology. We discuss the semantic gap between Oracle Forms and .NET forms and the design principles and solution strategies used to bridge this gap.

1 Introduction

Software application transformation is an active area in research and practice [10,12]. For many reasons organizations decide to migrate from one language to another, from a platform to another, from an operating system to another or a combination of these. The reasons for such migration are diverse ranging from moving away from an obsolete technology, to creating an integrated corporate information system, to moving from client-server architecture to a multi-tier or three-tier architecture. A few examples of such migration are [6, 7, 11]:

- Converting to a newer version of a language (COBOL 68 to COBOL 85),
- Converting from a language to another (COBOL to C or Java)
- Migrating an application to a different system that supports a different dialect of the same language (Cobol on IBM Mainframe to AS/400 Cobol)
- Migrating from a file system storage or a hierarchal database to a relational database (from VSAM files to DB2)
- Converting from an application framework to another (Oracle Forms applications to .NET applications in VB or C# or to Java applications for J2EE).

R. Lämmel, J. Saraiva, and J. Visser (Eds.): GTTSE 2005, LNCS 4143, pp. 261 – 277, 2006.

The width of the semantic gap between the source and target languages and/or platforms decides the feasibility and complexity of the conversion. The wider the gap, the less feasible, more complex and less automated the conversion is. This paper presents the challenges faced, design decisions made and solution strategies implemented in Forms2Net [1], a commercial tool for transforming Oracle Forms applications to C# applications for .NET. It gives an overview of Oracle Forms platform and discusses the reasons for converting Oracle Forms applications to .NET ones, the challenges in this conversion and the semantic gap between both frameworks (Sections 2 to 4). Then, it explains the strategies and design principles followed in designing Forms2Net (Section 5) and the conversion approach implemented in Forms2Net (Section 6). Next, the related work is discussed (Section 7). Finally, some conclusions are drawn (Section 8). Oracle Forms might be referred to as Forms only in the rest of the paper.

2 An Overview of Oracle Forms Applications

Oracle Forms is a 4GL rapid database application development environment plus a runtime environment where these database applications run [13]. Table 1 summarizes the elements of a Forms application. [9]

Figure 1 shows the structure of an Oracle Forms application from a developer's viewpoint and the relationships between its main components. The arrows represent a general relation that can be association or aggregation.

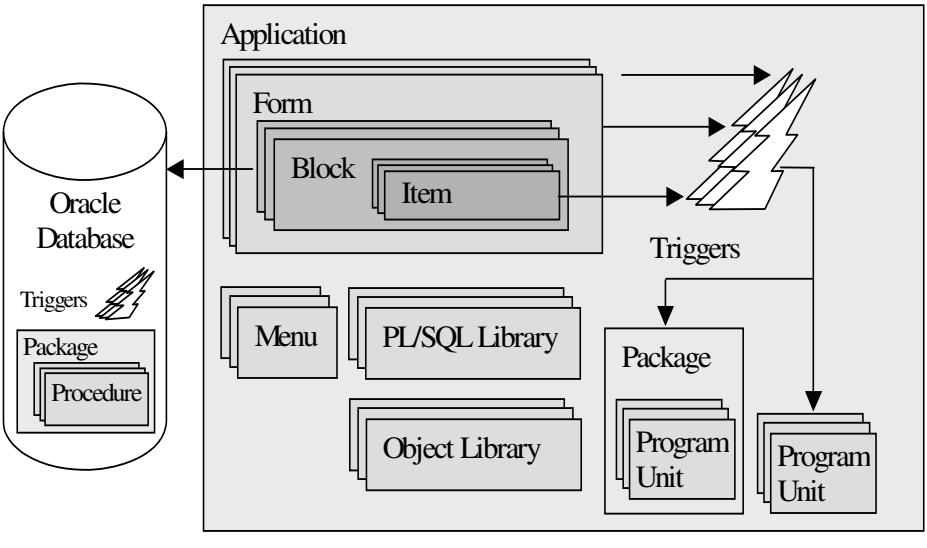

Fig. 1. The Structure of an Oracle Forms Application from a Developer's Viewpoint

Table 1. The Elements of an Oracle Forms Application

Oracle Concept	Description
Form	A Form is a collection of objects and code, including windows, items, triggers and program units. A form can include any number of separate windows.
Window	The usual window concept. A form may have several windows that are closely related.
Canvas	A canvas is a content area placed inside a window. A window may display several canvases.
Block	Represents a logical container for grouping related items into a function unit for storing, displaying, and manipulating records. Only item objects contained in a block are visible in the application interface.
Item	Items display information to users and enable them to interact with the application. Item objects include the following types: button, check box, display item, image, list item, radio group, text item and/or user area, among others.
Trigger	Represents a block of code that adds functionality to an application by one or more PL/SQL statements. A trigger object is associated with an event.
Program unit	Represents a named PL/SQL function or procedure that is written in a form, menu, or library module. It allows the reusability of code across different trigger behaviors.
Package	A package is a PL/SQL construct that groups logically related types, objects, procedures, and functions. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary.
Record Group	Represents a set of column/row values similar to a database table. However, unlike database tables, record groups are separate objects that belong to the form module in which they are defined.
LOV (list of values)	A LOV object is a scrollable popup window that provides the end user with either a single or multi-column selection list. It represents a set of column/row values similar to a database table.
Alert	An alert is a modal dialog box that displays a message notifying the operator of some application condition.
Visual Attribute	Represents a named visual attribute that should be applied to an object at runtime. A visual attribute defines a collection of font, color, and pattern attributes that determine the appearance of an object.
Menu	A collection of menus (a main menu object and any number of submenu objects) and menu item commands that together make up an application menu.
Library	A collection of user-named procedures, functions, and packages that can be called from other modules in the application.

Note in Figure 1 that the database may have some elements beside data, which are triggers and packages of procedures that are left untouched by the Oracle Forms application migration process. From the presentation or user interface viewpoint, an Oracle Forms application looks like Figure 2 [6]. Frames in Oracle Forms are visual containers similar to Group Box Controls in Windows Forms.

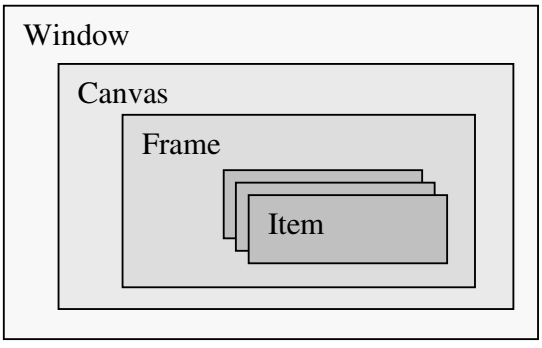

Fig. 2. The Organization of an Oracle Forms Application from a Presentation Viewpoint

3 Why Convert Oracle Forms Applications to .NET?

A .NET Windows Forms application in its essence is based on similar concepts for the presentation elements (Forms, Panels, Controls, Event handlers) and the code elements (class libraries). These are the building blocks of a .NET application and there are different ways to use them and organize them to make an application – this is the role of the application architecture. Microsoft makes available several application blocks based on the .NET Framework, but they are still low-level isolated blocks targeting a specific task/feature (logging, cache, exceptions, data access, etc.) [5].

Oracle Forms is already a legacy environment for Forms applications. J2EE and .NET are the major platforms to develop this kind of applications nowadays. They both have their own strengths. Forms2Net targets only the migration to .NET Framework. Although Oracle Forms was considered a powerful and productive environment for application development, the resulting applications lack the flexibility and the interface features available in modern applications. Several other factors may influence the decision to move from Oracle Forms to .NET:

- Easy to find and cheaper workforce
- Cost savings (database costs)
- Increased development productivity
- Platform harmonization/migration.
- Customer/partner alignment;

The conversion of legacy Forms applications is also an opportunity to integrate existing legacy applications into a service-oriented architecture if one is being constructed as the backbone of the company's information system. Applications may provide new (migrated) services to other applications or reuse already built services to replace or add functionality.

Forms2Net provides options to adapt the converted applications to multi-tier environments, enabling an easy path to an integrated service environment. Currently, Forms2Net supports Oracle Forms version 4.5 to 6i.

4 The Semantic Gap

As mentioned above there are several similarities between Oracle Forms and .NET Windows Forms, as well as some important and relevant differences that make an automated migration a complex process. For better understanding of the gap between the two approaches we present here the main differences.

4.1 Interface Elements

The most common interface elements are present in both platforms (windows, panels, labels, text boxes, combo-boxes, check-boxes, etc.). Nevertheless some differences also exist; an example is the radio group. In .NET Windows Forms, a radio group is created by a set of `System.Window.Forms.RadioButton` instances belonging to the same visual container. However, in Forms migration it is also necessary to allow the creation of radio groups that have no visual relation, i.e., they can be children of different containers and still act as a group of mutual exclusive radio buttons. In general, .NET framework provides richer pre-defined controls for better user interaction, e.g.:

- Data grids with scrolling/column sorting
- Calendar date pickers

The major semantic gap to be solved is to correctly map Oracle Forms multi-record display to a data grid preserving the associated interaction behaviour (validations, triggers, etc.). For instance, .NET data grids need to be extended with new column types (e.g., Combo Boxes) and corresponding event validation (the `validate` event) on data grid cells. Also, there is no direct mapping between Oracle Forms REQUIRED property and .NET control validation mechanisms. Hence, suitable extensions to the .NET control classes should also be provided for such Oracle Forms properties.

4.2 Data Organization

Oracle uses the data block concept to represent simple data (items) or data collections (table rows) that may be mapped to database entities. By using this concept, a lot of database read/write/commit behaviour is pre-defined in Oracle Forms without writing too many lines of code. This was one of the basics of 4GL applications, which results in minimum coding effort when following the typical patterns of Forms applications.

The major semantic gap to be solved is to ensure that access and management of data is done in a simple and uniform way, consistent with the original semantics and allowing an easy mapping of PL/SQL instructions (PL/SQL is Oracle's SQL language, with additional language constructs). For example, in Forms applications direct calls to database stored procedures are allowed (Listing 1) and, in fact, are a common practice. In .NET this is not possible and therefore a suitable mechanism must be devised (e.g., wrapping as in Listing 2) for accommodating this. The same applies to the Oracle cursors that are not present in .NET.

```
:EMP_CREATE.EMPNO := GET_NEW_EMPLOYEE_ID;
```

Listing 1. PL/SQL Call to a Stored Procedure

```
public class StoredProcedures {
       public static NullableDecimal GetNewEmployeeId() {

          IDataCommand cmd =
          DbManager.DataAccessFactory.CreateDataCommand
          ("GET_NEW_EMPLOYEE_ID", DbManager.DataBaseFactory);

          cmd.AddReturnParameter(typeof(NullableDecimal));

          cmd.Execute();

          object _retVal = cmd.GetReturnValue();
          return _retVal == DBNull.Value ? NullableDecimal.Null
          : Convert.ToDecimal(_retVal);
          }
  }

  //.NET invocation for previous stored procedure wrapper
  Model.EmpCreate.Empno = StoredProcedures.GetNewEmployeeId();
```

Listing 2. A Wrapper for the Stored Procedure and the Respective Invocation in .NET

4.3 Events

Oracle provides the Trigger concept. Triggers are events that are propagated up the object hierarchy as a chain of responsibility. In .NET the event concept is also provided but event propagation is flat, i.e., all event handlers of an event are fired and there is no event propagation from a child component to its parent component. The existing 'alphabet' of events used in Oracle is significantly different from the ones available in .NET, although some similarities may be found.

The semantic gap to bridge here is to define a correct mapping between both sets of available events, in a way that preserves most of the original semantics. By semantics here we mean 'when' the event happens, and its 'purpose'. For instance, Oracle Forms triggers can be organized in two categories: Model Triggers, fired by operations made on data or by data manipulation operations (ON-POPULATE-DETAILS, ON-COMMIT, ON-INSERT, ON-DELETE) and View Triggers, fired by user interaction at the UI level (WHEN-BUTTON-PRESSED, WHEN-NEW-BLOCK-INSTANCE, WHEN-NEW-ITEM-INSTANCE). Adequate mappings for .NET, such as the definition of such events, the event handlers and the event registration code together with the corresponding method signatures, must therefore be devised.

The mapping should be complete when semantic preservation is guaranteed, and partial when semantics are not the same. Partial here means that the mapping is provided as a possibility that should be completed during the manual 'completion' phase by a Forms2Net user. As an example consider the navigation between the several components of a Form (Next-Block, Previous-Block, Next-Item, etc.). It is common to have handlers for these operation triggers that just prevent the operation from execution or display an error message (e.g., forbid the navigation from block B1 to block B2).

.NET applications don't use this kind of navigation restrictions. If such behaviour is required, the conditions to enable/disable the relevant controls, must be performed in the code completion phase. Finally, some events are also discarded during the process. This aspect is closely related to the following one.

4.4 Behaviour

Oracle Forms runtime has a huge set of runtime features and implicit behaviour. An example of this is the behaviour associated with Execute and Commit actions that loads / saves the data being edited in the forms according to the form block types and definitions. Only some of these features are present by default in .NET framework. Some others are not relevant because .NET applications have different patterns of behaviour. For instance, validation of data in Oracle Forms is done in a complex way, with several levels of validation (item, block, form) that occur when some actions are taken. Standard validations in .NET applications are simpler, performed on single controls, when editing is finished.

Furthermore, in some cases, a Forms application may have a lot of code that overrides, controls and disables Oracle Forms implicit behaviour.

Mapping the behaviour correctly between the two approaches is the most challenging semantic gap to solve. This is typically where some rules and conversion tables may be used, but human effort is required in the migration process to check or complete the automated conversion.

4.5 Language

PL/SQL control constructs are not so different from C# constructs. The SQL part is what makes the difference, including embedded database operations (queries, cursors, etc). One of the gaps to be solved is correctly migrating all the SQL code instructions into corresponding ones using the .NET databases access infrastructure. However, the major semantic gap to be solved is the ability to work with null values on every PL/SQL data type (numbers, dates, booleans, etc.), that has no counterpart in .NET Framework 1.1. In .NET 1.1, data types (decimal, boolean, integer, etc.) do not accept null values. The only type that has this ability is string. Oracle Forms code is written with the implicit existence of null, and a straightforward transformation for C# code will not have the same behaviour, without adding lots of constraints and different rules. Therefore, the concept of Nullable type should be introduced to cope with this semantic gap. However, .NET Framework 2.0 has support for Nullable types through the `System.Nullable<T>` generic. For migration tools, such as Forms2Net, this

implies a strategic decision: either change the current code generation to incorporate such a feature or just alter the current implementation of the Nullable types support library (for instance, via inheriting from the .NET Nullbale generics). In other words, a decision has to be made on whether to abandon a support library and hence the corresponding support for Visual Studio 2003 or continue with the support library and support for VS 2003.

Another gap is that Oracle Forms has support for object inheritance. Objects can inherit from other objects defined in the same module or from objects defined in a different module. However, .NET does not support multiple inheritance. One solution is to deal with inheritance at the module level and only support one base module for each module being converted. The conversion tool user can then configure the base modules for the modules being converted. Then, during the module conversion, an object is considered as inherited if it was inherited from the module's base module. This implies that any objects inherited from a different module will not be considered as inherited.

5 Forms2Net Design Principles and Strategies

Before describing how Forms2Net deals with the semantic gap between the two platforms in the next section, it is necessary to describe the design principles enforced throughout Forms2Net and the solution strategies adopted. Three design principles were adopted:

- **100% Pure .NET Code.** The generated code should be pure .NET code following Microsoft's Best Practices. It should only bridge the semantic gap problems with solutions that are 100% based on .NET Framework.
- **Preserve the code structure as much as possible.** Although the converted application architecture has significant differences, the structure of the original code units should be preserved as much as possible to keep the functional model of the original application and to ease comprehending the converted code.
- **Do not impose key conversion decisions upon the user.** Keep it simple**.** If there are several possible alternatives to a particular semantic gap problem, the program maintainers should decide what to do. This is very important because some semantic gap problems may require minor changes to be made and it is important to allow the future developers to choose how to perform them so that the final result is the desired one.

The semantic gap problems were generally addressed by the following four different but related strategies:

- Well Defined Target Architecture.
- Semantic Oriented Migration.
- Well Documented Migration Process
- Lightweight Support Libraries.

5.1 Well Defined Target Architecture

Having a well-defined target architecture simplifies the code conversion process as it allows having well-defined conversion rules for certain objects, code patterns, and semantic gap problems. Although being different from the original, the target architecture adopted by Forms2Net was defined so that most of the concepts existing in the original application could be represented. The main objective was to build a semantic map or bridge between the original application architecture and the target architecture. However, this does not mean that there are one-to-one mappings between the artefacts of the original and target architectures. On the contrary, most mappings are one-to-many which means that an artefact in the original architecture is represented in the target architecture by two or more artefacts, their relations and their behaviour. Forms2Net adopts a target architecture based on the Model-View-Controller pattern [3], with some additional concepts that are particular to Forms applications.

5.2 Semantic-Oriented Migration

Semantic-oriented migration means that Forms2Net does not focus only on the conversion of PL/SQL code into .NET. It works on a semantic level by taking into account the target architecture. Also, Forms2Net was designed so that specific plug-ins can be developed to convert specific code constructs. The following further illustrates these points:

- **Original artefacts are converted and rearranged in order to fit in the target architecture.** The conversion process works from a model of the target architecture created from the original application, i.e., the first conversion step is to map the original architecture model into the target one.
- **Certain code patterns are recognized and transformed into more adequate code patterns.** For instance, PL/SQL code routines are analysed and depending on the manipulated blocks and Oracle Forms built-ins, the converted routines are parameterized in order to reduce the dependencies between the code and the model, allowing the business logic to be easily identified and isolated.
- **Conversion of Oracle Forms runtime built-in calls can be performed on one-by-one basis.** By using the extensible architecture of .NET, it is possible to develop new plug-ins to convert a particular usage of a particular Oracle Forms built-in. Since the number of Oracle Forms built-ins is very high, the extensible architecture of Forms2Net allows built-ins conversions to be dealt with in an incremental way, starting with the most used built-ins and adding new conversions when necessary.

5.3 Well Documented Migration Process

Every time the semantic gap cannot be solved or when there are several alternatives to solve a particular problem, comments are generated in the code that point out to the user the possible directions to be taken. These comments have links to a generated

migration guide specific for each form. This generated migration guide is customized for the converted forms and the specific issues encountered in the original code, and refers to the more generic documentation that is distributed with the tool.

By promoting a well-documented migration process, Forms2Net avoids imposing sensitive migration decisions on the user, and at the same time eases the code completion process by supplying code comments and a migration guide that help the user perform the necessary code changes.

5.4 Lightweight Support Libraries

Forms2Net supplies two lightweight support libraries that the converted code uses to help reduce the semantic gap and preserve the original code structure:

- **Application Data Layer Library.** ADO.NET is a set of .NET Framework classes containing the data access technologies used to manipulate databases through specific ADO.NET providers. This library is built on top of ADO.NET to allow code to be independent of the provider. It provides several other features like:
  - Simple classes to perform database operations: select/update/delete commands, cursor operations, etc.
  - Alternative interfaces to make the converted database manipulation code simpler (less verbose) while maintaining its original structure;
  - Manipulation of database null values in a transparent way
- **Application Support Library** is a set of utility classes that help reduce the semantic gap when there is not an alternative in the .NET Framework and when the solution to the problem is straightforward. It is composed mostly of user interface components that extend .NET Windows Forms Framework; and most of them are components that any user of .NET Windows Forms will eventually need. It is possible to find different flavours and implementations on the World Wide Web for these components, supplied by third party vendors or even as open-source code. The library uses .NET Windows Forms extension mechanisms (class inheritance or `IExtenderProviders` implementations) to extend .NET Windows Forms native controls.

6 Architectural Centric Conversion in Forms2Net

Forms2Net follows a 4-phases architectural centric conversion approach:

- **Target Architecture Definition.** This phase defines the architectural elements, their characteristics and relations.
- **Architectural Mapping.** In this phase, an architectural mapping of the source application elements into the target architecture is performed. Original application elements are rearranged and mapped into the target architecture according to specific rules.

- **Artefacts Generation.** In this phase, all the static architectural elements like models and views and all their components are generated into the target platform (Windows Forms).
- **PL/SQL code conversion.** In this phase all PL/SQL code existing in Forms' triggers and program units is converted into .NET taking into account their localization in the target architecture.

6.1 Target Architecture Definition

Forms2Net provides an architecture based on the well-known MVC (Model-View-Controller) pattern [3]. Forms2Net MVC architecture for migrated Oracle Forms applications decouples data access, business logic, and data presentation in a well-organized and scalable structure, mapping Oracle Forms concepts into core .NET framework concepts, using Microsoft's best practices.

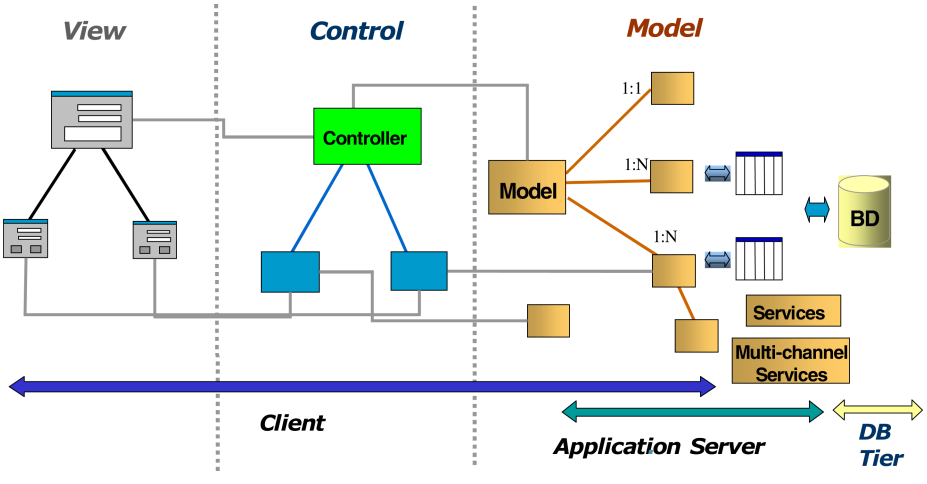

Fig. 3. MVC Architecture Targeted by Forms2Net

Using the MVC model as in Figure 3, the resulting application's design minimizes the interdependencies among the different parts. The role of each element in the MVC model architecture adopted by Forms2Net is described below.

- **The Model** component maintains and manages the information manipulated by the form. It manages the communication with the database, using Datasets[1] to store the data.
- **The View** component's role is visualizing the model state. It is responsible for handling user interaction.

[1] The DataSet is a component of the ADO.NET architecture, which is an in-memory cache of data retrieved from a data source. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects.

- **The Controller** is responsible for the relation and coordination between the other two components, as well as for the form's functional interface:
  - It manages user interactions by mapping user actions and events into application responses.
  - It translates the actions within the view to actions performed on the model.

Note that although the MVC architecture is the only target architecture currently supported by Forms2Net as an architectural centric migration tool, other architectures could be supported as well.

6.2 Architectural Mapping

In this phase, Oracle Forms objects of the source application are mapped into the target architecture. Table 2 shows some examples of how Oracle Forms objects are mapped into the target MVC architecture. Note that some of the mappings are one-to-many. For instance, each Oracle Forms Block is mapped into one *model* and one *controller*. The *model* maintains the block's state, whereas the converted code for the block's triggers and all of its items' triggers resides in the *controller*.

6.3 Artefacts Generation

In this phase architectural artefacts are generated into the target platform. Table 2 shows some of the mapping into .NET Windows Forms platform. Forms2Net is designed to be independent of the target platform. For each artefact there is a configured generator. Different platforms are supported by configuring different sets of generators. At the present moment generators exist for both Windows Forms and Web Forms platforms although only Windows Forms generators are available commercially. Also, Forms2Net design allows using different generators for the same kind of artefacts. For instance, this allows having generators for Microsoft .NET Windows Forms controls or generators for third-party .NET Controls suites.

6.4 PL/SQL Code Conversion

At this phase, all PL/SQL code is converted into .NET code. This conversion is not limited to language translation; it also applies some reengineering techniques in order to obtain better quality and higher conversion rate[2]:

- **Code routines Parameterization.** In Oracle Forms, a block item can be referenced anywhere in PL/SQL code (trigger, program unit, etc.). To reduce as

[2] In order to associate as much as possible the percentage of conversion with the effort needed to manually complete the application, the conversion rate measure adopted by Forms2Net is the percentage of the number of Oracle objects (e.g., interface items, Forms, triggers, properties, built-ins etc) that are supported for a given application and not the lines of code or number of Forms converted.

much as possible references to the Model objects, Forms2Net uses control flow analysis to parameterize the generated services methods and controller methods.

- **Code pattern recognition and transformation.** Certain code patterns are recognized in the original code and transformed into code patterns that are more suitable for .NET. Listings 3 and 4 illustrate a transformation of a block iteration pattern.

Table 2. The Architectural Mapping of Oracle Forms Objects into The Target MVC Architecture and The Native Target Objects of .NET Windows Forms Platform

Oracle Forms Object	.NET Code Replacement		
	Model	View	Controller
Window		`System.Windows.Forms.Form` subclass	Windows Controller class
Canvas		`System.Windows.Form.UserControl` subclass	UserControl Controller class
Block	.NET Model class	If the block has multiple records, a Data grid will be generated.	Controller class with all the block's and item's triggers
Item	Properties of the Model class (columns of DataSet table if the block is data-based)	Instances of .NET Framework `System.Windows.Forms.Control` class	
Form Module	A .NET Model class that aggregates all the block models. ADO .NET typed DataSet with all the DataTables, relations for all database blocks defined in the converted Form module.		Controller with all the form triggers. This is the base class for all the Window's Controllers
Relation	DataRelation in the DataSet		Master detail coordination logic
Program Units	Methods in a service class		
Triggers		Event registration and event handlers that call the correspondent Controller methods	A method for each trigger to be called by event handlers from view classes.
LOV		IExtenderProvider component that associates a ChooseValue form to each control (item) that has a LOV property	
...			

```
FUNCTION CALCULATE_REVENUES RETURN NUMBER IS
  total number;
BEGIN
  if :system.current_block != 'ord' then
    go_block('ord');
  end if;
  FIRST_RECORD;
 LOOP
    EXIT WHEN (:SYSTEM.LAST_RECORD = 'TRUE' );
    total := total + GET_ORDER_COST(:ord.ordid);
    NEXT_RECORD;
 END LOOP;
 return total;
END;
```

Listing 3. Original PL/SQL Code for Iterating over The Records of a Block

```
FUNCTION CALCULATE_REVENUES RETURN NUMBER IS
  total number;
BEGIN
  if :system.current_block != 'ord' then
    go_block('ord');
  end if;
  FIRST_RECORD;
 LOOP
    EXIT WHEN (:SYSTEM.LAST_RECORD = 'TRUE' );
    total := total + GET_ORDER_COST(:ord.ordid);
    NEXT_RECORD;
 END LOOP;
 return total;
END;
```

Listing 4. The Equivalent Converted C# .NET Code of a Block

7 Related Work

Software transformation is a multifaceted problem, with many applications and also many challenges, not just on the technical side but also on planning, management and risk-control side [12]. The specific version addressed here is language and architecture transformation, where not only the application will move to a different language and platform, but also its architecture has to significantly change to adapt to the architectural model of the target platform. Architecture transformation is a challenging problem, especially when the gap between the source and target architectures is wide. In this case, as Klusener et al. [4] explain is their discussion of architectural modifications to deployed software, the changes (transformations in our case) need to happen at system-wide level rather than on a per-function or per-module basis. This makes the problem harder and requires creating and possibly integrating advanced

and sophisticated transformation tools. Realizing this need, The Object Management Group (OMG) created and Architecture-Driven Modernization (ADM) Task Force (ADMTF) to create a set of standards to facilitate the interoperability of modernization tools. These interoperability standards are being established in a series of meta-models that facilitate the collection, analysis, refactoring and transformation of existing systems [8].

Having decided on the need for a certain type of transformation, one faces the issue of automated versus manual transformation. Or in more the precise words, the issues of availability of transformation tools, the cost of building such tools, the cost of transformation and the quality of the produced code. Klusener et al. [4] discuss and compare automated vs. manual transformations. They conclude that for any non-trivial transformation project, automation is vital to success, but the issue is how much automation is needed and at what cost. Baxter et al. [2] discuss the requirements of building robust automated tools for "practical scalable software evolution", as they describe it. They present their effort and approach in building DMS, a generic transformation environment and tool generator.

8 Conclusions

This paper presented Forms2Net, a tool for transforming Oracle Forms applications to .NET applications that use Windows Forms. The paper gave an overview of Oracle Forms platform, the motivations for transformation, the semantic gap between both platforms, the design principles and solution strategies adopted, and finally a general overview of Forms2Net implementation. It is important to draw some useful lessons from this experience.

First, despite the similarities of the two platforms, significant semantic differences exist. This makes transformation complex in the sense that there is a considerable effort involved in building an automated conversion tool. Moreover, it is important for similar transformation problems to focus on bridging the semantic gap using semantic transformations rather than trying to just find syntactic mappings between elements of both platforms. It is expected that some manual transformation will still be needed. Our experience advocates the Klusener et al. [4] view that:

"*A fully automatic solution is not always feasible, and it is sometimes not cost-effective. For instance, a modification problem that involves heuristics to determine affected parts of the system often necessitates interactive steps for approval by maintenance programmers. In an extreme case, the automation could be restricted to the generation of a report, which is then applied by maintenance staff in a manual manner. To this end, special interactive tool support can be provided such that programmers basically walk through the generated report and navigate to the affected code locations without ado. Similarly, there is a tension between handling less frequent or highly complex idioms by specific, manual changes per occurrence rather than providing a general rule for the underlying code pattern(s). The decision how much automation is necessary and whether generic modification rules are required has to be made while relating to the technical analysis of the problem at hand, and to the drivers for the project.*"

Second, several code generation techniques and technologies are available in the market or in the open-source community. In a complex process like Forms2Net migrations, one should not rely only on one technique. One should have a master driver for the generation, but then use the most appropriate technique in each situation. External generation configuration and a plug-in architecture for generators are also advisable solutions.

Third, one should give great attention to designing the target architectural model. On a process like this, the architecture model of a generated application is one of the most important issues, not only because it is the centre of the process, but also because it is the base or stable component of the final solution.

Fourth, invest in pattern recognition facilities. A migrated application has a much higher level of quality and satisfaction to the clients when the final result looks like it was 'written' in the target language and is able to use the language constructs in a 'natural' way. This can be highly improved using pattern detection and influencing the generation process according to those patterns.

Lastly, developers like to have control over the code that they will be in charge of. Whenever transformation rules are not clear, i.e., there is no solution or there are multiple-solutions, Forms2Net reports the case in the generated code and gives its user the choice of deciding what to do.

Acknowledgements

The authors like to thank the reviewers for their thorough reviews, detailed feedback and invaluable advice and comments. We also like to thank the editors for the great effort they put in editing and producing this volume.

References

1. ATX Software, Forms2Net. Available at http://forms2net.atxsoftware.com/
2. Baxter, I., Pidgeon, P., Mehlich, M.: DMS: Program Transformations for Practical Scalable Software Evolution. Proceedings of the International Conference on Software Engineering. IEEE Press (2004)
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P.: Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. John Wiley & Sons (1996)
4. Klusener, A., Lämmel R., Verhoef, C.: Architectural Modifications to Deployed Software. Science of Computer Programming, Vol. 54, Issue 2-3 (2005) 143-211
5. Microsoft Patterns and Practices Center: Application Blocks and Libraries. Available at http://msdn.microsoft.com/practices/AppBlocks/default.aspx
6. Microsoft: Solution Guide for Migrating Oracle on UNIX to SQL Server on Windows, Chapter 17 - Developing: Applications - Migrating Oracle Forms. Microsoft TechNet (2005)
7. Mossienko, M.: Automated Cobol to Java recycling. Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR), IEEE Computer Society (2003) 40-50

8. Object Management Group (OMG): Architecture-Driven Modernization Scenarios (2006). Available at http://adm.omg.org/adm_info.htm.
9. Oracle: Oracle Forms Developer's Guide, Release 4.5. Oracle Corporation (1994)
10. Seacord, R., Plakosh, D., Lewis, G.: Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. Addison Wesley (2003)
11. Sneed, H.: Risks Involved in Reengineering Projects. Proceedings of the 6th Working Conference on Reverse Engineering (WCRE), IEEE Computer Society (1999) 204-211
12. Ulrich, W.: Legacy Systems: Transformation Strategies. Prentice Hall (2002)
13. Zoufaly, F., Dermody, P.: Issues & Challenges Facing Oracle Forms to J2EE Evolution. Available at SearchWebServices.com (2003)