An open source software approach to combine simulation and optimization of business processes

Mike Steglich and Christian Müller
Technical University of Applied Sciences Wildau
Bahnhofstraße, D-15745 Wildau, Germany
E-mail: mike.steglich@th-wildau.de, christian.mueller@th-wildau.de

KEYWORDS
Simulation, Optimization, Business Process, Optimization web service, Mathematical Programming Language

ABSTRACT

Business processes of modern companies are characterized by a huge complexity which is caused for example by quickly changing markets, short product life cycles or dynamic interactions between particular subsystems of a company. Business process management is intended to implement efficient and customer orientated processes whereby the simulation of business processes can be used to evaluate the quality of processes and to identify areas of improvements. Since real business processes usually contain decision processes which can be solved by optimization systems, it makes sense to combine the simulation and the optimization of business processes. (März et.al. 2010, p 3ff.)
As an example of a reasonable combined simulation and optimization of business processes, the navigation in a road network is discussed in this paper. Consider vehicles seeking the fastest route from a starting node to a target node using a navigation system. The amount of time spent driving on an arc is influenced by the distance and the amount of the vehicles on this arc and is continuously changing. The structure of the road network and the traffic within the network is described in a simulation model while the fastest path decisions of each vehicle are made by using an optimization system. There is obviously a relationship between the individual decisions made for each of the vehicles and the state of the entire network.
The aim of this paper is to describe how a combined simulation and optimization of business processes can be created through using EPC-Simulator (Müller 2012) as a simulation system and CMPL (Steglich and Schleiff 2010) as an optimization system where the network traffic simulation is used exemplarily.

CREATING SIMULATIONS USING EPC-SIMULATOR

The EPC Simulator is a plugin of the EPC modelling toolbox bflow* (Kern et al. 2010). As shown in Figure 1 the first step to create a simulation model is the specification of a model document and several process documents in bflow*. The model document contains information about the model infrastructure (simulation time, available resources, inter-arrival times of entities, etc.). The process documents contain the process descriptions in EPC notation. Based on these documents, the EPC-Simulator can generate a simulation model. This is a Java Application that uses the DESMO-J Framework, whereby DESMO-J provides the basic functionality of a simulation. (Page and Kreutzer 2005)

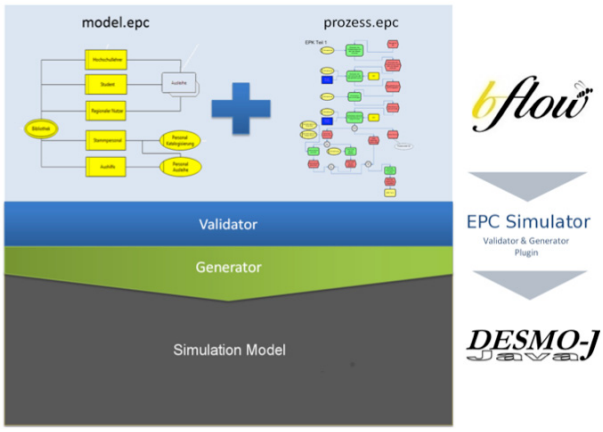

Figure 1: EPC-Simulator and its environment

The Java source code generated by EPC-Simulator contains marked areas where the source code can be extended manually by individual functionalities in functions or decision rules. In this way it is possible to integrate CMPL easily by a code extension through using its Java API.

IMPLEMENTING A ROAD NETWORK SIMULATION IN EPC-SIMULATOR

As described before, to create the road network simulation it is necessary to specify model and process documents. The model document in Figure 2 describes a series of vehicles with the specific inter-arrival times. Each vehicle has to process the vehicle process document. On the basis of this document the EPC-Simulator generates several Java classes on the basis of this model document that can be extended by individual Java code.
A class is `Information_Network_Data` (shown in Listing 1) for that a `NetworkData` object is created automatically by EPC-Simulator. This class is used to specify the road network that can be described as an undirected network $G=(V,A)$ where V is a set of nodes and A is a set of arcs joining pairs of nodes.

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

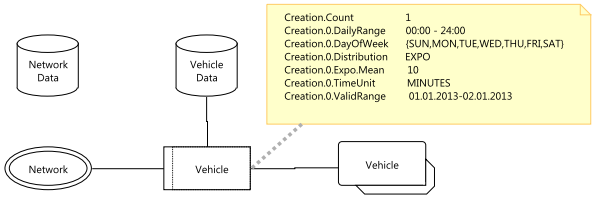

Figure 2: Model document

Considering the simple example in Figure 3, the first individual code after the code marker in line 03 is the definition of the set A of the arcs.

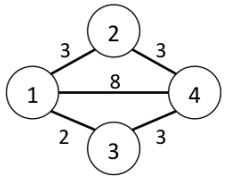

Figure 3: Example of a road network

The time a vehicle needs to drive from a node i to a node j is a function which consists of a fixed term ft_{ij}, which is proportional to the distance between the nodes i and j, and a variable term which is defined by the product of a time factor vt_{ij} and the amount of vehicles q_{ij} on the arc $i \rightarrow j$.

$$c_{ij} = ft_{ij} + vt_{ij} \cdot q_{ij} \; ; \forall (i,j) \in A \qquad (1)$$

Assuming that each arc is only used by one vehicle and the variable time factor is equal to 1, the lines 05 and 06 of Listing 1 describe the fixed and variable factors that are necessary to calculate the needed driving time in line 14. Additional data are the amount of vehicles of the arcs (line 08), the definition of the standard deviation in line 17 and a vector `meanDist` in line 07, which is assigned the current distances of all arcs calculated for each arc by method `getRandomDist`.

Another class is `Information_Vehicle_Data` shown in Listing 2 which is used to generate an individual object for each vehicle. This class is intended to specify vehicle specific data (e.g. the current location and the target node). Because for each vehicle a number of shortest path problems are to be solved it is also necessary to provide a vector of Boolean elements to store the solution of an optimization run (line 05). Additionally, a method for solving the shortest path problem is specified which contains the CMPL language bindings (line 13).

The vehicle process document describes the navigation process of the vehicle. As shown in Figure 4 a vehicle starts at its start position and drives to the next node on the determined shortest path.

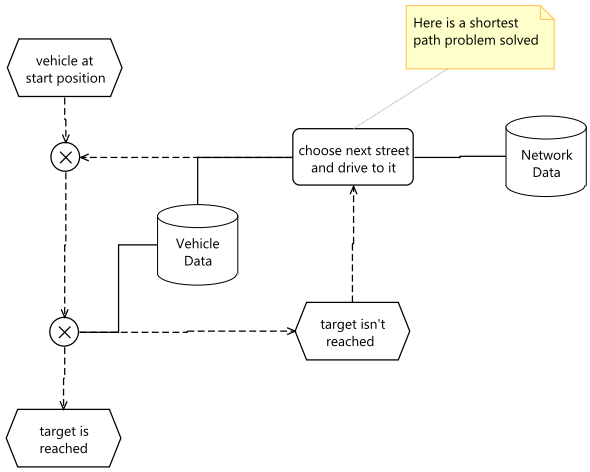

Figure 4: Process document

If the vehicle does not reach the target node a new shortest path depending on the current traffic situation in the network is determined by solving a shortest path problem. Based on the solution the next arc is chosen.

```
01 public class Information_Network_Data {
02
03     //@@_begin of declaration individual code
04     int[][] routes = {{1,2},{1,3},..,{3,4},{4,2},{4,3}};
05     double[] timeFix = { 2.0, 1.0,..., 2.0, 2.0, 2.0 };
06     double[] timeVar = { 1.0, 1.0,..., 1.0, 1.0, 1.0 };
07     double[] meanDist = null;
08     int[] vehicles = null;
09     int nrNodes = 4;
10     …
11     ContDistNormal normDist = null;
12     //@@_end of declaration individual code
13     …
14     //@@_begin of additional individual code
15     public double getRandomDist(int arcIndex){
16         double mean = timeFix[arcIndex]+  timeVar[arcIndex]*vehicles[arcIndex];
17         double stdDev = alpha * mean;
18         return Math.max(0.0, normDist.sample()*stdDev + mean);
19     }
20     …
21     //@@_end of additional individual code
   }
```

Listing 1: Information_Network_Data

```
01 public class Information_Vehicle_Data {
02
03     //@@_begin of declaration individual code
04     int locationNode, targetNode;
05     boolean[] shortestPath = null;
06     …
07     //@@_end of declaration individual code
08     …
09     //@@_begin of additional individual code
10     public boolean solveShortestPath(){
11         boolean ok;
12         …
13         return ok;
14     }
15     //@@_end of additional individual code
16 }
```

Listing 2: Information_Vehicle_Data

CMPL AND CMPLSERVER

Introduction

CMPL (<Coliop|Coin> Mathematical Programming Language) is a mathematical programming language and a system for mathematical programming and optimization of linear optimization problems. CMPL executes CBC, GLPK, Gurobi, SCIP and CPLEX directly to solve the generated model instance. Since it is also possible to transform the mathematical problem into MPS, Free-MPS or OSiL files, alternative solvers can be used. CMPL is an open source project licensed under GPLv3. It is written in C++ and is available for most of the relevant operating systems (Windows, OS X and Linux). CMPL is a COIN-OR project initiated by the Technical University of Applied Sciences Wildau and the Institute for Operations Research and Business Management at the Martin Luther University Halle-Wittenberg. (Steglich and Schleiff 2010)

Formulation of the shortest path problem in CMPL

Consider an undirected network $G=(V,A)$ where V is a set of nodes and A is a set of arcs joining pairs of nodes. The decision is to find the shortest path from a starting node s to a target node t. This problem can be formulated as an LP as follows (Hillier and Liebermann 2010, p. 383f.):

$$\sum_{(i,j)\in A} c_{ij} \cdot x_{ij} \rightarrow \min! \qquad (2)$$

s.t.

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = \begin{cases} 1 & , if\ i = s \\ -1 & , if\ i = t \\ 0 & , otherwise \end{cases} ; \forall i \in V \qquad (3)$$

$$x_{ij} \geq 0 \ ; \forall (i,j) \in A \qquad (4)$$

The decision variables are $x_{ij}\ \forall (i,j) \in A$ with x_{ij}=1 if the arc $i \rightarrow j$ is used. The parameters $c_{ij}\ , \forall (i,j) \in A$ usually define the distance between the nodes i and j, but as formulated in Expression (1) in this work these parameters are interpreted as the time a vehicle takes to drive from node i to node j.

To describe the formulation of the shortest path problem in CMPL the simple example shown in Figure 2 is used. Because parameters, variables, objectives and constraints are to be defined for each optimization model, a CMPL model usually consists of a parameter section, an objective section, a variable section and a constraint section. The set of the arcs A is specified in the lines 02-05 in Listing 3 in the form of a 2-tuple set. The set V of the nodes is a set of single indexing integers (line 07). The parameters s and t are scalar parameters (lines 08 and 09) and c is an array that is defined over the set A (line 11).

```
01 parameters:
02   A := set(  [1,2],[1,3],[1,4],
03              [2,1],[2,4],
04              [3,1],[3,4],
05              [4,2],[4,3]);
06
07   V := 1..4;
08   s := 1;
09   t := 4;
10
11   c[A] := (3,2,8,3,3,2,3,3,3);
12
13   { i in V: {    i=s : b[i]:=1; |
14                  i=t : b[i]:=-1; |
15                  default: b[i]:=0;}  }
16
17 variables:
18   x[A] :real[0..];
19
20 objectives:
21   sum{[i,j] in A: c[i,j]*x[i,j]}->min;
22
23 constraints:
24   node {i in V:
25         sum{ j in (A *> [i,*]):x[i,j]}-
26         sum{ j in (A *> [*,i]):x[j,i]}
27         = b[i];}
```

Listing 3: The shortest path problem in CMPL

The vector of the right hand sides b has been defined in the lines 13-15 by using a loop over the set V and a switch clause depending on the value of a current i and the values of s and t as in Expression (3).
Line 18 contains the definition of the decision vector x over the set A as non-negative continuous variables.
The sum over all valid arcs of the products of the c_{ij} and the x_{ij} is minimized. The formulation of the objective function (2) is done in line 21.
The flow constraints (4) can be formulated as in lines 24-27. The first step is to define a loop over all nodes in V (line 24) where the loop body contains the formulation of the flow constraints. The sums in lines 25 and 26 are defined over a set pattern matching expression. The expression `A *> [i,*]` returns a set consisting of unique elements of A which match the pattern `[i,*]` in order of their first appearance. That means this expression yields all nodes j that can be reached from the node i.

CMPL API and CMPLServer

The CMPL API is intended to integrate CMPL in other software and is available for Python and Java since version 1.9.0.
The main idea of this API is to define sets and parameters within the user application, to start and control the solving process and to read the solution(s) into the application if the problem is feasible. All variables, objective functions and constraints are defined in CMPL. These functionalities can be used with a local CMPL installation or a CMPLServer, but this paper describes only the use of a CMPLServer.
The CMPL API contains three major classes: `CmplSet`, `CmplParameter` and `Cmpl`. The classes `CmplSet` and `CmplParameter` are intended to define sets and parameters that can be used with several `Cmpl` objects. With the `Cmpl` class it is possible to define a CMPL model, to commit sets and parameters to this model, to start and control the solving process and to read the CMPL and solver messages and to have access to the solution(s).
`Cmpl` also provides the functionality to communicate with a CMPLServer that is implemented as an XML-RPC-based web service. XML-RPC provides XML based procedures for Remote Procedure Calls (RPC), which are transmitted between a client and a server via HTTP. (Laurent et al. 2001, p. 1.) Such client-server architecture is reasonable for a combined simulation and optimization of business processes because the capacities of the client can be used for the simulation procedures while the optimization problems can be solved remotely on the CMPLServer.
CMPL provides three XML-based file formats for the communication between a CMPLServer and its clients. A `CmplInstance` file contains an optimization problem formulated in CMPL, the corresponding sets and parameters in `CmplData` file format and all CMPL and solver options that belong to the CMPL model. If the model is feasible and the solution process is finished a `CmplSolutions` file contains the solution(s) and the status of the invoked solver. If the model is not feasible then only the solver's status and the solver messages are given in the solution file. The `CmplMessages` file is intended to provide the CMPL status and (if existing) the CMPL messages.
As shown in figure 5 the first step to communicate with a CMPLServer is the `Cmpl.connect` method that returns (if connected) a `jobId`. After connecting, a problem can be solved synchronously or asynchronously.

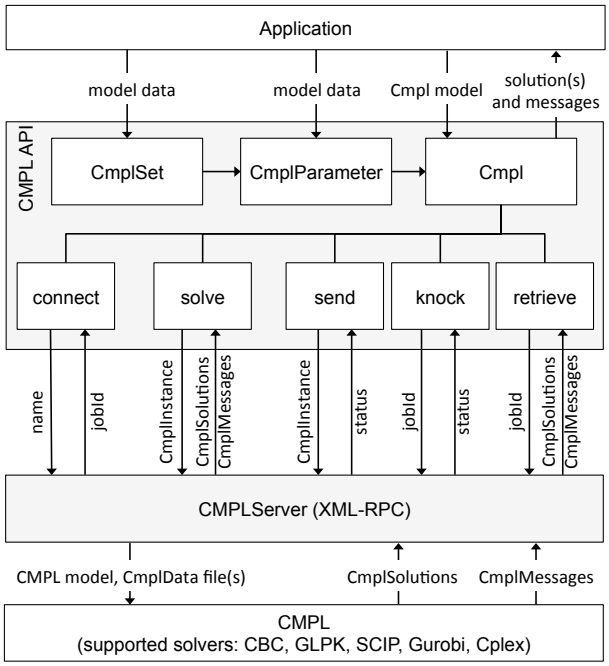

Figure 5: CMPL API and CMPLServer

The `Cmpl.solve` method sends a `CmplInstance` to the connected CMPLServer and waits for the returned `CmplSolutions` and the `CmplMessages`. After this synchronous process a user can access the solution(s) if the problem is feasible or if not one can analyse whether the CMPL formulations or the solver is the cause of the problem. To execute the solving process asynchronously the CMPL methods `Cmpl.send`, `Cmpl.knock` and `Cmpl.retrieve` have to be used. `Cmpl.send` sends a `CmplInstance` to the `CMPLServer` and starts the solving process remotely. `Cmpl.knock` asks for a CMPL model with a given `jobId` whether the solving process is finished or not. If the problem is finished the `CmplSolutions` and the `CmplMessages` can be read into the user application with `Cmpl.retrieve`.

INTEGRATING CMPL INTO THE ROAD NETWORK SIMULATION IN EPC-SIMULATOR

One of the advantages of EPC-Simulator is the opportunity for programmers to develop their own simulation models. Therefore it is possible to integrate optimization routines into simulation models through

using the CMPL Java API. The following steps have to be implemented for all of the simulation nodes.

To optimize the shortest path of each vehicle in the simulation nodes it is necessary to specify the network structure and the current traffic situation. The structure of the undirected network $G=(V,A)$ is given by the set of nodes V and the set A of arcs joining pairs of nodes. The current situation is given by the parameters $c_{ij}\ \forall(i,j)\in A$, which define the time a vehicle takes to drive from node i to node j. The current location of a vehicle describes the starting node s while t is the target node. These sets and parameters are the input of a CMPL model that is similar to the CMPL model in listing 3. But the difference is that this model does not contain static data. The sets and parameters have to be read into the model by using the `%data` entry (Listing 4 - line 1). This CMPL header entry is intended to read data from a `CmplData` file or from an application that uses the CMPL API.

The implementation of a CMPL model in Java is described in Listing 5 which is the completion of `solveShortestPath` in Listing 2. It starts with the definition of a `Cmpl` object (Listing 5 - line 09) with the filename of the model as an argument for the constructor.

```
01 %data : A set[2], V set , c[A], s, t
02
03 parameters:
04   { i in V: {    i=s : b[i]:=1; |
05                  i=t : b[i]:=-1; |
06                  default: b[i]:=0;}  }
07 …
```

Listing 4: sPath.cmpl

To define the sets A and V one has to create `CmplSet` objects (lines 10 and 12). The arguments of the constructor are the name of the set that has to be the same as specified in the CMPL model and the rank of the set if it is greater than one. The set A of the arcs is a set of 2-tuples and has therefore the rank 2 while V is a set of single indexing integers with the rank 1. Both sets are assigned data by using the method `CmplSet.values` where for the enumeration set `arcs` (A) a Java array is used as an argument (line 11)

```
01 import jCmpl;
02
03 public boolean solveShortestPath(){
04     boolean ok = false;
05     Cmpl m;
06     CmplSet arcs, nodes;
07     CmplParameter dist, sNode, tNode;
08     try{
09         m = new Cmpl("sPath.cmpl");
10         arcs = new CmplSet("A",2);
11         arcs.values( networkData.routes );
12         nodes = new CmplSet("V");
13         nodes.values(1,networkData.nrNodes);
14         dist = new CmplParameter("c", arcs);
15         dist.values(networkData.meanDist);
16         sNode = new CmplParameter("s");
17         sNode.values(this.locationNode);
18         tNode = new CmplParameter("t");
19         tNode.values(this.targetNode);
20
21         m.setSets(arcs, nodes);
22         m.setParameters(dist,sNode,tNode);
23         m.connect("http://194.95.45.70:8008");
24         m.solve();
25         if (m.solverStatus()== Cmpl.SOLVER_OK && m.nrOfSolutions()>0) {
26             ok = true;
27             shortestPathStatus = m.solution().status();
28             leftTime = new TimeSpan(m.solution().objValue(), TimeUnit.MINUTES);
29             shortestPath = new boolean[networkData.vehicles.length];
30             for(int i=0; i<m.nrOfVariables(); i++){
31                 shortestPath[i] = (m.solution().variables(i).activity() == 1);
32             }
33             } else {
34                 ok = false; shortestPath = null; shortestPathStatus = "";
35             }
36     } catch(CmplException e){
37         ok = false; shortestPath = null; shortestPathStatus = "";
38     }
39         return ok;
40 }
```

Listing 5: Use of the CMPL Java API

and for the set of ascending integers `nodes` (V) the first and the last element are given (line 13).
For the definition of a CMPL parameter a user has to create a `CmplParameter` object where the first argument of the constructor is the name of the parameter. If the parameter is an array it is also necessary to specify the set or sets trough which the parameter array is defined. Therefore it is necessary to commit the `CmplSet arcs` (beside the name `"c"`) to create the `CmplParameter` array `c` (line 14) while for the definition of the `CmplParameter sNode` and `tNode` only the name of the parameters have been specified (lines 16 and 18).
`CmplSet` objects and `CmplParameter` objects can be used in several CMPL models and have to be committed to a `Cmpl` model by `Cmpl.setSets` and `Cmpl.setParameters` (lines 21 and 22).
Before the solving process is started in line 24 a CMPLServer located at `http://194.95.45.70:8008` is connected (line 23). If `Cmpl.connect` is not executed, a locally installed CMPL is used automatically. After solving the model the status of CMPL and the invoked solver can be analysed through `Cmpl.solverStatus` (line 25) and `Cmpl.cmplStatus`.
If the problem is feasible and a solution is found it is possible to read the names, the types, the activities, the lower and upper bounds and the marginal values of the variables and the constraints into the Java application.
For the combined simulation and optimization of the shortest path problem now it is possible to get in each node s of the simulation the traffic for the next simulation step by analysing the activities of the variables x_{sj}, $\forall (s,j) \in A$ line 31.

SUMMARY

The aim of this paper was to develop an approach to combine a simulation and an optimization of business processes exemplarily described on the basis of the network traffic simulation problem.
It was shown how a network traffic simulation can be created with particular model and process documents in EPC notation, with which the EPC-Simulator (a bflow* plugin) can generate Java source code that uses the DESMO-J Framework. This Java source code contains marked areas for individual extensions which are used in this work to specify the specific data of the road network and the vehicles.
These marked areas for individual extensions were also used to describe how CMPL can be integrated by using the CMPL Java API. This API is indented to define sets and parameters within a Java application, to commit it to a CMPL object, to start and control the solving process and to analyse the solution(s) in Java. The CMPL Java API can be used with a local CMPL installation or a CMPLServer which is an XML-RPC-based web service for CMPL.
Client-server architecture for a combined simulation and optimization business process is reasonable because the capacities of the client can be used for the simulation procedures while the optimization problems can be solved remotely on the CMPLServer.

REFERENCES

Hillier F. S./Lieberman, G. J. 2010. “Introduction to Operations Research.” 9th ed., McGraw-Hill Higher Education.

März, L.; Krug, W.; Rose, O. and Weigert, G. 2010. “Simulation und Optimierung in Produktion und Logistik”. Springer, Berlin Heidelberg.

Kern, H.; Kühne, S.; Laue, R.; Nüttgens, M.; Rump, F.J. and Storch, A. 2010. “bflow* Toolbox - an Open-Source Business Process Modelling Tool”, In: *Proc. of BPM Demonstration Track 2010, Business Process Management Conference 2010* (BPM’10), Hoboken, USA.

Müller, Chr. 2012. “Generation of EPC Based Simulation Models ” In: *Proceedings 26th European Conference on Modelling and Simulation 2012*, Koblenz.

Page, B.; Kreutzer, W. 2005. “The Java Simulation Handbook - Simulating discrete Event Systems with UML and Java.” Informatics Series, Shaker Publ., Aachen.

St. Laurent, S.; Johnston, J. and Dumbill, E. 2001 “Programming Web Services with XML-RPC.” 1st ed., O'Reilly.

Steglich, M. and Schleiff, Th. 2010. “CMPL: Coliop Mathematical Programming Language.” In: *Wildauer Schriftenreihe - Entscheidungsunterstützung und Operations Research*, Beitrag 1, Technische Hochschule Wildau [FH].

AUTHOR BIOGRAPHIES

Mike Steglich is a Professor of Business Administration, Quantitative Methods and Management Accounting in the Faculty of Business, Administration and Law of the Technical University of Applied Sciences Wildau in Germany. He is also one of the authors and distributors of the open source project CMPL (<Coliop|Coin> Mathematical Programming Language).

Christian Müller studied mathematics at Free University Berlin. He obtained his PhD in 1989 on 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 gained his professorship for IT Services at the Technical University of Applied Sciences Wildau, Germany. His research topics are conception of information systems plus mathematical optimization and simulation of business processes.