A toolchain for Rapid Control Prototyping using Rexroth controllers and open source software

Nils Menager Niklas Worschech Lars Mikelsons
Bosch Rexroth AG
Rexrothstr. 3, 97816 Lohr a. Main

Abstract

Taking a look at project costs from a financial point of view, the commissioning times of new industrial systems become more and more important, as they significantly drive the costs. Hence, the reduction of commissioning times is part of current research. Besides the use of simulation and the coupling between hardware and software (Hardware-In-The-Loop-Simulations), *Rapid Control Prototyping* offers a huge potential to reduce commissioning times. Until now, most of the toolchains use special hardware in combination with commercial simulation software, which leads to some serious drawbacks. In this work, a toolchain for Rapid Control Prototyping using industrial controllers and open source software is presented.

Keywords: Rapid Control Prototyping; Modelica; OpenModelica; Hardware-In-The-Loop; Bosch Rexroth; real-time simulation

1 Introduction

1.1 Motivation

According to the V-model of mechatronic product development (VDI 2206, [1]), the first step during the design of a new industrial system, after a rough pre-calculation of the required components, is to set up a simulation model of the system under consideration. Therefore, at Bosch Rexroth, our in-house tool *Rexroth Simster* is used. *Rexroth Simster* is a simulation environment, which allows object-oriented modelling of technical, mainly mechanical, electrical and hydraulic systems. Clearly, the tool includes powerful numerical solver to perform the simulation. After suitable components and parameters are determined inside the simulation environment, the controller concept has to be tested using a real hardware controller. At this point, until now, the simulation model of the controller is not used any further and the controller is set up from scratch inside the development environment of the controller. At Rexroth, mostly *IndraWorks* as standard tool for the development of controller algorithms and the design of the whole controller is used. In many cases, even the developed controller structure inside the simulation tool is not used anymore. Instead, predefined standard control algorithms are used.

However, it is clearly desirable to adapt the complete control algorithm from the simulation environment. There are already possibilities to use virtually designed control algorithms on real-time operating systems using Hardware-In-The-Loop-Setups. Two possibilities are using a dSPACE [2] system or a xPC system in combination with *Matlab/Simulink* [3]. Though, using such a toolchain leads to three serious drawbacks. First, these systems are very expensive. Second, even in that case the control algorithm has to be re-implemented on the real control hardware after testing on the real-time system. Furthermore, it has to be taken into account that the usage of such commercial real-time systems leads to dependencies to external software (e.g. *Matlab/Simulink*). Clearly, such a dependency for the generation of code for the PLC (*Programmable Logic Controller*) is not desirable. Moreover, if new features should be implemented, this is a big disadvantage, because the code generation of the commercial tools can not be modified.

The aim of this work is to set up a toolchain for Rapid Control Prototyping with a Rexroth controller (IndraControl L45) using open source software and Modelica for the modeling part, i.e. a toolchain, which is completely independent from external software and hardware. To be more precise, this toolchain enables the engineer to transfer virtual controller models modeled in Modelica to standard Rexroth controllers. To validate the functionality, in this contribution, the controller is used in a Hardware-In-The-Loop setup. The validation of the controller in combination with a real system is part of current work.

1.2 Outline of this paper

In the first section of this paper, a short introduction into Rapid Control Prototyping is given. Furthermore, a standard toolchain for using Rapid Control Prototyping nowadays is discussed. In the second section, after requirements for the toolchain have been defined, the specific parts of the toolchain are presented. The functionality of the toolchain is finally verified using a Hardware-In-The-Loop setup consisting of a real Rexroth hardware controller running controller code of an industrial control algorithm created in Modelica and a model of a hydro-mechanic system. In the last part, a short summary and an outlook on further investigation is presented.

2 Rapid Control Prototyping

Rapid Control Prototyping is a computer-aided method for developing and testing control algorithms quickly on real-time operating systems. This approach allows to investigate how the control algorithm will behave later on the real hardware controller. Rapid Control Prototyping includes all steps between the definition of the controller specifications and the implementation of the final control algorithm. The single steps during the Rapid Control Prototyping process are shown in the V-model in figure 1. The left part of the V-model shows the way from the specification of the requirements to the implementation of the controller,

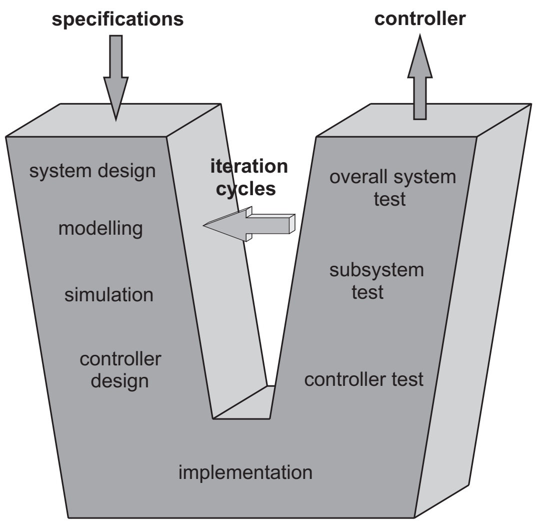

Figure 1: V-model showing the Rapid Control Prototyping process

in the right part the functionality of the implemented algorithm is verified. The functionality of the algorithm is then compared to the requirements specified at the beginning of the cycle. If there are differences between the required specifications and the actual functionality, another iteration cycle is necessary. This procedure is repeated until the actual behavior of the controller and the specifications fit together.

2.1 dSPACE/Matlab

Nowadays, the usage of Rapid Control Prototyping is already standard in different industry branches, e.g. the automotive industry. Therefore, for example dSPACE (*Digital Signal Processing And Control Engineering*) systems can be used. On a dSPACE box a real-time operating system is working which allows to execute code in real-time on the device. To generate the code, e.g. special toolboxes from *Matlab* can be used. These toolboxes allow to generate executable code for the dSPACE box in a very short time. Using this toolchain it is possible to develop and test control structures on a real-time operating system in an easy and fast way.

Although this method offers the possibility to test algorithms quickly, it does not avoid the re-implementation on the final hardware device, which is a big disadvantage. Besides the fact, that a re-implementation is time consuming and a possible error source, the portability is potentially incalculable. It cannot be guaranteed that the controller on the final target behaves in the same way as the controller on the test device. Other disadvantages are the costs of such a dSPACE system and the dependencies to the commercial software.

Because the *Matlab Code Generation* only works for special operating systems and special hardware, there are no direct possibilities to adapt the code generation for other devices like the Rexroth controller. Of course, it could be tried to wrap the code could for the use on other devices, but even in this case, this code has to be compiled for the target operating system. If only one single part of the code cannot be compiled for the operating system, it is impossible to execute the code on the hardware. Hence, it is necessary to develop a toolchain, which is open source and therefore applicable to different hardware devices. The development of such an open source toolchain is the topic of this work.

3 Realization of the toolchain

For the realization of the toolchain, clearly, different tools are needed. The structure of the toolchain is shown in figure 2. The starting point is the simulation environment *Rexroth Simster*. As already said in the introduction, the first step during the development of a new system is to set up a simulation model of the system and the controller inside the simulation environment. The simulation tool *Rexroth Simster* will, in one of the next releases, support both the usage of models from an internal library (written in C/C++) and of Modelica models. It makes sense to use Modelica models, because they offer three big advantages. The first one is, that Modelica models are easy to create and support an object-oriented modeling structure, either using a graphical user interface or the Modelica editor. Second, in order to simulate Modelica models, they have to be compiled to C/C++. There exist both commercial (e.g. *Dymola*) as well as open source (e.g. *OpenModelica*, *JModelica*) Modelica-compilers. In this work, of course the open source *OpenModelica*-compiler (OMC) is used. Furthermore, Modelica is a widespread language, which is used frequently in the industry.

The controller model shall then directly be used on the real PLC. The PLC used here is a common Rexroth controller (Rexroth IndraControl L45). Therefore it is necessary to compile the model of the controller for the operating system of the hardware. The real-time operating system running on the hardware is *VxWorks*. To execute the code and run the simulation on the PLC, a simulation core, which runs and manages the simulation, is additionally required. This simulation runtime has also to be compiled for the operating system *VxWorks*. To compile the system and the runtime and load the compiled libraries onto the hardware, the development environment *WindRiver Workbench* is used. This environment includes among other things a *VxWorks* compiler. The simulation of the controller model can then be executed in parallel to the main thread on the PLC. This first step, to compile and load the model on the PLC, is shown via the grey, dashed arrow in figure 2.

To allow the exchange of data between the PLC and the simulation environment during a HiL-simulation, an interface between the newly created thread running the controller code (controller thread) and the original *IndraWorks* thread (main thread) is required. To get access to the Rexroth IndraControl L45, the software *IndraWorks* is used. *IndraWorks* is a standard tool for the development of control algorithms and the design of Rexroth PLCs. The connection between the controller thread and the main thread is realized with a function block according to IEC 61131 [4] inside the *IndraWorks*-application, which is used as an interface. In order to run the HiL-simulation, data has to be exchanged between the PLC and *Rexroth Simster*. Therefore, the MLPI (*Motion Logic Programming Interface*) is used [5]. The MLPI is a programming interface for high level programming languages (C/C++/C#/VBA/Java/LabVIEW/...). It can be used to write applications, which can be used to configure and run a Bosch Rexroth controller which supports the MLPI interface technology, like the IndraControl L45.

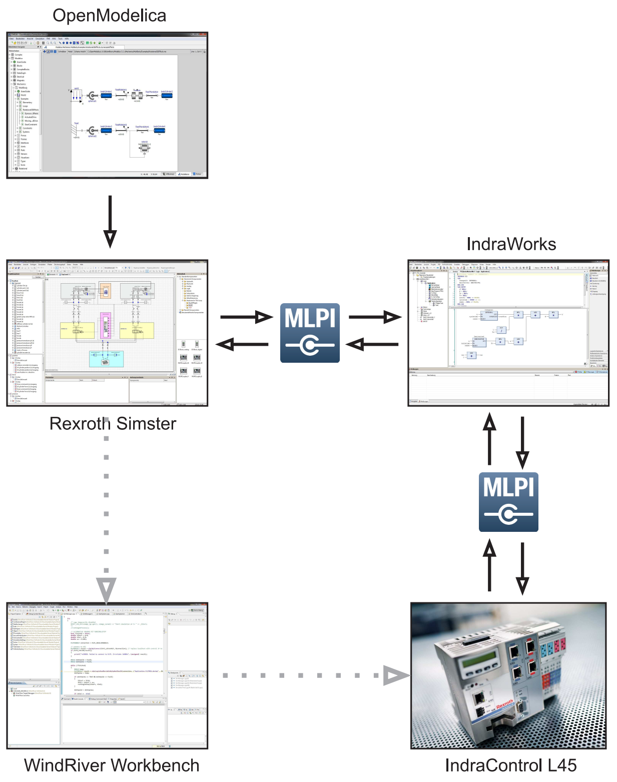

Figure 2: Structure of the toolchain

3.1 Used software components

In the following sections, a short overview and more detailed information about the tools used in this work are given.

3.1.1 Rexroth Simster

The simulation environment *Rexroth Simster* is an in-house tool developed by Bosch Rexroth. It covers multiple domains (mechanic, hydraulic, electric) and has been developed for the design and optimization of controlled automation systems. The component library includes both generic and Rexroth specific components, which can simply be placed on the worksheet using drag and drop. *Rexroth Simster* includes a special simulation core. This simulation core offers an interface for models from the standard *Simster* libraries, which are implemented in C/C++. The C/C++ code generated by the *OpenModelica*-compiler implements the same interface. Thus, the simulation core can handle both models from the own internal library and Modelica models. Detailed information about the developed simulation runtime is given in [6].

3.1.2 WindRiver Workbench

WindRiver Workbench is an eclipse-based development environment for *VxWorks* and is used in version 3.3. *VxWorks* is a real-time operating system, which is mainly used in embedded systems and is the operating system running on Rexroth hardware controllers. Included in the development environment is a *VxWorks* compiler, which generates executable code from C/C++-code. The tool is used to compile both the controller model and the simulation core for the use on the hardware controller. The classes inside the simulation core are compiled into dynamically linked libraries, which leads to *.out*-files. These *.out*-files have then to be moved to the internal flash card of the Rexroth hardware. This is done using an FTP client.

3.1.3 IndraWorks

IndraWorks is a tool developed by Bosch Rexroth and is used as standard tool for the development of control algorithms and the entire configuration of the PLC. Inside *IndraWorks*, an application to run on the hardware can be created. After having configured the connection parameters (IP address, type of connection), the algorithms are developed using the IEC 61131-3 standard PLC programming languages. Furthermore, additional languages especially for the use of motion commands (PLCOpen) are available. During the runtime of the controller, the process can be visualized and monitored using plotter and other visualization tools.

3.1.4 Motion Logic Programming Interface

The *Motion Logic Programming Interface* is an interface supporting many high level programming interface and is also developed by Bosch Rexroth. Using this interface, it is possible to write applications to configure and run Bosch Rexroth devices which support the MLPI interface technology. It contains a set of headers and libraries. There are 8 different libraries, which allow access to different parts of the controller:

- *mlpiAPILib* includes functions to connect and disconnect MLPI
- *mlpiSystemLib* includes functions to read system information like temperature, diagnosis data and the firmware version
- *mlpiParameterLib* includes functions to read-/write parameter
- *mlpiLogicLib* includes functions to start/stop/reset the PLC, load PLC programs, browse/read-/write symbol variables
- *mlpiMotionLib* allows access to general motion functions, single axis motion, cyclic commands and synchrone axis motion
- *mlpiContainerLib* allows cyclic read/write access with fast container buffer mechanism
- *mlpiWatchdogLib* includes functions to monitor the user application
- *mlpiTraceLib* includes functions for the trace configuration and to add, collect and view debug information.

There are four different MLPI toolboxes, each supporting a different programming language. Here, the toolbox for C/C++ is used. In this work, MLPI is used on the hand side to realize the data exchange between the simulation tool *Rexroth Simster* and the hardware controller. Furthermore, MLPI is used as interface between the user and the controller to change controller parameter inside the controller code. The structure and functional principle of MLPI is explained in [5].

3.2 Connecting the different components to the toolchain

To ensure the functionality of the toolchain, some additional aspects must be considered while connecting the different parts to the toolchain. The different aspects are discussed in the following, each in an own subsection.

3.2.1 Modifications inside the simulation core

An important point is the library handling in *VxWorks*. Hence, each location inside the code loading a library has to be modified. To ensure the functionality in both the new operating system (*VxWorks*) and the old environment (*Microsoft Windows*), pre-processor commands are used to decide which implementation is used. Using this method the same runtime can be used in both *Rexroth Simster* and on the PLC, which is an important requirement. To load dynamic libraries in *VxWorks*, the following basic framework has to be used:

```
int libraryFile = open("lib.out", O_RDONLY, 0777);
if (libraryFile == ERROR)
  // Error loading library

MODULE_ID c_moduleId = loadModule(libraryFile,
                          LOAD_ALL_SYMBOLS);

close(libraryFile);
if (c_moduleId == NULL)
  // Unable to load as module

extern SYMTAB_ID sysSymTbl;
SYM_TYPE symType;
double (*funcPtr)(int);

if (symFindByName(sysSymTbl, "name",
     (char**) &funcPtr, &symType) == ERROR)
  // Symbol not found

double a = funcPtr(2);
```

After the library is loaded by the *open* command, all symbols are loaded using the *loadModule* function. This allows to get access to the functions inside the library. The next step is to create a function pointer. The function pointer in this example points on a function, which gets an integer as input variable and which returns a double value. The last step is to search a specific function in the symbol list using the *symFindByName* function. This function pointer can then be used to call the function inside the library.

3.2.2 Synchronization of the HiL-setup

The next aspect is the synchronization between the simulation of the system inside *Rexroth Simster* and the code execution on the PLC. It is clear, that both processes have to run synchronized, so that the exchanged data fit together. *Rexroth Simster* is a windows application and therefore, without any modifications, not real-time capable. That means, that the calculation time depends on the complexity of the model and the workload of the used operating system. Thus, the simulation can be faster or slower than real-time. In contrast, a PLC is a hard real-time system with a fixed cycle time. At the beginning of one cycle, the inputs of the controller are read. Then, the control algorithm is executed and the output data based on the input data is computed. The last step is to write the calculated values to the output. Therefore, the controller expects input signals in real-time. Using a standard PLC containing a standard *IndraWorks* application (i.e. the controller computes the output in real-time) in a Hardware-In-The-Loop-setup with the *Rexroth Simster*, there are two possibilities: either the simulation is forced to run in real-time or the controller has to be adapted to the simulation speed of the simulation environment.

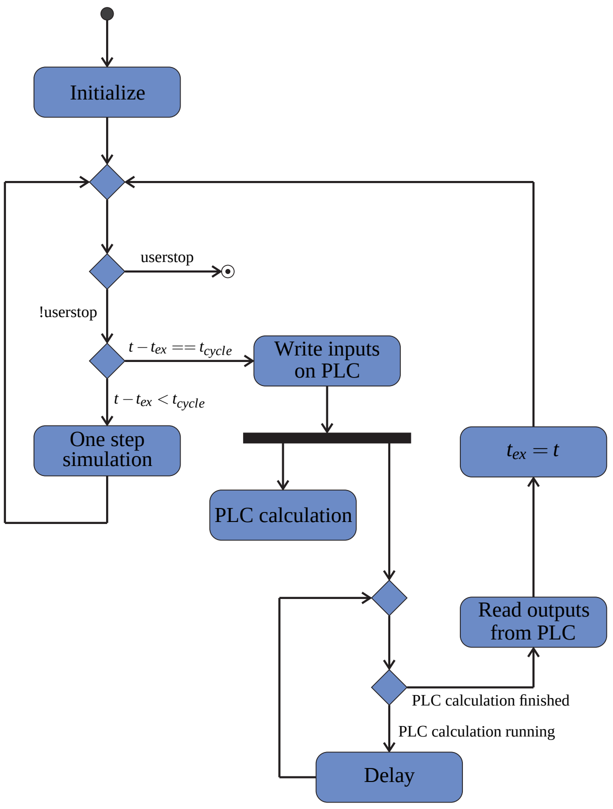

Figure 3: UML Diagram showing the synchronization

Because the second way has some big advantages, the slowdown of the controller has been realized. One main advantage is the user-friendliness. Main users of this toolchain are engineers like start-up engineers, who shall design a new industrial system and do not have the possibility to use a real-time operating system on their working computer. Additionally, this way is more comfortable, as there are no limits with regard to applicable numerical solve algorithms or the complexity of the model.

To realize the slowdown of the controller, a trigger variable inside the controller application to start the task is necessary, which activates one calculation step on the controller. After the simulation is progressed by the length of the cycle time, the simulation is stopped and the values of the input variables on the controller are set inside the *IndraWorks* application using MLPI commands. After that, the trigger variable is set to *true*, which starts the calculation of one single step of the controller. At the end of the computation, another variable, which indicates that the computation is finished, is set to *true*. The simulation environment reads the output values from the controller and sets both the trigger variable and the variable indicating the end of the calculation to *false*. These steps are repeated until the end of the simulation time is reached.

In the setup described in this contribution, however, the control algorithm is not implemented in *IndraWorks* and therefore not computed inside the main thread of the controller, but in the separate controller thread running in parallel to the main thread, which makes the situation more complicated. This means, that the program is not controlled via an *IndraWorks* task like in the case discussed before.

3.2.3 Establishment of a connection between the different threads

The next challenge is the establishment of the connection between the controller thread and the main thread, because the MLPI commands allow only access to variables and functions inside the *IndraWorks* application running on the PLC. The connection between both threads is realized via a function block inside the *IndraWorks* application. It is possible to link an external implementation to a function block, so this function block has no own implementation.

To ensure that the application will find the missing function implementations, the external implemented functions have to be registered using the MLPI function *mlpiLogicPouExtensionRegister* from the *mlpiLogicLib*. This function provides the possibility of using C/C++ extensions within the IEC 61131-3 environment *IndraWorks* and describes the mapping between the function block name in *IndraWorks* and the function name in the C/C++ implementation. The variables that shall be exchanged can now be defined as variables inside the function block in the main thread. Then, both the simulation inside *Rexroth Simster* and the controller on the PLC can get access to the variables using MLPI functions as well as read and write the variables. The structure of the communication between the two parallel threads is shown in figure 4.

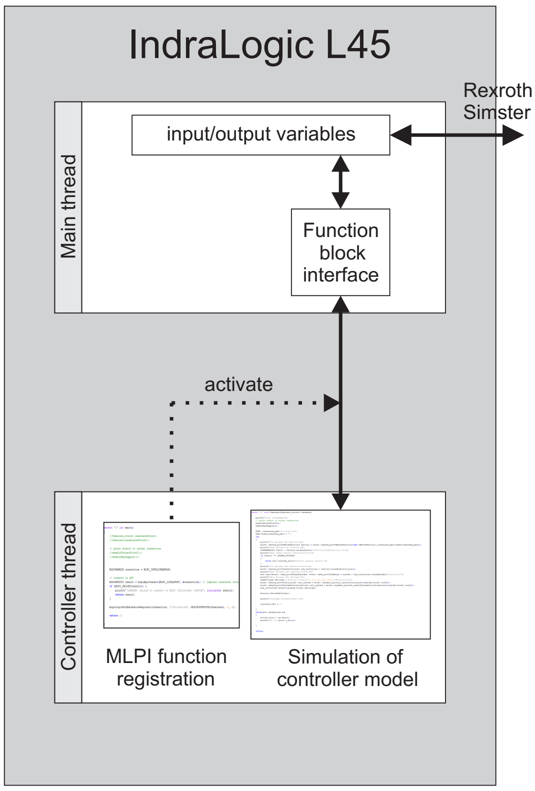

Figure 4: Interface between main thread and controller thread on the hardware controller

After having established the connection between the two threads, the synchronization between the simulation of the system inside *Rexroth Simster* and the controller can be realized analogous to the technique described before. The trigger variable is defined inside the *IndraWorks* application and can be set by both the simulation environment and the controller thread.

3.2.4 Initialization of the toolchain

The next aspect to be considered is the initialization of the code execution on the controller. After the compilation of the simulation core and the controller code, all dynamic libraries are available on the internal memory. To start the controller, a main function to manage the code execution (load the libraries in the correct order, call the functions to initialize the solver and the system, start the code execution) is necessary. This function has to be executed before the controller starts, so that all libraries are loaded and all instances of the classes are already initialized. This is the function later called automatically via the function block interface.

3.2.5 Resulting workflow for the toolchain

Regarding all the aspects discussed before, the resulting workflow for the toolchain can be derived. Assuming, that the simulation model of the system and the controller inside the simulation environment *Rexroth Simster* already exist, the first step is to compile the simulation core and the controller code for the operating system of the controller (*VxWorks*). Therefore, a new project inside WindRiver Workbench has to be created. The classes inside the simulation core have to be compiled into dynamically linked libraries, which leads to *.out*-files. These *.out*-files have then to be moved to the internal flash card of the Rexroth hardware. This is done using an FTP client. The next step is to register the executable main function containing the initialization of the simulation on the controller (see section 3.2.4). The registration is done using the MLPI function *mlpiPouExtensionRegister*. The syntax is as follows:

```
MLPIRESULT mlpiLogicPouExtensionRegister (
                    const MLPIHANDLE connection,
                    const WCHAR16* name,
                    const MLPIPOUFNCPTR function,
                    const ULONG signature = 0,
                    const ULONG version = 0).
```

The first input parameter is the connection handle automatically created when a connection to the hardware via MLPI is established, the second parameter is the name of the POU (Program Organization Unit) in *IndraWorks*, in this case the name of the function block, the third parameter is the function pointer to the C/C++ implementation, while the fourth and fifth describe the signature of the POU interface and the version of the POU library, if implemented within a library, and have not necessarily to be set, as they are predefined with 0 [7].

Now the implementation of the function block interface inside the *IndraWorks* application is made known to the *IndraWorks* application. As the next step, the *IndraWorks* application, which only consists of the function block with the external implementation and the definitions of the variables to be exchanged during the simulation as well as the trigger variable (see section 3.2.2), can also be uploaded to the hardware device (if the *IndraWorks* application is uploaded before the registration of the functions is executed, there will be linker errors for the external implementation of the function block).

The next step is to start the initialization of the controller on the hardware, i.e. to active the function block. Therefore the task controlling the function block has to be started. This can again be realized via MLPI. The task is defined as *triggered task*, which allows to start the task setting the activation variable to *true*. The start of the main function effects, that all necessary libraries are loaded and the simulation manager is started. Inside the simulation manager, a query is continuously performed, whether the trigger variable to start one calculation step is set or not. For the connection to the hardware device from the *Rexroth Simster* side, a special component is necessary, which has been developed for HiL-tasks (MLPI-Coupler). The component has several inputs and outputs and contains the MLPI-commands to both write the data from the different inputs on the device and read the data from the device and set the values to the outputs of the component. The names of the variables inside *IndraWorks* can be set as component parameters.

The last step is to start the simulation inside *Rexroth Simster*. The cycle time between the exchange of the data can also be set in the MLPICoupler component. The simulation of the system triggers then the simulation on the hardware device. The synchronization between both simulations is realized as described in section 3.2.2.

3.2.6 Automation of the toolchain

The toolchain presented in this work is not fully automated until now. In one of the next releases of *Rexroth Simster*, Modelica support will be added to the simulation environment. Until now, an additional Modelica environment is necessary to build up the controller simulation model. In the future, Modelica models can directly be created inside the simulation tool. The code generation is integrated into the simulation core, so that the executable code can directly be generated. This allows a fully automated toolchain, where the controller model can be set up inside the simulation environment and automatically be compiled and sent to the controller. Starting the simulation inside *Rexroth Simster* activates the toolchain (compile the controller model, transfer the code to the controller, load the simulation core libraries, start controller code execution).

4 Application on an example system

To verify the functionality, the toolchain is used to develop an appropriate control structure for the control

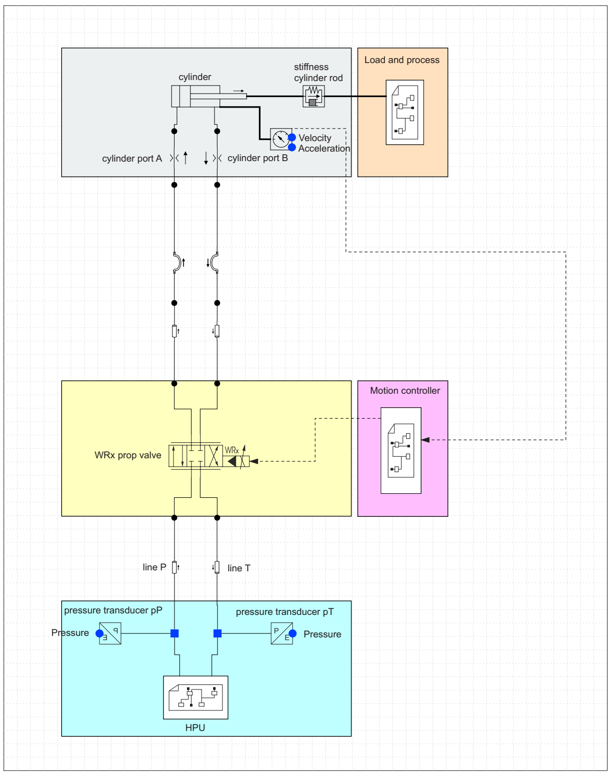

Figure 5: System model inside Rexroth Simster

of an industrial hydro-mechanical system, namely a single hydraulic axis. This system model is build inside *Rexroth Simster* and simulates the behavior of the hydro-mechanical system. The structure of this simulation model can be seen in figure 5. This model also includes a control loop which has been developed virtually inside the simulation environment.

4.1 Model of the system

The single axis model consists of five sub-blocks, each highlighted in a different color. The light blue block is the HPU (hydraulic pumping unit), i.e. it realizes the oil supply for the hydraulic system and includes the oil tank, a variable pump which is powered by an electric motor. The motor speed is power- and pressure-controlled. Additionally an accumulator is included in the HPU to ensure the oil supply for temporary high demands on oil.

The yellow block shows a generic WRx proportional valve, which limits the volume flow of the hydraulic fluid. Using the input signal port of the valve, the spool position of the valve can be modified. The valve's dynamics is modelled with a PT2-behavior with power limit, the flow is modelled via a characteristic curve depending on the piston stroke ($Q = f(s)$).

The connections between the pump and the valve are modelled by lines including frictional losses. The differential cylinder is modelled inside the grey box. The simulation model of the cylinder considers Stribeck friction (static friction, running friction and Coulomb friction), internal leakage and external leakage. Additionally the cylinder model has two end stops for the piston, which are implemented using momentum conservation (optionally a coefficient of restitution can be specified).

The load is modeled inside the orange sub-box and considers the force resulting from the load mass, the gravity force, the plastic and the elastic deformation. The velocity of the cylinder piston is defined by the user and is available as characteristic curve in the form $v = f(t)$. The position profile for the cylinder, which can be obtained through integration of the velocity profile, is shown in figure 6 (blue curve).

4.2 Modeling the controller

To realize the control, the current position of the cylinder piston has to be measured. Therefore, the internal position measuring system of the cylinder, which is included in the cylinder model, is used. As first try for the controller structure a position controller is used. The position controller compares the current position of the piston with the desired position from the profile. The difference (control error) is then multiplied by a gain factor (P-controller). The profile and the controller structure are implemented inside the rose box in figure 5.

4.3 Starting the RCP process

To transfer this controller model to the real hardware device, the toolchain which has been explained in section 3 is used. As numerical solver, the explicit Euler algorithm is used. Note, that the controller model contains an ODE from the integrator component to calculate the position from the velocity profile. For the verification, the results of this HiL-simulation using the developed toolchain are compared with the results produced by a simulation of both the system and the controller inside *Rexroth Simster*. Both results are shown in figure 6, the complete simulation inside *Rexroth Simster* in red, the simulation using the Rapid Control Prototyping toolchain in green.

It can be seen that both curves are very similar, but of course not identical. This is because the controller

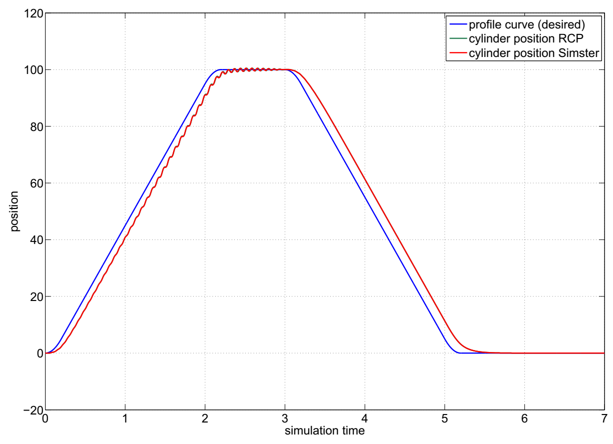

Figure 6: Results from the first controller implementation and comparison with the Rexroth Simster simulation

inside *Rexroth Simster* is a continuous controller and gets the position update from the cylinder in every solver time step, while the simulation on the hardware device (like every hardware controller) is updated only in the cycle time of the connection between simulation and hardware. But as the differences are very small, the functionality of the toolchain and the data exchange can definitely be verified.

4.4 Improvements on the controller

If again the V-model of the controller development in figure 1 is considered, the first cycle is now finished. But, if the current result is compared to the desired result, another iteration cycle due to the existing oscillations is necessary. It is clear, that a simple P-controller cannot fulfill the control task. In the second iteration, a velocity feed forward to minimize the gap and an additional control part to minimize the oscillations is integrated into the controller. Therefore, the Modelica code is modified inside the simulation environment. After suitable parameters are determined, the controller structure is again transferred to the IndraControl L45 controller using the toolchain to investigate the functionality on the hardware. Figure 7 shows the result of the improvement of the control algorithm.

Taking a look on the results after this iteration cycle, it can be seen, that the oscillations could be removed. It can be assumed, that the developed controller structure, in general, is suitable to solve the control task in this example (the desired positions are reached without oscillations). In the practice, one or two additional iteration cycle would be performed in order to tune the parameters to maybe get an even better parameter set, that shifts the current position more towards the desired position to minimize the gap. However, this is skipped at this point.

5 Summary and outlook on further investigations

In this work, a toolchain for Rapid Control Prototyping using an industrial Rexroth hardware controller based on open source software is presented. This toolchain allows to reduce commissioning times, avoiding the re-implementation of the controller structure from the simulation environment inside the development environment of the hardware controller. Furthermore, the toolchain is based on open source software. This ensures, that the functionality is independent from software developed by external companies, i.e. additional features can easily be implemented.

The functionality of the toolchain is also verified with an example. Here, the big advantages of Rapid Control Prototyping get visible. The control structure is developed and pre-tested easily inside the simulation environment. To test this control algorithm on the real hardware, until now, it was necessary to re-implement the algorithm. Using the developed toolchain, the re-implementation is no longer necessary, because the final hardware, which is applied later on the real system, is used for the Rapid Control Prototyping Process. The development process consists of several iteration cycles (see 1). In this example, we used three iteration cycles (third one not explicitly shown here), hence, three re-implementations could be saved. In

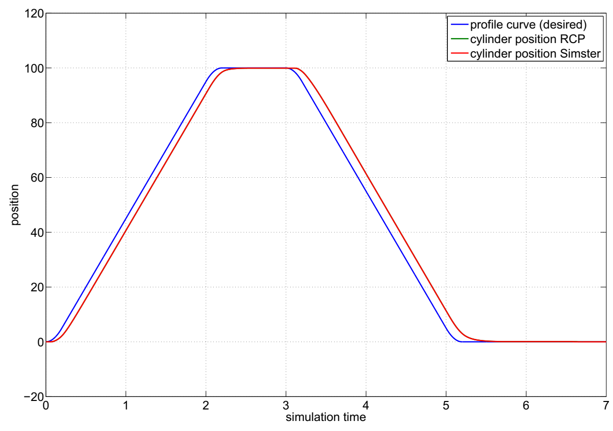

Figure 7: Results from the second controller implementation and comparison with the Rexroth Simster simulation

more complex systems, of course, the controller structure also gets more complex.

In this contribution, the RCP toolchain is verified virtually in a HiL-setup. In the future, the controller executing the code has to be tested on a real system. Therefore, the *IndraWorks* program has to be adapted. In the HiL-setup, the simulation inside *Rexroth Simster* uses MLPI commands to write the values to the variables in the *IndraWorks* application. If the controller is connected to a real system, the input and output signals are transferred via the input and output ports on the controller. Additional code for the mapping between the I/O ports and the variables is necessary. An important point is the observation of the calculation times. It has to be investigated in the future, how the strict observance of the calculation times can be guaranteed.

Another part of the future is work is to fully automate the toolchain, as described in section 3.2.6.

The simulation core can not only be used to simulate controller models to realize Rapid Control Prototyping and couple hardware and software in a Hardware-In-The-Loop simulation. It is also possible to simulate whole system models, which opens the door to many other fields of application. One field of application are alternative control concepts like Model Predictive Control. Model Predictive Control calculates the current control action by solving an optimal control problem at each sampling instant using the current state of the plant as initial state. The cost function of an optimal control problem is optimized subject to different constraints. One main constraint is the system dynamics in the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$. This condition requires the simulation of the system in each optimization step.

References

[1] Richtlinie, V. D. I. "2206"' - Entwicklungsmethodik für mechatronische Systeme, Berlin 2004

[2] http://www.dspace.com/en/pub/start.cfm

[3] http://www.mathworks.com

[4] John, K.H.; Tiegelkamp, M. - IEC 61131-3: Programming Industrial Automation Systems, 2010

[5] Engels, E.; Gabler, T. - Universelle Programmierschnittstelle für Motion-Logic Systeme - Struktur, Funktionen und Anwendung in Forschung und Lehre, Tagungsband AALE 2012

[6] Worschech, N.; Mikelsons, L. - A Toolchain for Real-Time Simulation using the OpenModelica Compiler. In: Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany.

[7] Bosch Rexroth AG - Motion Logic Programming Interface (MLPI) Documentation

[8] GNU GCC Release Information, URL: http://gcc.gnu.org/gcc-3.4/