Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

EMULATION WITH DSOL

Peter H.M. Jacobs
Alexander Verbraeck

Department Technology, Policy and Management
Delft University of Technology
Delft, THE NETHERLANDS

William Rengelink

TBA Nederland
Delft, THE NETHERLANDS

ABSTRACT

Manufacturing control systems are extremely hard to design and test. Testing Programmable Logic Controller (PLC) software in an on-line manufacturing setting can be costly, dangerous, and inefficient. The availability of a seamless transition between the real manufacturing process and a simulated manufacturing process on the one hand, and a real PLC and a soft PLC on the other hand might help to solve these problems. Using the Java-based object oriented simulation library DSOL (Distributed Simulation Object Library), a case study was conducted for a concrete floor manufacturer to see whether these problems could be overcome. The full simulation and hardware-in-the-loop tests with DSOL, with the Modbus middleware protocol, and with real and soft PLCs went fine, and showed the added value of the distributed, service-oriented paradigm on which DSOL is based.

1 INTRODUCTION

A case study conducted for TBA Nederland is presented in this paper. TBA is a simulation consulting firm active in the domain of logistic business process (re)engineering using simulation and *emulation* (TBA Nederland 2000). TBA's areas of expertise includes (re)design of manufacturing processes, airport capacity systems, i.e. luggage and cargo systems and container terminals.

The case presented in this paper concerns an emulation study conducted for Dycore b.v., a concrete floor manufacturer. The value of this case mainly results from the conditions under which it was conducted: the case study was done as a competition between a team of developers from TBA and a team from Delft University of Technology. TBA's team, which consisted of two senior engineers, used their de-facto simulation environment, eM-Plant, while the TU Delft team, which consisted of the TU Delft authors of this paper, used DSOL. DSOL is an open source, Java based, multi-formalism simulation suite (Jacobs, Lang, and Verbraeck 2002). The idea behind this, real life, real time competition, was for TBA to be able to assess to what extent DSOL is a serious alternative to eM-Plant and whether they might wish to use it in the future for emulation projects.

We introduce the case in section 1.1, then we discuss the importance of *emulation* in the design of control systems in section 1.2, and illustrate the advantages of using DSOL for emulation projects. We conclude the introduction of this paper with a requirement analysis.

A conceptual model of the case is presented in section 2. The emulation bottleneck, i.e. the communication between the realtime system and the simulation model, forms the topic of section 2.1. The specification of the model in DSOL and a comparison with the specification in em-Plant is presented in section 3. After presenting the experiments in section 3.4, we conclude this paper with conclusions on DSOL's applicability in the domain of emulation. Note, this paper has been read and its content has been endorsed by TBA Nederland and Dycore.

1.1 60m^3 of Concrete Floors on an Automated Guided Vehicle

Dycore is a concrete floor manufacturer. As such Dycore produces annually more than 3,000,000m^3 of concrete floors for the Dutch and global markets. Dycore employs more than 500 employees in their combined facilities. All their production plants are KOMO certified and comply to the NEN ISO 9001 standards (KOMO is a hall-mark of the SBK foundation (<http://www.komo.nl>) and NEN is the Dutch institute for normalization (<http://www.nen.nl>)).

The case presented in this paper deals with the production of *sheet piling floors* at the manufacturing plant in Breda, the Netherlands (see Figure 1).

The production of sheet piling floors is fully automated. All machines, sensors, conveyors, cranes and vehicles are controlled by a *programmable logic controller*, or PLC. The case deals with a subsystem of this production plant, called the *bewapeningsomloop* which is best translated as the *reinforcement gallery*. Dycore commissioned TBA to

Figure 1: Sheet Piling Floors

test a newly designed and developed programmable logic controller. The aim of this commissioning was to debug the engineered system in the lab - not on the factory floor, for a number of reasons. The first and most important one is that this plant works with a 2 shift production that should not be disturbed. A second reason, which results from the first reason, is that testing can only be done in a limited way during night and weekend hours, which are very costly. Finally the operators should be trained in an environment that does not disturb the operations.

1.2 The Importance of Emulation in the Design of Realtime Control

Testing the behavior of a PLC, which controls one or more devices that are part of a logistic system, is usually done by connecting the PLC to stand-alone versions of each individual device, called mock-ups (Schludermann, Kirchmair, and Vorderwinkler 2000, Schiess 2001). This approach to device testing is expensive and the test conditions are hard to reproduce. Above all these test are incomplete since the interaction of a device with other devices is ignored and the system as such cannot be tested as a whole (Schludermann, Kirchmair, and Vorderwinkler 2000, Schiess 2001).

Emulation is a *hardware-in-the-loop* approach that is designed to solve this incompleteness. Emulation implies that all *inputs* and *outputs* of a controller are connected to *simulated devices*. This enables better reproduction of test conditions and allows the tester to reproduce the interaction of various parts of the complete system (Schludermann, Kirchmair, and Vorderwinkler 2000). (Whorter, Baker, and Malan 1997) state that using the same model for system development, system testing using emulation and staff training, can reduce costs and plant set-up times.

1.3 Emulation with DSOL

We expected DSOL to perform better than traditional simulation environments for a number of reasons that are given below. One, DSOL clearly distinguishes a model from a simulator. This distinction supports the usage of one model for system development, system testing and training purposes, as the same model can be deployed for different purposes and with different simulators, e.g. a wall clock simulator for real operations and training, a stepwise simulator for testing, and a paced simulator for demonstrations. Two, the service oriented, open architecture underlying DSOL should make the deployment of an emulation model in a distributed, networked environment more straightforward. Three, the multi threaded, scalable characteristics of the Java programming language should make DSOL more effective in the performance-defiant domain of emulation. Four, the support for CAD drawings in combination with the inclusion of Java's 3D modeling should give DSOL advantage with respect to the straightforwardness of infrastructure modeling.

The value of this case for the validation of DSOL was is found in the opportunity provided support these claims in a real time, real life case. A specification of the requirements for the emulation model is given below.

1.4 Requirement Analysis

To help us understand the requirements for an emulation model of the reinforcement gallery, we will first briefly introduce the internal structure of a PLC. The major components of a PLC are its *CPU*, its *memory*, its *power supply*, its *inputs* and its *outputs*: where inputs provide a PLC with the ability to read signals from different input devices, e.g. sensors and buttons, and outputs provide a way for a PLC to control output devices, e.g. motors and cranes. Data exchange between a PLC and these physical devices is based on special *industrial protocols*. This leads to a first requirement for our emulation model. The most important two requirements from Dycore and TBA were the following:

Requirement 1. *The emulation model must ensure that no modification has to be made in the PLC or the PLC program for testing.*

Requirement 2. *The emulation model should support the industrial data exchange protocol used by Dycore's PLC.* Memory in a PLC can be distinguished into *system memory*

and *user memory*. System memory is used by a PLC for its internal process control system. The user memory contains a user program translated from a *ladder diagram* to a binary form. User memory is divided into blocks having special functions. Some parts of a memory are used for storing input and output status. The real status of an input is stored either as "*1*" or as "*0*" in a specific memory bit. The combination of 16 bits is called a *word*, and each input or output has one or more corresponding bits in memory. An example of two lamps is presented in Figure 2 to illustrate how memory is used to control specific devices. Other parts of memory are used to store variable contents for variables used in user program. For example, timer value, or counter value would be stored in this part of the memory (Matic 2001).

A PLC reads its inputs and sends its outputs on a regular basis. This time interval is called the *period* of the PLC. Since this period defines the accuracy of the control system and as such of the controlled devices, a third requirement is that

Requirement 3. *The emulation model should meet the realtime period of the PLC.*

To ensure that an emulation model meats this real time constraint, emulation requires an underlying real time operating system (Schludermann, Kirchmair, and Vorderwinkler 2000).

Requirement 4. *In situations where an emulation model is nevertheless deployed on standard, i.e. non real time, operating systems, e.g. Linux or Microsoft Windows, the emulation environment should report potential unacceptable backlog.*

Where both requirements focus on the usefulness of the emulation model, TBA presented a further set of requirements with respect to the usability of the testing environment.

Requirement 5. *The emulation model should animate all devices in realtime on top of the CAD layout which was well known to the controllers of the physical system.*

Requirement 6. *All simulated devices should be* controllable *at runtime through a graphical user interface provided by the emulation model. This implies clicking on a simulated device and stopping or resuming its operation, creating failures, pressing buttons, etc.*

2 CONCEPTUALIZATION

According to (Banks 1998) conceptualization implies the abstraction of a real system using a conceptual model, i.e. a series of mathematical and logical relations between objects. This conceptual model underlies both the DSOL specification presented in section 3 and TBA's eM-Plant specification. In this section a clear distinction is made between the *control system* and the *controlled system*. The control system, i.e. the PLC controls the controlled system, i.e the simulated devices in the emulation model. We start in section 2.1 with a conceptual model of the control system, i.e. the PLC. We continue in section 2.2 with the conceptual model of the controlled system, i.e. the DSOL emulation model. In section 2.3 we discuss the communication between the emulation environment and the real time PLC, and we conclude this section with conceptual models of the overall architecture.

2.1 The Conceptual Model of the Control System

To understand how the control system functions, it is necessary to reiterate what we stated in the introduction of this paper, i.e. that the inner works of a programmable logic control are based on changing bit values of internal memory addresses. This is illustrated in Figure 3. The memory of a PLC is divided in a number of *registers*, i.e. data elements containing 16 bits. A clear, but subjective, distinction is made between that part of the memory in which write operations take place, and that part which is read-only for external devices to support the consistency of the PLC. The registers that are read by external devices are called *Registers*, while those registers used for writing are called *InputRegisters*.

Some inputs and outputs use more than one bit to read or write a specific value. For example, consider an automated guided vehicle the speed of which is presented as a double value, i.e. a `double` value in Java. In this specific case, 32-bits are needed to store this speed value, and as such two *InputRegisters* are dedicated to the speed of a specific vehicle.

2.2 The Conceptual Model of the Controlled System

The conceptual model of the DSOL emulation model, i.e. the controlled system is presented, in this section. We introduce three types of devices (see Figure 4) to illustrate the emulation model.

- *Input devices* which *only send* data to the programmable logic controller. Examples of this type include a sensor, an emergency button and a GPS-device.
- *Output devices* which *only receive* data from the programmable logic control. Examples of this type include a lamp and a siren.
- *Combined devices* which *both send and receive* data to the programmable logic controller. Examples include a motor, the operation of which is controlled (or received), and which might send emergency events.

A number of remarks must be made with respect to the conceptual diagram presented in Figure 4. One, the terminology is rather confusing. Although the name `Interface`

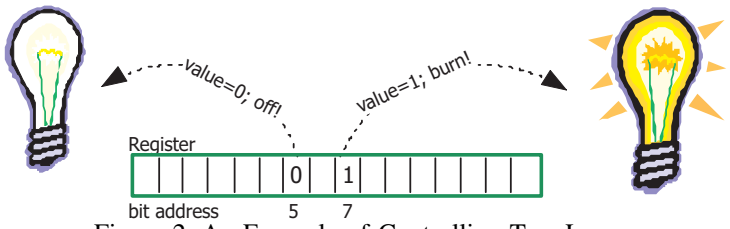

Figure 2: An Example of Controlling Two Lamps

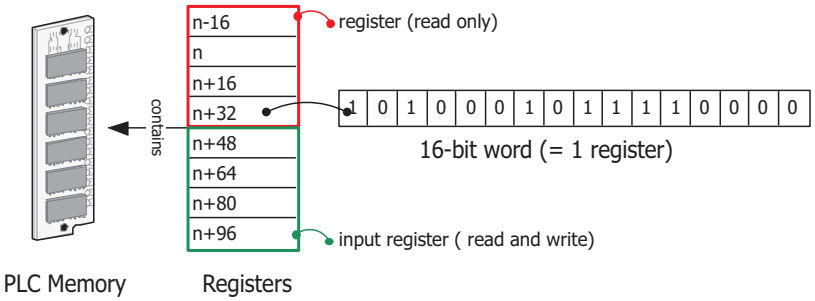

Figure 3: The Memory of a Programmable Logic Controller

might suggest that a Java interface is meant, it represents an abstract class specifying the interface between a simulated device and a PLC. Two, a `0..N` associative relation is presented between an interface and a specific device. This illustrates the fact that several simulated devices may share the same interface. Three, we see that combined devices are associated with both input and output interface. Four, the interface in Figure 4 is associated with the PLC. Having introduced conceptual models of the control system and the controlled system, a more detailed conceptual model of the communication between the control system and the controlled system is presented in the following section.

2.3 The Model-PLC Interface

The emulation model must conform to the PLCs industrial *Modbus* communication protocol to ensure that no modification has to be made on the PLC. Modbus is an application layer messaging protocol, positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of buses or networks (Modicon 1996). Modbus can furthermore be accessed over a reserved system port 502 on the TCP/IP stack.

Modbus offers services specified by function codes, which are elements of request/reply protocol data units (Modicon 1996). A *Master-Slave* concept is applied in the field of programmable logic controllers to govern the lower level communication behavior on a network using a shared signal cable (Modicon 1996).

A number of basic public functions have been developed in the Modbus protocol for exchanging data that is typical for the field of automation. These functions are presented in Table 1.

Any TCP/IP based implementation of the protocol should extend the protocol data units with an IP specific header. As we will show in the next section, a detailed, reliable and above all high performance implementation of this protocol is crucial for using emulation for PLC testing purposes successfully.

The conceptual architecture for the emulation model is presented in Figure 5. On the left, the programmable logic controller (PLC) is attached to the TCP/IP network. On the server side of the network a Java implementation of the Modbus protocol ensures adequate communication with the PLC. To minimize the amount of communication over the network, this communication library is connected to a *shadow memory* of the PLC which is part of the emulation model. The shadow memory limits communication in the following ways.

- Although all registers are read periodically from the PLC, the shadow memory will only fire value change events whenever *input values*, i.e. values to be written into the memory of the PLC, are actually changed.
- The shadow memory only sends changed input registers values to the PLC.

Emulated components either write to this shadow memory, and thus to the PLC, or are asynchronously subscribed to changes on the memory addresses which control their behavior. A sensor is an example of such writing, i.e. of an input, component, while a crane is an example of a reading, i.e. of an output, component. The behavior of, and interaction between, components in the emulation model is time dependent and this requires a simulator.

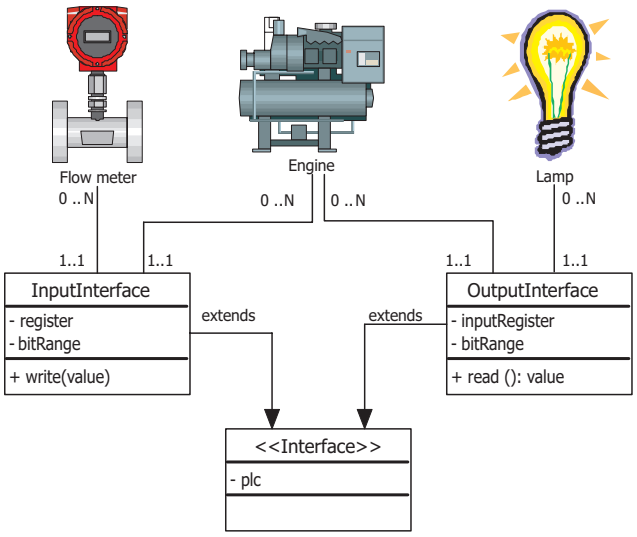

Figure 4: A Conceptual Model of Devices

Table 1: Basic Modbus Functions

Name	Type	Access	Visual representation
discrete input	single bit	read-only	
discrete output, i.e. coil	single bit	read-write	
input register	16-bit word	read-only	
output register	16-bit word	read-write	

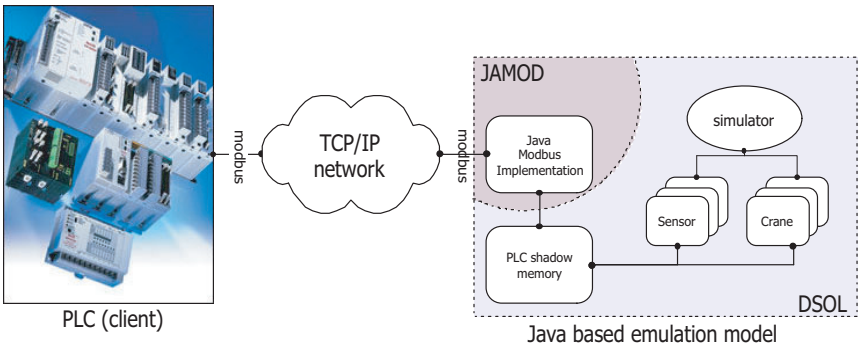

Figure 5: The Emulation Architecture

3 SPECIFICATION

The next activity in the design of the emulation model was to specify the model in DSOL. We begin in section 3.1 with the specification of the Modbus-DSOL communication. We continue with the specification of the simulated devices in DSOL in section 3.2.

3.1 Modbus Communication with DSOL

As argued in the introduction of this paper, great emphasis is placed on the usefulness of the simulation suite based on the ease with which developers can integrate or communicate with external libraries and services.

Although Modbus communication is required for this case to be successful, a Java implementation is clearly not considered among the tasks for a general purpose simulation

suite. A Google search on *"java Modbus"* presented an open source implementation of this protocol named *Jamod*. While the development of a dedicated eM-Plant-Modbus library took several ($\approx$ 3) weeks, the availability of this verified, validated and documented library allowed us to achieve DSOL-Modbus communication within hours. This we consider to be a scientific validation of the value of service oriented computing applied to a simulation suite.

The seamless integration between DSOL and Jamod is presented in Figure 6. The class diagram illustrates how the shadow memory, the input registry and DSOL's event library provide the communication between the PLC and the emulated devices.

The `ModbusMemoryImage` class specifies our implementation of the shadow memory. It implements Jamod's `ProcessImageImplementation` interface which embodies a set of operations for resolving and installing the basic functions presented in Table 1. The interface is implemented by our `ModbusInputRegister` class, which extends DSOL's `EventProducer` class. The `ModbusMemoryImage` further implements DSOL's `EventListener` interface and is asynchronously notified whenever the bit value of an input register is changed. The `ModbusOutputRegister` follows the same structure, but is, because of readability, not presented in this Figure.

3.2 The Specification of Emulation Components

A class diagram of one of the simulated devices is presented in Figure 7. For reasons of readability only this simple component, i.e. the sensor, is presented. All other input and output devices are specified in a similar manner.

The `Sensor` class extends the `Device` class. The fact that a sensor has no embedded knowledge of any communication protocol is illustrated in Figure 7. Since only semantic operations, e.g. the `isState()` operation, are implemented, this simulated device class could well be included in a more general purpose simulation library.

Specific bitwise Modbus memory updates are accomplished by the `SensorInterface` class. This class is asynchronously subscribed to state changes of the sensor to ensure a loosely coupled, efficient communication protocol. The principle of inheritance is applied with the creation of the `AbstractInterface` class. This abstract class uses the *ModbusInput* shown in Figure 6.

We expected a more elegant simulation model due to our ability to use Java's 3D library. Instead of defining hard coded relations which would inevitably result in emulated objects moving over predefined tracks, the DSOL model uses 3-dimensional bounds to see whether objects intersect. To illustrate this functionality, we present the `detect` method of a `Sensor` in Figure 8:

The current `view`, i.e. the volume representing the range of sight, of the potentially moving sensor is computed in line 235. This `view` is an instance of `Bounds`, which defines a convex, closed volume that is used for various intersection and culling operations. In line 240 the `intersects` method is invoked on this `view`. The `targets`, e.g. pallets or cars, are provided as an argument; these `targets` are resolved from a `context`.

3.3 The Specification of the DSOL-PLC Communication

(Schludermann, Kirchmair, and Vorderwinkler 2000) discuss time constraints in an emulation model. These constraints impose great challenges on the simulation environment and on the underlying operation system. To understand these challenges we illustrate the activity sequence of DSOL's simulator, i.e. the realtime clock.

Every time period the simulator sends the input part of the shadow memory to the PLC over the Modbus protocol. Then the simulator reads the output memory from the PLC and notifies subscribed listeners, i.e. Modbus outputs, in case the values have changed. These Modbus outputs block the simulator thread while they invoke mapped semantic operations on the device they control, e.g. `crane.stop()`. After completing this notification, the simulator fires an update animation event, and the CAD based graphical user interface will be redrawn.

The reinforcement welding equipment at Dycore requires a maximum time period of $30 \cdot 10^{-3}$ seconds. For the emulation to succeed, DSOL's emulation framework must guarantee the above sequence is completed within this period. The DSOL emulation model therefore was designed with two high-priority threads, i.e. the simulator thread and the communication thread, and one low priority animation thread. The loosely coupled relation between component behavior and animation furthermore ensured that while the refresh rate of the model was related to the period of the PLC ($\approx$ 35 Hz), the refresh rate of the animation could be slower ($\approx$ 5 Hz). The value of this distinction is that the increased priority of the communication thread allowed the required period of the emulation model to be reached (see Figure 9). Although the values presented in Figure 9 show that DSOL was in general well able to communicate every $30 \cdot 10^{-3}$ seconds with the PLC, there were occasions when a positive backlog occurred.

Because of Java's ability to spread tasks over multiple threads and to differentiate their priorities, DSOL outperformed eM-Plant by a wide margin in this task. Although we have not been given detailed information on the performance of TBA's eM-Plant model, the claim of DSOL substantially outperforming eM-Plant has been made by TBA's engineers. This achievement supports the value of interface based design. The openness of the simulation suite invited us to design a specific, performance aware `RealTimeClock` class which implements the `SimulatorInterface`.

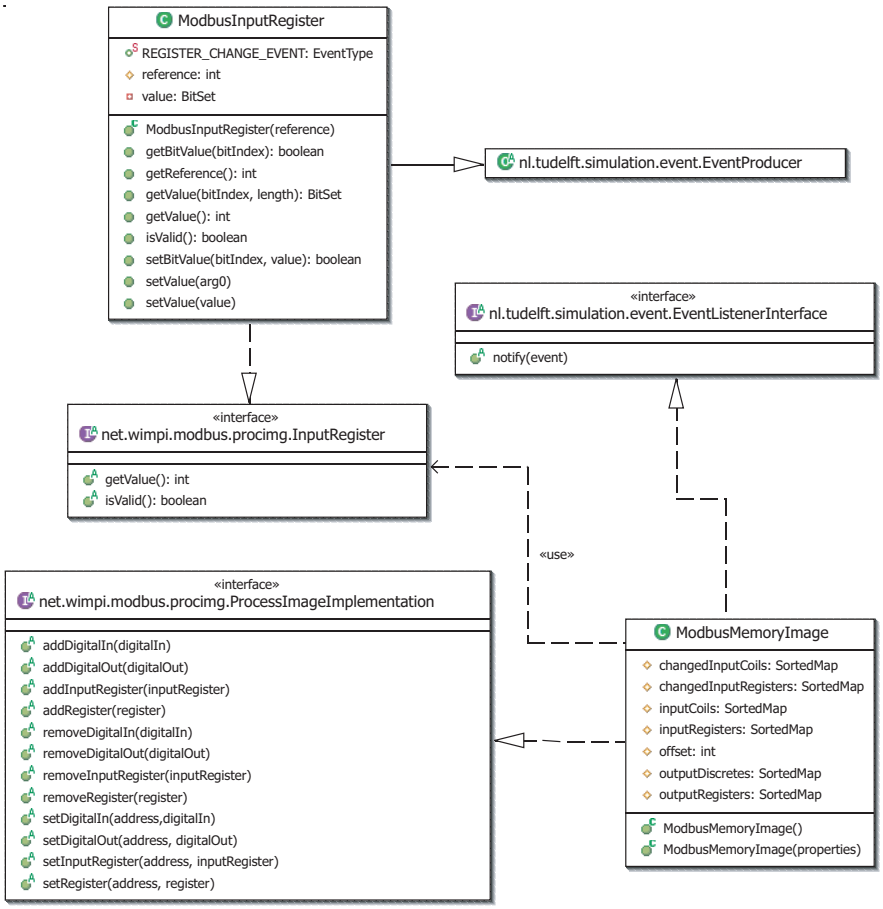

Figure 6: DSOL-Jamod Interdependence

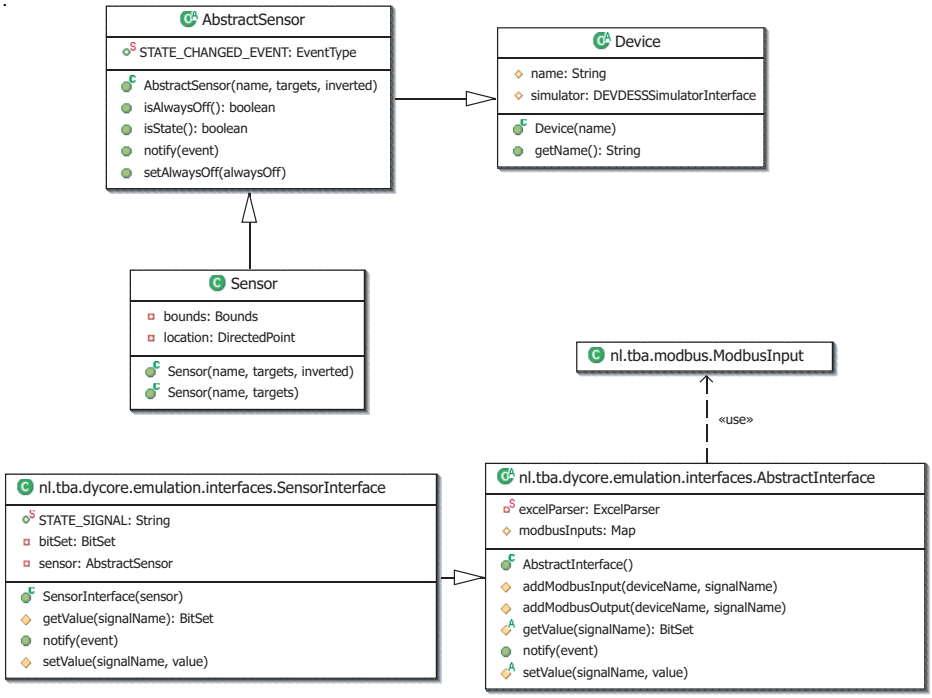

Figure 7: The `Sensor` Emulation Component

```
225 /**
226   * detects a locatable object
227   *
228   * @return whether the sensor has detected a Locatable
229   */
230 private boolean detect()
231 {
232   try
233   {
234     //We compute the space we can currently oversee.
235     Bounds view = BoundsUtil.transform(this.getBounds(), this
236         .getLocation());
237     for (Iterator i = his.targets().iterator(); i.hasNext();)
238     {
239       LocatableInterface locatable = (LocatableInterface) i.next();
240       if (view.intersect(BoundsUtil.transform(locatable.getBounds(),
241         locatable.getLocation())))
242       {
243         //Our view intersects with this target.
244         return true;
245       }
246     }
247   } catch (Exception exception)
248   {
249     Logger.warning(this, "detect", exception);
250   }
251   //Nothing detected
252   return false;
253 }
```

Figure 8: The `Detect()` Method

```
-----------------------------------------------------------------%
Tally: Backlog of ModbusConnectionThread                         %
N=8840 MAX=305.0 MIN=-35.0 AVG=-29.9495475113 STD=16.00013964397 %
-----------------------------------------------------------------%
```

Figure 9: The Measured Backlog (milliseconds) in the DSOL-PLC Communication

3.4 Experimentation

We present the graphical user interface of the Dycore emulation model in Figure 10. In this user interface the underlying Autocad files are rendered by an external, open source, geographical information system service named *Gisbeans* (Jacobs and Jacobs 2004). We consider the ease of using this external service, which is in no way related to the domain of simulation, to again act as a validation for the service oriented paradigm. The popup screen titled H2031 illustrates how devices can be individually and independently controlled.

4 DISCUSSION AND CONCLUSIONS

The most important question yet to be answered is to what extent this case supports our hypothesis that a service based simulation suite provides more effective emulation support. To answer this question we need to clarify the differences between the two specifications.

The first appearance of the service oriented paradigm underlying DSOL is the seamless integration of several off the shelf libraries and services. Poi was used to read/write Microsoft Excel files, Gisbeans was used to render CAD files and Jamod was used to communicate over the Modbus protocol. This is in clear contrast with the eM-Plant implementation developed by TBA. Requirements in eM-Plant are either natively supported or require dedicated, proprietary engineering.

The openness of the DSOL platform proven to be crucial for meeting the performance constraints of the case. The fact that tasks can easily be dispersed over several threads, each with a specific priority meant that the performance of DSOL's realtime clock outperforms eM-Plant substantially.

The loosely coupled relation between component behavior and animation ensured that while the refresh rate of the model was related to the period of the PLC ($\approx$ 35 Hz),

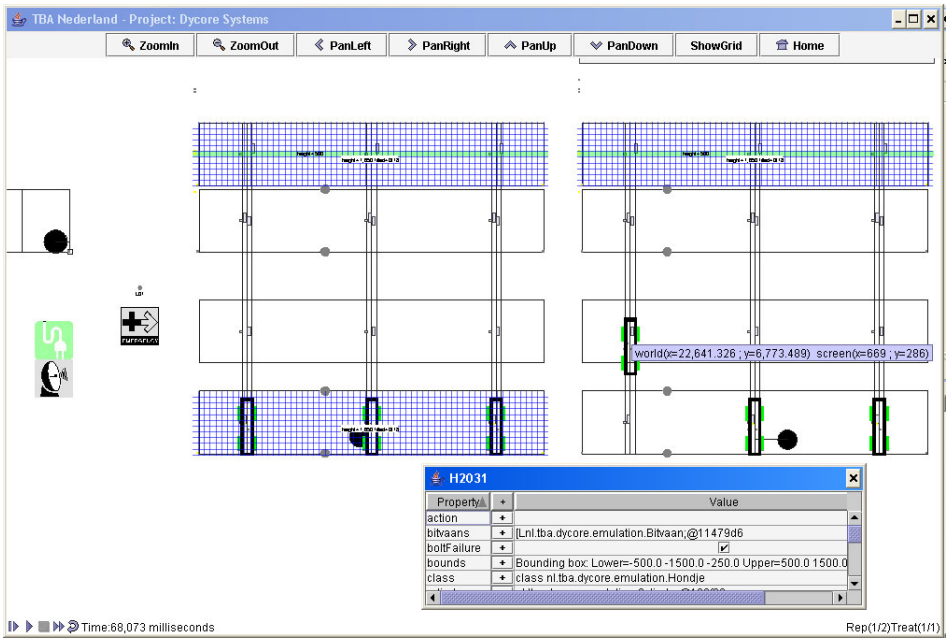

Figure 10: The Graphical User Interface

the refresh rate of the animation could be slower ($\approx$ 5 Hz). These measurements are conducted on a 1Ghz Pentium III, 512MB RAM system with DSOL 1.6 on JRE 1.4.2. The value of this distinction is that through the increased priority of the communication thread, the required period of the emulation model could be reached (see Figure 9). Although the values presented in Figure 9 show that DSOL was in general well able to communicate every $30 \cdot 10^{-3}$ seconds with the PLC, there were occasions were a positive backlog occurred. This justifies further research on the applicability of DSOL on a real-time operating system.

The inclusion of Java's 3D model resulted in a track-less infrastructure model. The sensors, conveyors and cranes can actually scan their neighborhood for pallets to be moved or lifted. Such a three-dimensional library is not available in the eM-Plant specification, which results in more constraining, dedicated relations between infrastructural objects if eM-Plant is used.

As these differences show, the service oriented paradigm underlying DSOL especially seems to show its added value whenever models require sophisticated domain specific challenges. In the comparison between the two projects several advantages of DSOL can be seen. For a large number of cases the currently used software still meets most of the requirements. But where traditional simulation environments are designed for a one purpose, one formalism and one target audience, this case clearly shows the added value to be gained from an interacting, open simulation suite.

5 OBTAINING THE SOFTWARE

DSOL is published under the General Public License. More information on the license can be found at `<http://www.gnu.org/copyleft/gpl.html>`. The DSOL project description can be found at `<http://www.simulation.tudelft.nl>` and the software can be downloaded from `<http://sourceforge.net/projects/dsol/>`.

REFERENCES

Banks, J. 1998. Principles of simulation. In *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, ed. J. Banks, 3–31. New York: NY, USA: Wiley-Interscience.

Jacobs, J., and P. Jacobs. 2004. Gisbeans: a Java library for geographical information systems. Available via `<http://gisbeans.sourceforge.net>` [accesed October 21, 2004].

Jacobs, P., N. Lang, and A. Verbraeck. 2002. A distributed Java based discrete event simulation architecture. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C.-H. Chen, J. Snowdon, and J. Charnes, 793–800. San Diego, CA, USA: Institute of Electrical and Electronics Engineers: ACM Press. Available via `<http://www.informs-cs.org/wsc02papers/102.pdf>` [accesed October 21, 2004].

Matic, N. 2001. *Introduction to plc controllers*. Belgrade, Serbia: mikroElektronika. Available via `<http://www.mikroelektronika.co.yu/`

english/product/books/PLCbook/plcbook.htm> [Accesed October 28, 2004].

Modicon 1996. Modbus protocol reference guide. Technical Report PI-MBUS-300, Modbus-IDA. Rev. J.

Schiess, C. 2001. Emulation: debug it in the lab — not on the floor. In *Proceedings of the 33rd conference on Winter simulation*, ed. M. Rohrer, D. Medeiros, and M. Grabau, 1463–1465. Arlington: VA, USA: Institute of Electrical and Electronics Engineers: ACM Press.

Schludermann, H., T. Kirchmair, and M. Vorderwinkler. 2000. Soft-commissioning: hardware-in-the-loop-based verification of controller software. In *Proceedings of the 32nd conference on Winter simulation*, ed. P. Fishwick, K. Kang, J. Joines, and R. Barton, 893–899. Orlando: FL, USA: Institute of Electrical and Electronics Engineers: ACM Press.

TBA Nederland 2000. Tba nederland specialised in simulation of factories, harbours, airports and railsystems. Accesed via <http://www.tbanederland.nl/default.asp> [accesed October 21,2004].

Whorter, S., B. Baker, and G. Malan. 1997. Simulation system for control software validation. In *Proceedings of the 1997 SCS Simulation Multiconference*. Atlanta: GA, USA.

AUTHOR BIOGRAPHIES

PETER H.M. JACOBS is a PhD. student at Delft University of Technology. His research focuses on the design of simulation and decision support services for the web-enabled era. His working experience within the iForce Ready Center, Sun Microsystems (Menlo Park, CA), and engineering education at Delft University of Technology founded his interest for this research. His e-mail address is <p.h.m.jacobs@tbm.tudelft.nl>.

ALEXANDER VERBRAECK is an associate professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of open and generic libraries of object oriented simulation building blocks in Java. Contact information: <a.verbraeck@tbm.tudelft.nl>.

WILLIAM RENGELINK (M.Sc. in Aerospace Engineering) is a senior consultant at TBA Nederland , a leading simulation consultancy company in the Netherlands and active for ports, airports and factories all over Europe. TBA is involved in a number of state-of-the- art automation projects concerning baggage handling at Amsterdam Airport Schiphol and the testing of the Control System in the Port of Rotterdam.His work is in the fields of simulation and emulation for logistic systems within airports and factories. He is involved in various international projects. His email address is <william@tba.nl>.