GENERATION OF EPC BASED SIMULATION MODELS

Prof. Dr. Christian Müller
Department of Management and Business Computing
Technical University of Applied Sciences Wildau
Bahnhofstrasse
D-15745 Wildau, Germany
christian.mueller@th-wildau.de

ABSTRACT

Bflow is an Eclipse plugin for modeling business processes in event driven process chain (EPC) notation. A code generator and its integration into Bflow to build a DesmoJ simulation model is presented. Possibilities to extend the generated models are also presented.

INTRODUCTION

The analysis and simulation of business processes is an important way of process optimization [Oberweis (1999), Böhnlein(2004), Böhnlein(2010)]. In the area of business process modeling, Business Process Modeling Notation (BPMN) [Object Management Group (2011), Freud(2010)] and Event-Driven Process Chain (EPC) [Rump(1999), Scheer(2000)] are the most popular standard notations. On the market, there are a lot of business process modeling tools that support the BPMN and EPC Notation. Some of these tools have an extension for business process simulation [ARIS Business Simulator(2012), IYOPRO(2012), Kloos(2010)]. For these approaches, the business process model is extended with some simulation related data, for instance, data about process times and decision probabilities. These simple models assume that the probability distribution for a decision is independent. This assumption is not realistic because a lot of decisions depend on some other system-properties and some earlier decisions.

Another way to build simulation models is based on universal simulation frameworks, simulation languages. or GUI simulation tools [Kennington(2012), OR/MS Today(2012)]. With these tools, it is possible to construct simulation models with arbitrary level of detail. On the other hand, a normal business person is not able to write and read these models.

In this paper, we present a approach, where the business process is formulated in EPC notation with some extensions about simulation relevant data. For EPC modeling Bflow [Kern et. al.(2010) Nüttgens(2011)], an Eclipse [Eclipse(2012)] plugin, is used. From this EPC model we generate a DesmoJ [DesmoJ(2012), Page(2005), Müller(2012)] simulation model. A DesmoJ model contains a set of Java classes that use the DesmoJ simulation framework. Based on the Java programing language, the generated models can be extended in all directions. The generation mechanism is designed in such a way that the modeling extensions are not lost by regeneration.

In this approach, the main part of model building can be done by business persons. Only for model extensions, executing and analyzing simulation, experts are necessary.

The base technology of this approach, Bflow and DesmoJ, are both open source projects. Bflow runs under Eclipse License [Eclipse License(2004)] and DesmoJ under Apache License [Apache License(2004)]. It is planed to publish the devoloped code generator and model validator also on an open source licence.

CONCEPT

All parts of this EPC Simulator are running in an Eclipse IDE. These are:

- Bflow as EPC modeling tool
- the model validator
- the code generator and
- the generated DesmoJ simulation model

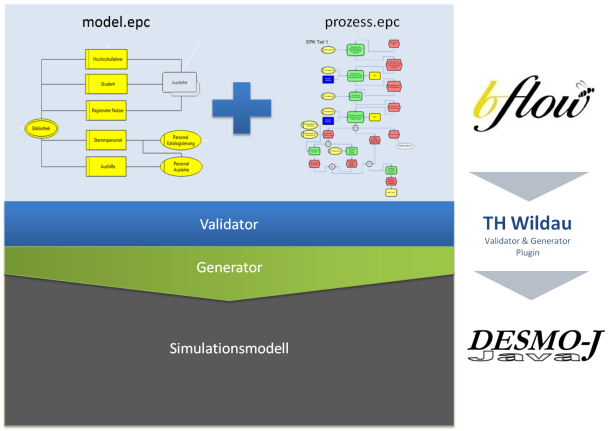

Figure 1:

Bflow is an Eclipse plugin that offers integration of external applications into Bflow. The model validator and code generator are developed as such kinds of

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

integrated programs. The code generator writes the generated code directly into the source code areas of its workspace. The simulation model can be started directly from the Eclipse IDE.

In some circumstances it makes sense to extend the generated code by some manual implemented extensions. The code generator supports some features that ensure that no extensions are lost by code regeneration. This mechanism is described in section “model extensions”.

The simulation process model stored in Bflow contains two type of documents. One is a model file and the others are process files. The common simulation attributes with information about the master and slave entities are stored in the model file. The process files contain classical process information and some extensions about distributions of processing times and decision probabilities. The simulation process model is described in the next section.

All parts of the project can be found in [Müller(2012)]. It embraces the specification of simulation process model, the model validator, the code generator and a toolbox (SimTools.jar) with some DesmoJ extensions to simplify the generated code.

SIMULATION PROCESS MODEL

In terms of simulation data structures, this simulation process model is an application of a master slave waiting queue. [Page(2005) p.289]

There are master and slave entities. The masters are walking as active entities through the processes described by EPC. For some tasks, they require some resources, represented by slaves. All free slaves are waiting in a master slave waiting queue for a new task. When a master requires a slave, it looks in the waiting queue for appropriate slaves.

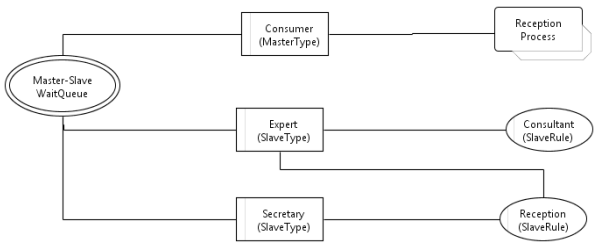

Figure 2: Model.epc

When no appropriate slaves are present, it waits for one. The master entities are grouped in different master types with special properties. For each master type, we have a master generator that produces master entities. The interarrival times are given by a random process. The distribution formulas of these processes can be different for different schedules of simulation time. Each master type has an unique associated EPC, that must be processed by the masters of this type. In a business process, the master types may represent customers with different visit frequency and different service requirements.

The slave entities are grouped in different slave types too. All slaves of a type have the same working time schedule and the same qualifications, represented as slave rules. When a master asks for a slave, it asks for slaves with special qualifications that are active in its working time schedule. When a slave has finished its job, it walks directly back to the waiting queue.

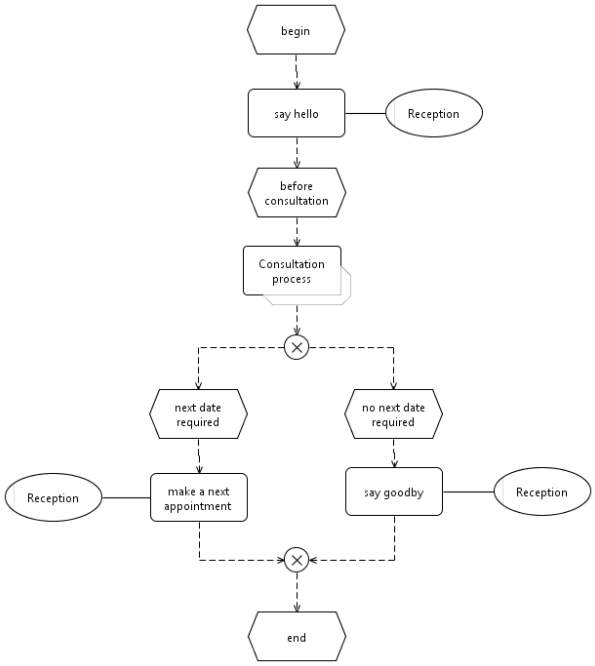

Figure 3: Reception.epc

In the given example (Figure 2), each customer starts in a reception process (Figure 3). In this process it needs a resource with qualification “Reception”. This can be satisfied from both slave types. Additionally, there is a process interface with a process link to a ”Consultation Process” (Figure 4). When the “Consultation Process” is finished, the process is continued according to the process interface in “Reception Process”. Each function has an individual random distribution of processing time.

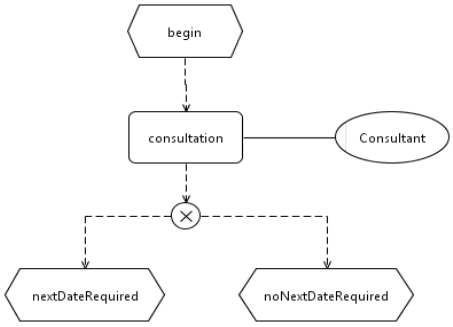

Figure 4: Consulting.epc

Both example processes (Figure 3 and 4) contain a XOR decision. This decisions can be driven by probabilities or by conditions. They are stored as additional data in the event-elements following the decision-element. Each master entity, that flows throw the process, stores the names of each passed event. One type of condition asks a master if it has passed a specified event. This technique is used in "Reception Process" to decide when a next date is required.

Besides XOR decisions, the EPC Simulator supports also OR decisions and AND splits. In these cases we get parallel working process lines. For each process line, a clone of the master entity is build. At the end of the parallel process lines, the clones are synchronized with its original master entity

SIMULATION CODE MODEL

The code generator transforms the simulation process model into a DesmoJ Simulation model. A DesmoJ simulation model is a Java Program which uses the DesmoJ Framework. The components of this framework can be divided in black- and white-box components. The black-box components are data collectors, statistical distributions, data structures (queues, etc) and visualization tools. They can used without modifications. On the other hand the white-box components are interfaces and abstract classes. These classes must be completed. DesmoJ supports an event orientated and a process orientated modeling style. For the process orientated style, processes, process generators and a simulation model must be implemented. The processes inherit from a class "SimProcess" and must implement a method "lifecycle". This method describe the behavior of its process entity. A process generator is a special process, that generates other processes. Between the generation of two process entities, it waits a interarrival time, that is normally given by a random distribution. The simulation model is an executable class which initializes the black-box components and starts some processes.

The code generator builds all necessary classes from a simulation process model.

The generated classes embrace

- for each epc in the model an Epc_ and Logic_ class. The Epc_ class inherit from the Logic_ class and contains all methods that can be extended by the user. The Logic_ class contains a life cycle method that describes the process logic of the EPC in a procedural formulation.
- a class for each master type. Each process of this type uses the life cycle of the associated EPC.
- a class for each slave type. The slaves wait in a master-slave-wait queue for a master that needs their service. When a service has finished, they walk back into the waiting queue. Each type of slave has qualifications described by one ore more slave rules.
- a class for each slave rule and
- an executable class for the simulation model. This class starts all slaves and lets them go into the waiting queue. Additionally it starts a process generator for each master type. The generator produces a sequence of master processes with inter arrival times described in the simulation model above.

MODEL EXTENSIONS

Not all situations that we are interested in for simulation can be described in a formalized way by an EPC. The generated classes offer a possibility for the controlled extension by the user. The generated classes contain markers for individual code extension. The idea is shown in an excerpt of the class Epc_Consultation.

```
@Generated(value = { "RememberCode", "yes" })
public class Epc_Consultation extends
                              Logic_Consultation{

   //@@_begin of id="_id_26_Function_consultation"
                     generated at 06.02.12 09:29
   /**
    * Type:      epc:Function
    * Name:      consultation
    * Resources: Consultant,
    * Id:        _id_26_Function_consultation
    * EpcFile:
       /Generator_2508/Artikel/Consultation.epc
    */
   protected void _id_26_Function_consultation(
                  EpcProcess proc,
                  List<EpcResource> resources){

       //@@_begin of individual code
       doSomeAdditionalThings(proc);
       //@@_end of individual code
   }
   //@@_end of id="_id_26_Function_consultation"
}
```

When a master entity arrives at a function element in the EPC, the simulator provides the necessary slaves needed to execute the function. In this excerpt the consultation function depends on the master entity (proc) and the provided slave entities (resources). This function describes what the master and slave entities are doing together and not how long they are working together. Usually, this function contains no code, but it can be extended by some individual code.

It is common practice that during the modeling process, simulation code is generated several times and also modified by the user. To account for this, the code generator reads the old version of each class and stores the extensions in memory before generating the new version. At code generation the generated code is extended by the stored extensions. This mechanism can be switched off by the "Generated" annotation of the class Epc_Consultation.

The same mechanism is offered for condition methods of events when they are marked in the "Simulation

Process Model" with a "condition : generated" attribute.

With this approach all properties of the Java language and the DesmoJ framework can be used. It is expected that the extensions contain only a few lines of code to manage additional information.

TRANSFORMATION OF EPC PROCESS LOGIC INTO A PROCEDURALE FORMULATION

To transform a simulation process model into a DesmoJ simulation model the generator builds logic classes from Epc process diagrams. Therefore, it transforms each Epc process diagram into a procedural formulation. The transformation algorithmus works in 2 phases. In the first (preprocessing) phase it is determined whether an Epc node is a

- split connector node (isSplit()) or a
- join connector node (isJoin()) or a
- end node (isEnd()) in the Epc process diagram.

The split connector nodes have normaly multiple and the other nodes only one direct successor. The set of direct successors of a split connector node (elementsOfsplit()) and the single succssor of the other nodes (successor()) are also determined (see Figure 5).

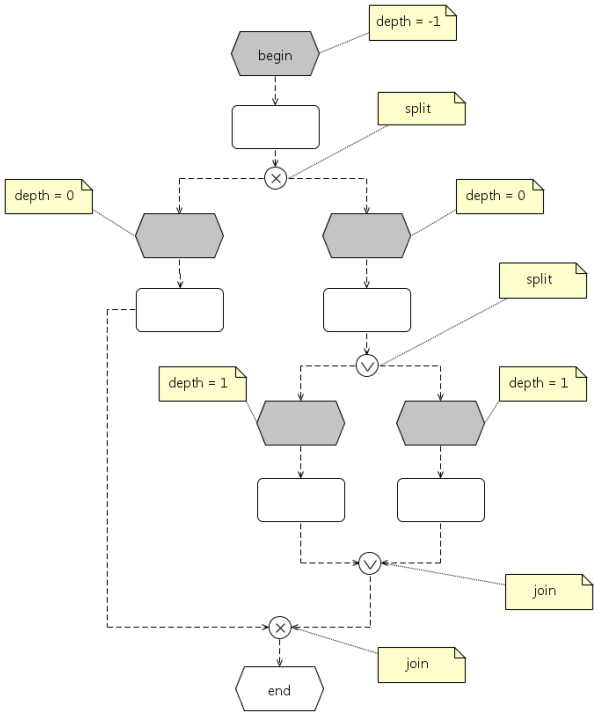

Figure 5:

In the second phase the procedural code of the lifecycle methods of logic classes is generated. Therefor, the begin node and the nodes in the elementsOfSplit() sets (gray nodes in figure 5) are traversed in a fifo order. All nodes of the same elementsOfSplit() set have the same depth and their code is generated together.

```
public void generatorPhase2(EpcNode begin){
   int depth = -1;
   stack.clear();
   stack.push(begin);
   begin.setDepth(depth);
   main: while(! stack.empty()){
      EpcNode a = stack.peek();
      // open if-clause of a
      a.generateCodeIfBegin();
      // run a code sequence
      while(! a.isEnd()){
         if(a.isSplit()){
            depth = stack.peek().getDepth()+1;
            for(EpcNode b: a.elementsOfSplit()){
               stack.push(b);
               b.setDepth(depth);
            }
            continue main;
         }else if(a.isJoin()){
            EpcNode c = stack.pop();
            int d = c.getDepth();
            // close if-clause of c
            c.generateCodeIfEnd();
            if( stack.peek().getDepth() == d)
               continue main;
         }else{
            a.generateCodeFunction();
         }
         a = a.successor();
      }
   }
}
```

This basic algorithm can be extended for processing epc's with loops and multiple end nodes.

ACKNOWLEDGMENTS

My special thanks go to Prof. Dr. Ralf Laue from FH Zwickau, Germany. He developed a Bflow extension for additional attributes. In 2011 we had an intensive discussion about Bflow and useful extensions for storing the simulation process model into Bflow. Only on the basis of his participation, the integration of the code generator into Bflow was possible.

This work was part of a student project at TH Wildau in 2011. My thanks go to B. Schneider, Chr. Thiemich and F. Kropp for writing a specification on the simulation process model, and to M. Till for developing a model validator based on this specification. The team of Chr. Krüger, M. Kannengiesser and N. Herm developed a method to transform the logic of EPC diagrams into the procedural formulation used by the code generator. This method works successfully and is a kernel functionality of the code generator.

REFERENCES

Apache License: Apache License, Version 2.0, 2004
http://www.apache.org/licenses/LICENSE-2.0.html

ARIS Business Simulator: 2012
http://www.softwareag.com/de/products/aris_platform/aris_design/business_simulator/capabilities/default.asp

Böhnlein, C: Simulation in der Betriebswirtschaft 2004 eds: Mertins, K; Rabe,M Experience from the Future – New Methods and Applications in Simulation for

Production and Logistics, Frauenhofer IRB p 1-22

Böhnlein, C: Simulationsunterstützte Spezifikation und Analyse von Geschäftsmodellen und Geschäftsprozessen 2010 eds: Claus, T; Herrmann, F; Simulation als betriebliche Entscheidungshilfe 14.ASIM Fachtagung p.83-104

DesmoJ: A Framework for Discrete-Event Modelling and Simulation 2012 http://desmoj.sourceforge.net/home.html

Eclipse: Eclipse Homepage 2012 http://www.eclipse.org/

Eclipse License: Eclipse Public License - Version 1.0, 2004 http://www.eclipse.org/legal/epl-v10.html

IYOPRO: IYOPRO Premium – Geschäftsmodelle simulieren und optimieren 2012 http://www.iyopro.de/pages/de/produktinformationen/produktuebersicht.html

Freud, J; Rücker, B: BPMN 2.0; 2010 Hanser

Heiko Kern, Stefan Kühne, Ralf Laue, Markus Nüttgens, Frank J Rump, Arian Storch: bflow* Toolbox - an Open-Source Business Process Modelling Tool 2010, Proc. of BPM Demonstration Track 2010, Business Process Management Conference 2010 (BPM'10), Hoboken, USA

Kloos, O; Nissen, V: Vom Prozess zur Simulation – Ein Transformationmodell-Ansatz 2010 eds: Claus, T; Herrmann, F; Simulation als betriebliche Entscheidungshilfe 14.ASIM Fachtagung p.105-119

Kennington, A: Simulation Software Development Frameworks 2012 , http://www.topology.org/soft/sim.html

Müller, C: Ein Ansatz zur Visualisierung von Desmo-J Simulationen 2011 http://www.th-wildau.de/cmueller/Desmo-J/Visualization2d/Visualisierung_DesmoJ_Simulationen.pdf

Müller, C: EPC Simulation Project Homepage 2012 http://www.th-wildau.de/cmueller/SimulationERP/

Nüttgens, M: Bflow Toolbox 2011 http://www.wiso.uni-hamburg.de/professuren/winfo-prof-nuettgens/forschung/projekte/bflow-toolbox/ and http://sourceforge.net/projects/bflowtoolbox/

Oberweis, A; Lenz, K; Gentner, C: Simulation betrieblicher Abläufe 1999 eds: wisu das wirtschaftsstudium 28(1999)02 p 216-223, 245

Object Management Group: Business Process Modeling Notation (BPMN) Version 2.0; 2011; http://www.omg.org/spec/BPMN/2.0/

OR/MS Today: Simulation Software Survey 2012, http://lionhrtpub.com/orms/surveys/Simulation/Simulation.html

Page, B et. al.: The Java Simulation Handbook, Shaker 2005

Rump FJ: Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs. 1999 Teubner Verlag.

Scheer, AW: ARIS Business Process Modelling. 2000 Springer Verlag.

AUTHORS BIOGRAPHIES

CHRISTIAN MÜLLER has studied mathematics at Free University Berlin. He obtained his PhD in 1989 about network flows with side constraints. From 1990 until 1992 he worked for Schering AG and from 1992 until 1994 for Berlin Public Transport (BVG) in the area of timetable and service schedule optimization. In 1994 he got his professorship for IT Services at Technical University of Applied Sciences Wildau, Germany. His research topics are conception of information systems plus mathematical optimization and simulation of business processes.

His email address is: christian.mueller@th-wildau.de and his web page is http://www.th-wildau.de/cmueller/ .