FIRST EXPERIENCES WITH A DEVICE SERVER GENERATOR FOR SERVER APPLICATIONS FOR PETRA III

J. Wilgen, DESY, Hamburg, Germany

Abstract

In recent control systems at DESY, a device server generator and framework is used for the production of device servers in the TINE/Java environment. The generator significantly simplifies development and provides a standardized architecture for device server programs.

INTRODUCTION

In previous control systems at DESY, device servers were written in Visual Basic or C. Most of them were customized very individually, although simple one-way template generators existed [1]. Due to the change to Java and object orientation, more powerful means became available which made it possible to generalize device server tasks as far as possible. This reduces effort in producing device servers since programmers can essentially focus on the implementation of the device functionality. In addition, common libraries and standardized program structures are used, which unifies device servers to a certain degree and consequently simplifies their maintenance.

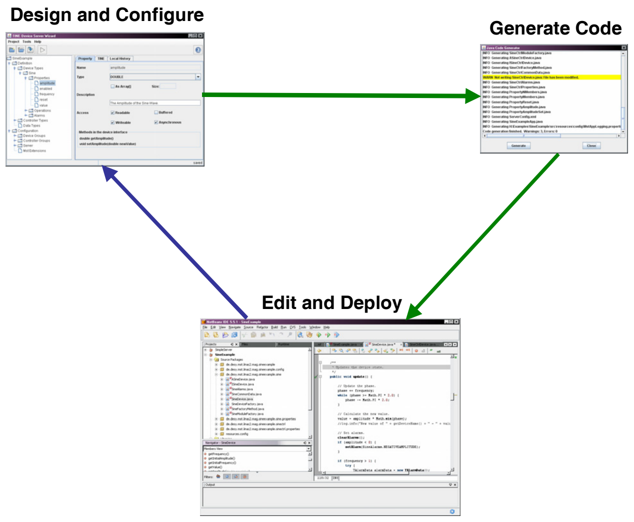

Figure 1: The Production Workflow.

Figure 1 shows the workflow of a device server programmer. Programmers usually begin a device server project by using a GUI program, the device server wizard. With the device server wizard the device interfaces, the groups of devices, the initialization parameters and the TINE [2] interface configuration are defined. All these data together form the device server model, from which code can be generated. The code generator is also part of the device server wizard. The generated code provides the necessary bridges between the business logic, the TINE interface and the generic device server framework. Additionally, Java classes and interfaces are generated for each device interface. Some of them are templates which can be edited or modified by the programmer.

The device specific code can be edited with a text editor or an IDE like NetBeans or Eclipse. Round-trip engineering is possible in case the device interface or the configuration needs to be changed later on. The changes can be made in the device server wizard and the code can be re-generated without destroying modified classes.

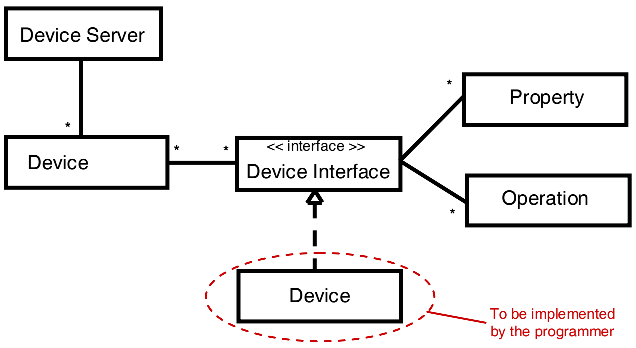

Figure 2: The Basic Model.

The device server framework relies on a basic model which is shown in figure 2. A device server has device groups which contain devices of the same interface. A device interface can contain properties, i.e. values which can be read or written, and operations, i.e. methods which may have input and output data. The depth of the hierarchy is limited because it is necessary to map the object hierarchy to a TINE interface.

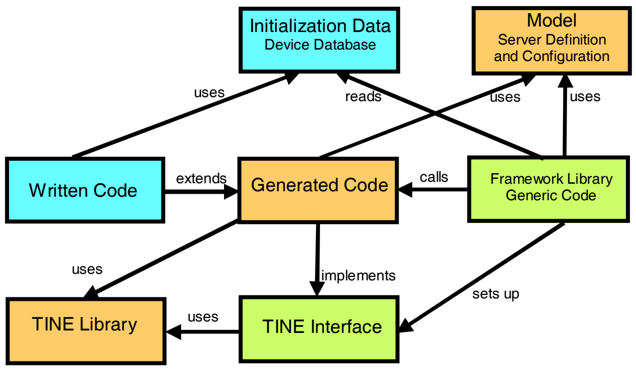

Figure 3: Constituents of a Device Server.

In most practical cases, this limitation is not a problem. Beyond the basic model there are also more complex concepts like group controllers, which are not discussed here.

The final device server program consists of various constituents as shown in figure 3. The core is the generic device server framework which is contained in the device server library. The generated code mainly implements the TINE interface, and the written code implements the device interface which itself is also generated. The device server program also needs the device server model at runtime.

In order to initialize the devices, initialization data must be provided. The device server framework assumes that initialization data can be accessed via JDBC. This makes it possible to support a wide range of file formats and database systems. In our case, most device servers use local CSV files with a commercial JDBC driver.

EXPERIENCES

The accelerators DESY II and Linac II have been commissioned with new control systems in 2008. The majority of the device servers has been built with the device server generator.

Due to its generalized structure, the device server framework cannot support all features of TINE, which has been considered in [3]. After a few extensions had been made to the original concept, this didn't turn out to be a problem. Up to now, the available functionality proved to be absolutely sufficient.

Our expectations were that the device server generator should be suitable for about 60-70% of device server tasks. For DESY II and Linac II, 32 of 39 Java device servers, about 80%, were produced with the generator by 7 of 8 programmers.

A survey among the programmers yielded that all who have used the device server generator clearly prefer it over manual server programming. However, exact comparison with previous projects is often hardly possible, since many colleagues have not previously written device servers with TINE and Java. Nevertheless the majority agrees that the device server generator significantly saves time, makes production of device servers easier, and makes device servers better maintainable.

The device server generator is suitable for a task if the subject matter can be described with the basic model. Most programmers regard the basic model as well-suited for device server tasks, but most also know use cases where they would prefer simple arrays of values instead of having sets of devices. For very simple tasks, the device server generator may sometimes appear a bit oversized. Nonetheless, this does not prevent it from being used.

Of course there are also issues which leave room for improvements. Currently, all data which belong to a device server model are held in a single XML file because of their strong interrelation. The model consists of interface definitions, server configuration and TINE configuration. This is normally no problem, but in some special cases more flexibility is needed. On the other hand, a separation of these data would also have an effect on the device server wizard GUI, which would become more complicated and harder to use.

The fact that code needs to be re-generated whenever properties have been changed can sometimes be annoying, but is unavoidable when code is generated. This problem can only be solved by a totally generic framework a la tomcat, which would require much more development effort. In this context, the idea of defining the device interfaces solely in the program text by using Java annotations is also worth having a look at.

CONCLUSION

The device server generator has proven to be a very productive tool in the DESY II and Linac II control systems. Its acceptance exceeds previous expectations. Although further improvements would certainly be possible, the current version fulfils our needs and will be used for device server production in the upcoming PETRA III control system. In principle, due to the dual concept of generated and generic code, the device server generator could even be ported to other platforms and languages. Which further developments will be made depends on the needs and plans of future projects.

REFERENCES

[1] P. Duval, V. Yarigin, "The Use of Wizards in Creating Control Applications", ICALEPCS Proceedings 2001, San Jose, California, THAP026.

[2] P. Duval, TINE (Thee-fold Integrated Network Environment); http://tine.desy.de.

[3] J. Wilgen, P. Duval, "A Device Server Generator for Control Systems", PCaPAC 2006, Newport News, USA; http://pubdb.desy.de/fulltext/getfulltext.php?lid=1377&fid=2776.