From SQLWindows to .NET

An automated process supports developers to port 1.2 million lines of code

by Frank Wuttke

translated from the original German version

The software house *nGroup* ported a complex ERP application from SQLWindows to .NET. The motivation for the porting was to use an up-to-date development environment, integrate ready-made components in their own software, and to make customizations easier for the customer.

What to do with a "Legacy" application that has grown to over 1.2 million lines of code in 17 years and shall be ported to .NET now? For cost and time reasons, manual porting and new development were considered unrealistic options and have been excluded. After long discussions, the development team decided for an automated porting approach followed by manual post-processing. The IcePorter tool was selected, which has been especially designed to convert Legacy SAL applications to .NET [4].

Several differences between the source and the target platform led to numerous and comprehensive post-processing tasks. Polymorphism, DB access, window receive parameters, missing reporting engines, and a special customization module required most of the post-processing work. Porting of existing and self developed COM objects to .NET was much simpler.

After a total of nine months of porting, the next steps for the company are now the planning of the refactoring of the source code and further development of the solution in the new environment, adding features that could not be addressed during the porting phase due to time constraints.

Need for progress

The product *eEvolution* of the nGroup software house in Hildesheim (Germany) is an ERP application that has evolved from the Microsoft Business Solution Apertum and is deployed to more than 1,000 medium-sized companies. To take advantage of the new possibilities offered in recent years by modern platforms and tools, nGroup decided for the porting of their large and comprehensive ERP solution, consisting of several complex applications for a total of more than 1.2 million lines of code. The source platform is the 4GL language SQLWindows by Gupta. For nearly 20 years, business applications in many different areas have been developed using this language. Compared to other tools it offered great benefits at the time, such as embedded SQL and a powerful IDE.

In the last few years SQLWindows was surpassed by more powerful languages like Java and C#. Additionally, qualified employees for SQLWindows are hard to find and the future of the language is uncertain. Therefore, the decision for porting was not made only for technical reasons.

The .NET Framework 2.0 and Visual Studio 2005 were chosen as the target platform. This decision was supported by the underlying technical concepts, the productivity in developing new components and the possibilities of refactoring, as well as the similarity with the look & feel of previous Win32 applications. After the goal was clear the porting team started to explore how to accomplish the mission.

Porting support for SQLWindows

A group of developers, operating under the name Ice Tea Group, started working on a porting solution for SQLWindows years ago and developed a tool named IcePorter as being part of a larger project named The Porting Project (PPJ). The tool fully and automatically translates existing SQLWindows source code to C#

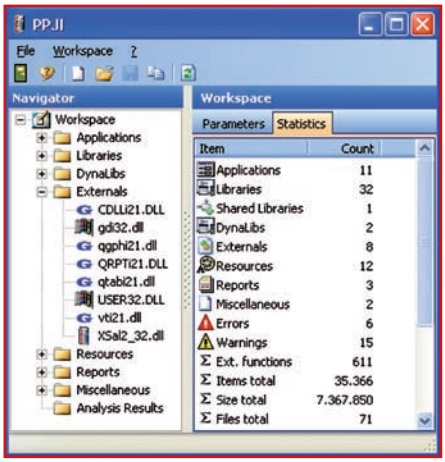

Fig.1: Automated transfering from SQL Windows to .NET with tooling

code based on a porting framework (PPJ/FW). Generally spoken the PPJ/FW contains the complete language implementation of SQLWindows. A comparison of SQLWindows and C# code lines of a simple ported function (Listing 1) shows the benefit of this framework.

The German partner of Ice Tea Group, the company Fecher in Rodgau (Hessen), was hired to execute the porting in close cooperation with nGroup, the producer of the ERP system. A first assessment phase initially promised that a "new" product would magically be generated quickly. Like every bigger project the porting was divided into phases. The following description is specific to this project, but may as well be adapted to other porting projects of this size.

A transparent and structured approach is important for all phases. The first analysis was performed using a tool. The source code was analyzed using the PPJ Inventory. The result contains statistical data that is also used to calculate the cost for the porting. The number of SalCompileAndEvaluate functions, as well as all external DLLs and ActiveX objects, was also immediately known. For "cleaning" the code a log file was used that indicates incompletely qualified references, permutation of data types, and code considered incorrect for the stricter target language. The cleaning also reduced the cost because the price for porting, according to the calculation used by Ice Tea Group, depends on the number of lines of code.

Listing 1

SQLWindows

```
Set sName = ´ngroup eEvolution´
Call SalStrLeft(sName, 6, sname)
Call SalMessageBox(sName, ´Hinweis´, 0)
```

C# .NET

```
sName= "ngroup eEvolution";
Sal.StrLeft(sName, 6, sName);
Sal.MessageBox(sName, "Hinweis", 0);
```

A framework standardizes proprietary functions

One of the project's objectives was to preserve the functions and the design in a new "world". The future application has to be developed further and maintained by the same employees. For these reasons the porting process was organized using a highly structured and automated approach and involved developers with distinctive knowledge of both worlds. Due to the utilization of a framework, the structure of the source code was maintained in the new target system so that the developers of the old system are able to follow the new source code without significant amount of training. There were, however, some complaints by ".NET purists" because they had to familiarize themselves with the old language up to a certain degree. The most serious objection was that the ported code was not object-oriented. Thus *IcePorter* offers a large range of options, allowing for the translation of many constructs into their equivalent object-oriented syntax. Therefore, the code in the previous example is also translated as:

```
sName = sName.Left(6);
```

An extensive ERP is mostly a dialogue oriented program with countless SQL calls.

Database access in SQLWindows is written using inline instructions. The statement

```
Select name1, name2 into :sName1, :sName2 from
                                    kunde where…
```

selects Name1 and Name2 into the variables sName1 and sName2. The porting framework does not use a proprietary database protocol to submit this statement to the database. SQL code is ported using the same syntax as in the source code. The porting framework performs substantial standardization work when executing these statements because it converts every SQL command to standard ADO .NET calls. The database connection can be configured from the outside of the application. For applications that have only used the SQLBase database by Gupta so far, a SQL translator module is also available, allowing the ported application to use Microsoft SQL Server or Oracle without further interaction.

Even if C# is generally a more modern language, the possibilities of embedded SQL statements is what makes a 4GL language stand out. Therefore there are numerous functions calls in a SQLWindows application that represent very complex statements without having to program a lot. If you look at the result of the function

```
Sal.TblPopulate( hWndGrid, hSqlHandle, 'SELECT *
                                    FROM COMPANY', TBL_FillAll);
```

It is obvious that the PPJ/FW is not only a temporary library to provide the bridge to .NET. It is a framework that makes 4GL functions available in .NET and that certainly provides valuable services to further developments. The statement above fills a heavily extended VSGrid with all data of the Company table. It wouldn't make sense to replace this function with a native .NET implementation.

Specialties of the source system have to be considered

SQLWindows is a loosely typed system. It is object oriented but doesn't support overloading of methods or private and public variables. On the other hand it supports multiple inheritance and late bound method calls. The possibility to program COM servers with this language is rarely used. Instead, applications often utilize COM and ActiveX objects, including COM servers written using C++. Some of these COM servers may link to the Gupta runtime, something that is very important to know especially in the context of a porting project. SQLWindows is interpreted and needs a runtime that consists of several DLLs. Certainly a COM server used by the new ported .NET application cannot have any reference to any part of the Gupta runtime. Special challenges with porting are mostly caused by the "carelessness" of the SQLWindows compiler. The poor strictness of the data types leads to the fact that a Boolean can be interchanged with a

Number, as well as WindowHandle with File- and Sql-Handle types. Unqualified references to windows, functions, variables and controls are allowed and may lead to a runtime error when the objects are not available. The message system supports PostMessage, which puts a message at the end of the message queue and delays its processing until after the currently running function has completed, sometimes leading to unpredictable results. One of the most powerful functions available in SQLWindows is SalCompileAndEvaluate. It allows the execution of any SAL statement at runtime.

The porting technology must provide solutions for all the technical possibilities available in the old system. References and data types have to be examined analytically. Some modifications may be automated during the translation. But in special cases the solution is simply a technique that has to be implemented and approved manually. But how do you deal with the challenge to provide a solution in .NET that covers the complete scope of the old system? It is clear that traditional porting approaches, like a semi automated process mixed with new programming, would fail.

Manual post-processing

Since parts of the code were not under version control yet, a version management and a Defect-Tracking-System have been introduced with the new project. These quality-assurance tools have been employed to ensure the success of the porting project. The prepared source code was then passed to the automated conversion tool and further edited until the ported code was able to be compiled error–free in .NET. After this initial phase, the first .NET version of eEvolution had seen the light of day. However, it's not enough to deliver compilable code to the customer, it should also work well and look as good as the original application, if not even better. In order to reach this goal, extensive steps were taken to manually post-process the code that IcePorter couldn't transfer in the first pass. First of all the initial version was checked for design errors. In case IcePorter hadn't worked correctly then "thousands of places" would have to be rewritten. Therefore, it was quite possible that an early version was destroyed immediately. Here the Fecher porting specialists proceeded quite pragmatically: if a compiled and ported version contained more than 20 instances of the same error type, they searched for the root cause and modified IcePorter accordingly.

The porting specialists can also write their own porting plug-ins, for example they can write one that adds additional code before and after all calls to external functions. IcePorter has been modified until the effort to further improve the automated porting process was no longer reasonable compared to the manual post-processing effort. The remaining manual work was done only after the last and final automated conversion pass.

Examples of the post-processing tasks are: check unqualified expressions, modify code that relies on window receive parameters (allowed in SQLWindows but not supported in .NET) , and implement functions from third-party modules ad-

Five questions to the developer

When and why was the decision for the .NET Framework made? Were any alternatives under consideration?

As a company that stands close to Microsoft, nGroup has already been familiar with the .NET framework from other customer projects. The concept is convincing, the integration in Visual Studio is important. The primary goal of porting was the change to an up-to-date IT platform. In our case only Java or .NET was applicable. The effort to port to a Java application in terms of the multi-tier architecture is substantially higher than to port to .NET. A SQLWindows client-server application is 2-tier application (the focus is on the user interface). 4GL functions execute complex statements in a single line. Porting of these complex statements while changing the architecture simultaneously is hardly possible. Thus .NET seemed to be the only feasible solution to guarantee a fast and successful success of the project.

How did the .NET framework positively affect the development? Which were the biggest obstacles to overcome?

The development of the porting project was only possible because of the features of .NET and the high flexibility of C#. With Java it would have been impossible to reach a similar high degree of compatibility between the old and the new world. Overloading of operators, return parameters and delegates were the most important core features of C# that were needed to write the porting framework. WinForms eased the development of visual controls for the PPJ framework. The biggest obstacles were the replication of multiple inheritance using delegates and the overloading of operators.

Did weak spots in the .NET framework become obvious during the conversion or is there anything that Microsoft could have solved differently or better?

In our opinion there are basically two aspects that could have been solved better by Microsoft: 1. ADO.NET doesn't have a base class for exceptions. We had to code known exceptions in the source code and solve the rest via reflection. 2. The design of "short circuit overloading" in the C# compiler is conceptually wrong (Chapter 7.11.2. of the C# specification). It unnecessarily mixes the operators op_True and op_False with the bitwise operators. That makes it impossible to implement a bitwise operator and a logical operator in the same class. This design problem is solved during the translation process by generating an explicit cast to Boolean. Additionally, there were problems with the integration of Oracle databases and the data type NUMBER, and there are problems with the performance of the WinForms editor on complex forms with many controls. The designer sometimes crashes or it's very slow.

Why was the decision made for C# and which language characteristics were considered?

C# was already deployed in other projects and C# offers generally more possibilities than Visual Basic. The necessary mapping of the whole SQLWindows language to the .NET framework and the transformation of the project specific code to .NET would not have been possible using VB.NET and the .NET framework 1.1. In the meantime the porting technology has been made available for Visual Basic 2.0 as well.

What does the future of the project look like? Are there any considerations of using future versions of the .NET framework, and if yes, which features particularly?

An important point for the future is transferring parts of the application into a web-based application and the utilization of the .NET framework or Visual Studio as a part of such a migration.

ded to SQLWindows and not implemented in the porting framework.

And now to the cost

The cost of the manual post-processing work is, on average, 25% of the cost for the total project. This amount may as well be much higher for a project of above-average complexity, which is measured by the amount of modules, embedded external functions, COM servers, third-party *SQLWindows* libraries, and usage of scripting technology.

Extensive testing – with the support of tools if requested

The finalization phase ends when the project is handed over to the client's test team. They have the specific knowledge to test the application in depth. The quality assurance phase must not be underestimated, because all parts of the application have to be fully tested.

Automated tools are also available for this phase – on request the porting partner Fecher can automate the testing of the core areas of the SQLWindows application with the test tool Tosca Commander, part of the Tosca test suite. This tool does not produce scripts but stores the technical information in so-called XML-GUI-Maps instead. It is sufficient to port just the XML-GUI-Maps because the transformation applied by the porting tool is very clear and straightforward in terms of design and functionality. Tosca Commander provides adaptors for both SQLWindows and .NET, so that all automated test cases can be executed immediately after the porting. Therefore an overview of the quality of the new software is guaranteed quickly.

Transferring customer-specific configurations to the target system

A ERP standard software is usually configured to meet each customer's special needs. All these configurations have to be functional also in the new target system in order to ensure a high acceptance and low update effort. No customer likes to pay again for customizations that have already been paid for. In parallel to the conversion of the source code a way had to be found to preserve and convert customer-specific reports. The reporting engine used in the source system is proprietary and does not use a dedicated DB interface. Reports are defined in templates and receive their data from the application through an API. The underlying queries are either stored in the database, if they can be modified by the customer, or they are directly stored in the application. Because the report engine included in Visual Studio 2005 is not a native .NET application we searched for an alternative. The decision was made in favor of List&Label, which has a data transfer philosophy similar to SQLWindows and because the license policy of the manufacturer Combit is very comfortable. The last big obstacle was the so-called Customizer. Using SQLWindows it is not easily possible to customize an application at runtime. Visual customizations of the application's forms, as well as adding new objects and new code, should work for every new release of the system and should not have to be developed from scratch each time. In SQLWindows there is the powerful command SALCompileAndEvaluate, which allows for the execution of external code in the interpreter at runtime. But this was not sufficient. Many solutions to this problem have been created for eEvolution during the years. Unfortunately none of these solutions worked under .NET. Therefore a completely new Customizer was developed in .NET, which met all the requirements that we had wished for since years. User specific changes of the screens, including code written using either SQLWindows or C# syntax with access to all functions, classes and their methods, can be executed at runtime. One of the biggest strengths of .NET, compared to the previous system, was immediately clear in this case: A large developer community offers an even larger choice of tools that you can use at a relatively low price without having to reinvent the wheel every time. For the Customizer we used a.NET control by Greatis[3], which comes with the basic functionality we needed.

Conclusion

After nine months the conversion was completed successfully including final refinements. Amongst other things, we needed a new installer and a new source code management system. Once again the advantages of Visual Studio 2005 were obvious. The support offered by the development environment is close to a miracle. In the original environment such needs had to be resolved in a time-consuming manual way. In retrospect it can be concluded that some parts of the project required more work than expected. But all parties involved would choose the same approach again. The simplifications and the gain in productivity on the software development side allow for a reasonable and effective refactoring at the necessary points. This work will be the next major projects for nGroup, besides the further development of the solution.

Frank Wuttke is CEO of the nGroup and in charge of the Transformationproject on .NET. You can reach him by email wuttke@ngroup.info.

Links & Literatur

[1] www.ngroup.info
[2] www.fecher.de
[3] www.greatis.com/dotnet/
[4] www.iceteagroup.com