Case Study: Palladio-based Modular System for Simulating PLC Performance

Jens Frieben
Fraunhofer IPT
Project Group Mechatronic Systems Design
Zukunftsmeile 1
33102 Paderborn, Germany
Email: jens.frieben@ipt.fraunhofer.de

Dipl.-Ing. Henning Heutger
PHOENIX CONTACT Electronics GmbH
Business Unit Control Systems
Dringenauer Strasse 30
31812 Bad Pyrmont, Germany
Email: hheutger@phoenixcontact.com

***Abstract*—Modern facilities controlled by programmable logic controllers (PLC) have to fulfill a broader range of tasks and meet higher performance requirements than a few years ago. They have to cope with their basic control tasks as well as communicating with each other or external systems. The selection of an appropriate performance class during the early development phases is based on the experience of a developer or expensive and time consuming tests. In this paper, we present an approach to simulate the PLC and its environment to predict the CPU utilization. For this purpose, the PLC and its internal and external influences on the CPU utilization have been modeled with Palladio. Based on the simulation results of different usage scenarios, the developer is now able to choose a fitting performance class without conducting expensive tests.**

I. Introduction

The vision of increased flexibility in production by smart factories is becoming reality. Modern facilities and production plants are developed towards smaller production sizes and better reuse of machines. For this goal, the communication between different stations and even different plants, as well as more self-aware systems, have to be realized. This for example allows ad-hoc reconfiguration resulting in shorter setup times. Therefore, the Programmable Logic Controller (PLC) controlling and coordinating the different machines in a factory, need not only to handle more actuators and sensors (I/Os), but also react faster to more events and even provide additional services to external systems. To cope with these requirements, the PLC must fulfill higher and higher performance requirements.

Figure 1 shows a standard PLC communicating with various systems over one or more fieldbusses. It is mandatory that the provided input by different sensors is processed by the PLC and the correct commands are send out to the actuators in time. Any violation of these hard real-time constraints could lead to failures in the plant or, in case of safety-critical systems, cost lives. Their controlling functionality is performed by several programs running in tasks. These tasks are regularly executed in intervals (e.g. every four milliseconds) and called *CyclicTasks*. Also, other execution strategies like *EventTasks* exists, which are triggered by events. If the execution of the programs set inside a cyclic task takes longer than their specified time intervals, a run-time exception is thrown and the PLC stops all programs for safety reasons. Therefore it is most important to assure that the PLC provides enough performance for the in-time execution of programs. On top of the basic controlling functionality the PLC has

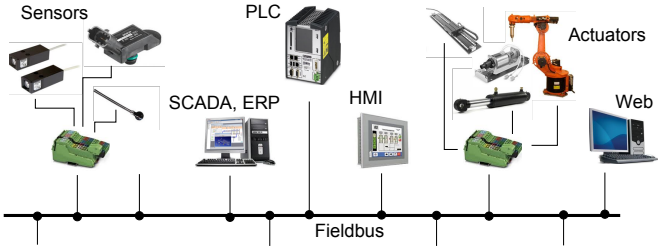

Figure 1. The PLC and its environment

to provide additional services. Figure 1 shows also external systems connected via fieldbus that communicate with the PLC. These services might belong to the upper management layer that coordinates the whole production plant (e. g. SCADA[1]) or machine to machine communication to negotiate new setups or configurations for different products. These service functions influence the performance of the PLC as well.

During the development of automated systems, the specific details of the plant, the control programs and the services are not fully worked out. The most important factor, the control programs, are often finished after the plant has been set up or at least close to its end. This complicates the early selection of a suitable PLC performance class, considering the performance of the CPU.

In practice, an appropriate performance class is estimated based on the experience of the developer or with the help of time-consuming and expensive tests. Despite the fact that these tests run in parallel to the late development phases, they can often not consider or replicate

[1]Supervisory Control And Data Acquisition (SCADA) systems monitor and control industrial processes

all the factors and scenarios that should be taken into account.

In cooperation with Phoenix Contact, we developed an approach for simulating their PLC to predict the future utilization of the CPU. This allows the developer to pick a fitting PLC performance class for the specific task at hand. These simulations, which can be easily set up and executed, provide a fast feedback to the developer and can be refined along the development process of the automation system. The key data used as parameters for the simulation is updated based on the current or future development states. This approach helps to cover more usage scenarios than it would be feasible with tests and to save costly setups.

The focus of this work lies on predicting the CPU utilization of the PLC and not its hard real-time behavior like it is done for other embedded devices[1]. However, the simulation can provide approximated insights to the response times of the running programs and anticipate possible run-time exceptions.

Figure 2 shows the process from specifying the automation system to its simulation. First, the developer creates an abstract *automation model* containing the key data used for the simulation. In the next step, this information is transformed into Palladio Component Models (PCM). These models belong to the Palladio [2; 3] performance prediction framework which carries out the simulation. The result of this simulation is a collection of sensor values which are used to calculate the overall CPU utilization.

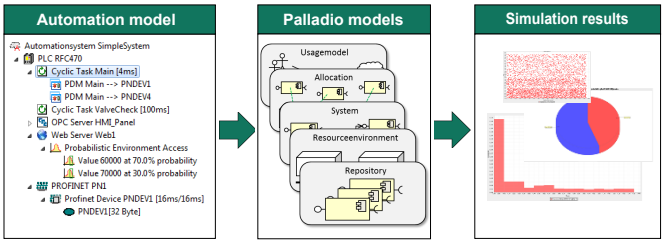

Figure 2. From automation model to simulation results

The paper is structured as follows. We introduce the different influence factors of a PLC in Section II. In the following Section III the automation model used to specify these influence factors is presented. In addition, the Palladio models used for the simulation and the transformations to generate them from the automation model are described. Section IV gives a short overview of the simulation results. We conclude this paper in Section V and give a short outlook of future work.

II. Influence factors

In this section we explain the influence factors that are covered in our approach and shown in figure 3. We introduce each factor but will detail only a few to give an idea of the different levels of detail used for the final simulation model.

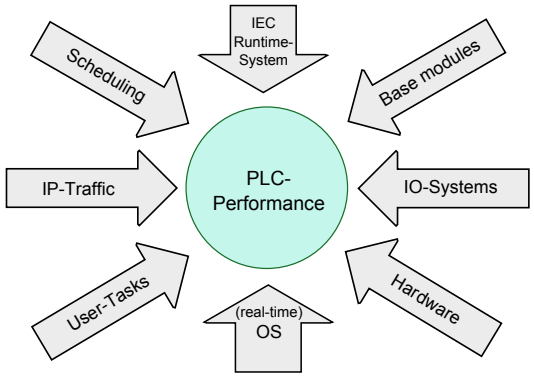

Figure 3. Performance influence factors of a PLC

- **Hardware** is one of the most obvious influence factors on PLC performance. Primary factor is the clock rate of the PLC, which is specified as instructions per simulation time unit. For our first attempts to simulate the Phoenix Contact PLC we focused on a specific product, the RFC 470. This PLC has a clock rate of 1 GHz and therefore is specified with 1.000.000 instructions per time unit, setting the simulation time unit to 1 ms.
- **User-Tasks** are the most important influence factors on the PLC performance. Each task carries out one or more programs that calculate the commands for the actuators and therefore controls the behavior of the machine. There are different kinds of tasks: event based, system, default, and cyclic. For this use case, only the cyclic task - which is the most commonly used task type - has been regarded. For the cyclic task, three values can be specified. The interval between executions (*cycletime*), the priority of the task (*priority*), and an estimated worst-case-execution-time (WCET) of the programs (*executiontime*). The execution time is specified in milliseconds and currently is based on the developers estimation or measurements, in case the programs already exists.
- **(Realtime) OS** and **IEC-Runtime system** belong to the PLC performance base load. The IEC-Runtime system (called ProConOS in Phoenix Contact PLC) executes the programs defined in the tasks, written in the languages specified in the IEC 61131-3[4] standard. The IEC-Runtime system is executed on top of an operating system. Both systems, IEC and OS, put a certain amount of load onto the PLC.
- **Scheduling** takes place in the IEC-Runtime system to schedule the different tasks according to their priorities and on the level below regarding the operating system. Caused by preemption, low priority tasks might have significantly increased response times. Therefore the scheduling must be incorporated into the simulation to get precise results.

- **IO-Systems** are a crucial part of automation systems. Sensors provide input for a PLC and actuators influence their environment. The data send from and to the PLC is transmitted via fieldbusses. A fieldbus is an industrial network system for real-time distributed control. A PLC might be connected to one or more different fieldbus systems like PROFINET[5] or INTERBUS[6]. Depending on the PLC, the communication over fieldbusses is either realized in hardware (e.g. DPM[2], FPGA[3]) or in software. A communication over hardware also uses a certain amount of CPU time, but is significantly faster than a software solution. Due to the real-time constraints (fieldbusses must adhere short communication cycles down to $250\mu s$), they consume a significant part of the CPU performance.
  For the case study conducted with Phoenix Contact, the first fieldbus system under investigation is PROFINET. We focused only on the CPU utilization, not remote communication effects or response times. It supports different operating modes ranging from non-real-time (e.g. HMI access) to real time use in motion control. Sensors and actuators are called PROFINET-Devices (short pndevice). Data between PLC and pndevices are send in predefined time slots in a cyclic manner. Each pndevice consists of one or many modules, on which sensors and actuators can be connected to. The module size defines the amount of data that can be send to or from a pndevice. At the PLC, the data is provided to the running programs as variables, also called process data. Therefore, each task has a connection to a specific module in the PROFINET. A message send from the PLC to a pndevice has a fixed length based on the sum of all modules. The payload it transports is data that will update the programs variables. Figure 4 shows a short summary over the important properties needed to specify the influence factor IO-System on the PLC. The pndevice that *sends* and *receives* messages in specified intervals, the modules (*modulsize*) attached to it, and the data (*datasize*) transferred to the PLC.
- **Base modules** (services) are used for secondary functions of the PLC. Most of these services provide means for communication with other devices or systems. The three services handled for this study are:
  - **OPC**: OPC stands for for Object Linking and Embedding (OLE) for Process Control and is used to exchange data between control devices. The OPC-Server provides the variables used by the programs for either visualization (user interfaces) or for SCADA-systems for controlling purposes. The OPC-client requests these variables in predefined *intervals*. In addition, the *variable type* and the number of variables have an impact on the PLC performance.
  - **FTP-Server**: A FTP-Server is used to download log-files from the PLC or to upload new projects (specifying tasks and programs) to it. The *filesize* and the access *frequency* is important for simulating the CPU utilization.
  - **Web-Server**: The Web-Server allows remote users to view the state and variables of the PLC. Of course this generates a certain amount of CPU usage depending on the *access time* and the *file size*.
- **IP-traffic** is an influence factor depending on the fieldbus implementation. Traffic, whether from the control network or other IP-based devices, can cause additional CPU load for analyzing and forwarding the messages either to the windows- or the PROFINET communication stack.

[2]Digital Processing Module (DPM)
[3]Field-Programmable Gate Array (FPGA)

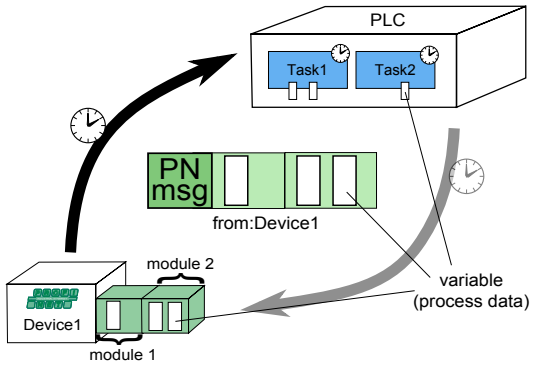

Figure 4. Sending and receiving of PROFINET messages

III. Modeling

In this section the different models and their relations to each other are described. One important goal is the easy use and handling of a simulation. The target audience for carrying out performance simulations are electrical engineers and, for future plans, non-technical personnel. Due to the complexity of modeling (even a medium sized PLC) with Palladio, a domain specific language for abstracting the details was necessary. In Section II the most important influence factors have been identified. These factors can now be specified with the automation model. The following subsections highlight some details about the automation model, the Palladio models that are used for the final simulation, and the transformation rules for making the transition between them.

A. Automation model

The automation model is used to specify the influence factors of the PLC in a simple way. Its structure is similar

to the tool PC Worx[4] used to program and configure the fieldbusses of PLC. First we created an Eclipse Modeling Framework (EMF) based meta model to be able to specify the automation system in a precise and automatically analyzable form. This allows the developer to create an automation system with a PLC and its specific settings and environment.

Figure 5 shows an excerpt from the automation model meta-model without properties. Root element is the *AutomationSystem* element, which can contain multiple PLCs. Currently the simulation only supports one, but for future use, a complete system of PLCs communicating with each other, should be realized. Several *Service*s may be added to the PLC, which can be used to set up Web-Server or OPC-Server loads. Also, one or more *Task*s can be added to the model. The properties that can be specified for a task, like calculation time or priority, are identified in Section II. The *ProcessDataMappings* are used to connect variables of the task with specific *IOModules* which are added to *IODevices*. They are part of a fieldbus like PROFINET.

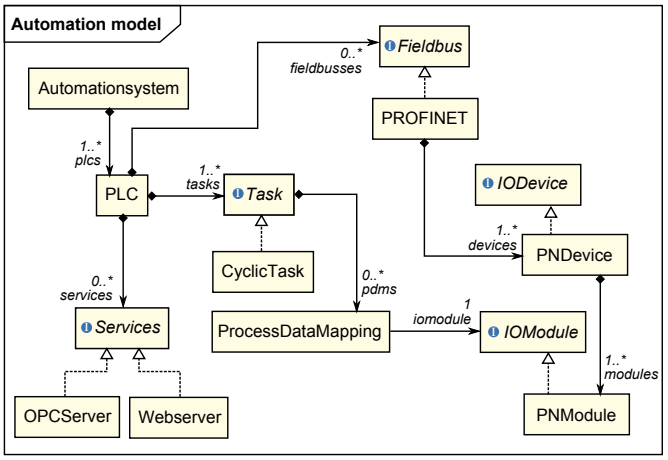

Figure 5. Automation model meta-model (excerpt)

A possible instance of this meta-model is shown in figure 6. The screenshot of the generated EMF-Editor shows two *CyclicTasks* (*Main* and *ValveCheck*) which are executed every 4 ms, resp. every 100 ms. Each task needs exactly 1.5 ms execution time. In this example, different variables are connected to the pndevices via *ProcessDataMappings* (PDM), which specify the size of the variable send over the fieldbus in bytes. The pndevices haven been created with an send/receive interval of 16 ms. Two additional services are the OPC-Server, providing several variables at a fixed refresh interval and a Web-Server with varying access times.

B. Palladio models

In this subsection, we explain how we measured the different influence factors and captured them in a simulation model. For our approach, we used the Palladio Component model (PCM) and SimuCom[7] for the simulation. PCM consists of five models (repository, usage, system, allocation, and resource environment) each specifying a specific aspect of the automation system. The repository is used to model the influence factors and internal processes of the PLC. Most important element to model the CPU utilization is the *ResourceDemand*. It is specified in *Service-Effect-Specifications* (SEFF) which can be compared to methods or functions of programming languages. They might call internal and external actions which refer to other SEFFs. Inside a SEFF, a *ResourceDemand* for a given resource (e.g. CPU or HDD) can be defined. When calling for example the SEFF named *receive_PROFINET_Message*, the simulation will stress the CPU with 1000 units of CPU time. This corresponds to the number of instructions used to read from the network hardware and write the data into the memory. A SEFF also supports a conditional control flow, which allows the modeling of alternative resource demands depending on given parameters.

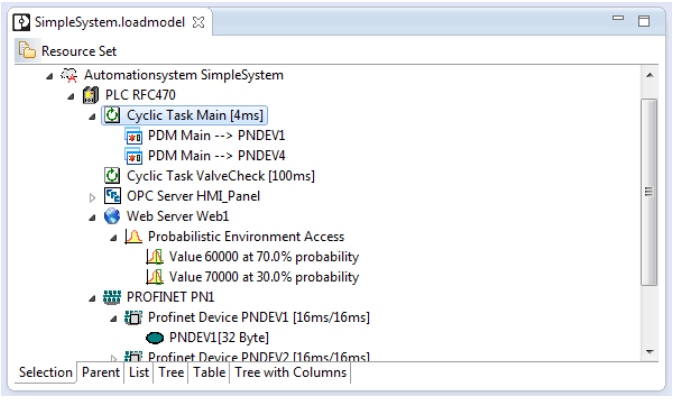

Figure 6. Screenshot of the eclipse based automation model editor

For our approach, we provided only the resource "CPU", since it is the primary goal to predict the CPU utilization. The HDD, a compact flash card, was neglected in our models due to rare accesses and its fast response time. Including the network interface into the model, as a possible bottleneck for some operations, is up for future work. The other PCM models are used to specify how a system is composed from elements specified in the repository (system model) or how the system is deployed (allocation model) onto the hardware (resource environment model). The usage model is used to specify how often, in which intervals, and with what parameters the SEFFs are called.

In the following we will give an insight how the PLC and its environment have been modeled with PALLADIO and how we measured the necessary resource demands for the different parts of the model. For this case study, we focus only on four influence factors: the operating system including its scheduler, cyclic tasks, the PROFINET

[4]Phoenix Contact development and configuration tool

load, and the OPC-Server.

Operating system & scheduler: To model the operating system in PALLADIO, two points have to be considered. First the scheduler and second the base load generated by the OS internal processes. Scheduling influences the response times for the different processes, not the CPU utilization in general, which is focused in this work. Still, a realistic preemption of processes was requested for future work. High priority tasks for the communication (like PROFINET) will always be executed before tasks, which can lead to run-time exceptions. Palladio supports some abstract out-of-the-box schedulers, but don't provide the means to set up the process priorities in the PCM. Therefore the scheduler and means to model the process priorities had to be implemented. The operating system for the RFC 470 is a modified version of WindowsCE 5.0, for which a new scheduler specific *resourcetype* has been created. This resourcetype is referenced in the resource environment, which is used to specify the hardware and its properties. Accompanying the resourcetype is a scheduler-file that configures the properties of the OS scheduler. These properties influence how the OS handles the different running processes. The most important settings are:

- Prioritylevel, which specify the range of integer values that can be used to put processes in an order. This order is used to process higher priorities (0) before lower priorities (255).
- Process-Handling defines in which order the processes are run or continued. The strategy used by WindowsCE 5.0 is that lower priority processes are preempted by higher ones. Processes with the same priorities are handled with a Round-Robin strategy.
- Timeslice/Quantumsize defines a unit for available running time. After a process completes its quantum, WindowsCE 5.0 may choose to run another process based on priority or state. The default quantum time set in the resourcetype is 25 ms.
- Starvationboosts are used to push threads which have been preempted for too long. The setting for the simulation is 0.

The simulation uses the configuration to set up the inbuilt scheduler, which manages the different incoming resource demands. For further information about the scheduler, its configurations, and functionality see [8].

In addition to the scheduling, the basic load of the PLC has to be considered. With a special version of the RFC 470 is it possible to see the CPU utilization calculated by the operating system. Unfortunately this includes not only the basic OS functions, but also the IEC-Runtime system executed on start-up. Since both influence factors could not be separately measured, the IEC-Runtime load has been combined with the OS basic load to a module named *WinCE5* load. To determine the ResourceDemand, the PLC has been set into a running state with no projects, services and connected devices. The CPU utilization has been read off the display, showing 16%, spiking between 15 % to 17 %. With a 1 GHz CPU, resulting in 1,000,000 instructions processable per millisecond in the simulation model, the ResourceDemand for WinCE5 is set to 160,000 CPU units. To reflect the spikes, a range of possible values has been set up in the SEFF's ResourceDemand with a DoublePDF-formula:

$$DoublePDF[(155200.0;\ 0.0)(157600.0;\ 0.05)\ (162399.99;\ 0.90)(164800.0;\ 0.05)]$$

The deviation of the exact value can also be specified in a transformation rule (see section III-C) allowing a fine tuning without changing the repository model. To trigger the SEFF containing the resource demand for WinCE5, a usage scenario with an closed workload has been created. The workload, containing just one worker, starts every millisecond. The result of a simulation with just the WinCE5 load is shown in figure 7. The red dots show the simulated load of the windows operating system. The load varies exactly in the ranges specified by the DoublePDF function.

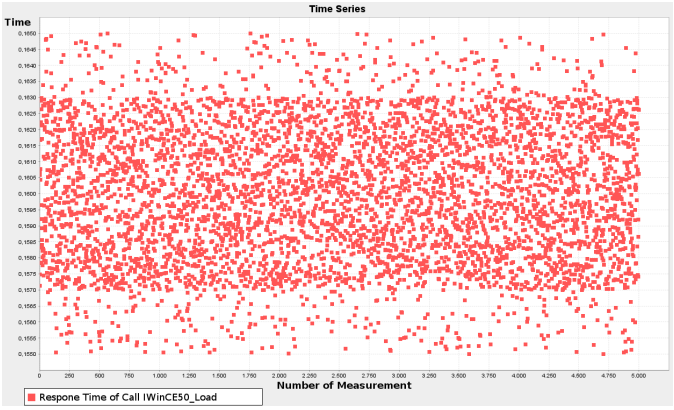

Figure 7. TimeSeries for the WindowsCE and ProConOS baseload

CyclicTasks: The creation of the PALLADIO models for cyclic tasks is pretty straight forward. Each CyclicTask has a property *executiontime* which specifies how many milliseconds the task will put a load on the CPU. This time can be converted to a ResourceDemand by checking how many instructions the hardware can process per millisecond. More complex is the issue of cyclic executions with fixed intervals. PALLADIO supports two types of workloads - open workload and closed workload. Both types wait a predefined time and call the specified SEFFs. After the SEFF has been executed, they start waiting again. This behavior can be applied if no preemption takes place and the tasks are executed always with the same response times. In this case, the execution time can be subtracted from the cycletime and the difference used for the workload specification. Due to preemption it is possible that the task response times vary. Therefore we use

a workaround to start the SEFFs in exactly the same intervals via closed workloads. To do so, we use a TASK-component with a SEFF including a fork. This fork calls in one execution path the SEFF with the actual ResourceDemand inside a LOAD-component. The other path contains no actions and therefore returns immediately to the workload, restarting the waiting time. Figure 8 shows the two components and their interfaces for the task *Main*.

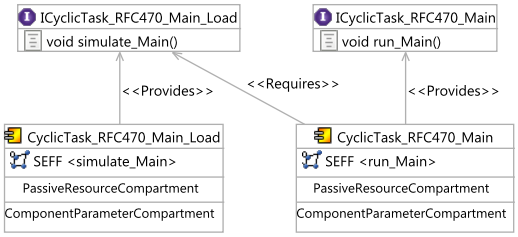

Figure 8. Interfaces and components for CyclicTasks

IO: PROFINET: A more sophisticated example is the modeling and simulation of PROFINET, which supports different performance modes. One of this modes, IRT (Isochronous Real-Time), has been investigated in more detail and its CPU utilization depending on the previously identified parameters (see Section 3) modeled in Palladio. For determining the resource demands, a series of measurements conducted by Phoenix Contact has been used. They instrumented the communication stack and important parts of the data flow from the network interface up to the tasks. Based on these information, four phases have been identified, which are shown in figure 9. In the first phase (receive), the pndevices send their

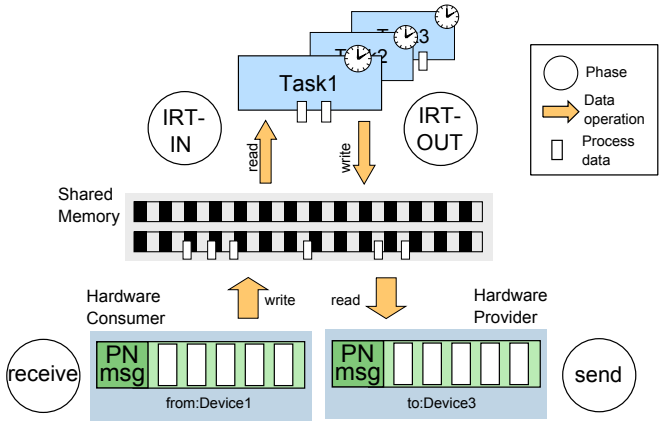

Figure 9. The four phases for the processing of PROFINET data.

messages via the network to the PLC. There, a function takes the message from the hardware consumer (interface buffer), decodes and analyses it, and puts the data into a predefined position in a shared memory. This function uses a certain amount of processing power, depending on module size (=message size) and the size of the data that will be copied. In the next phase (IRT-IN), each tasks refreshes its variables with data from the shared memory. The cycletime of the tasks and the size of the variables influence the consumption of CPU resources. After the task has finished, the variables are written back into the shared memory (IRT-OUT), packed into a message and send out to the pndevices (send). Based on the provided measurements, a function for each phase could be determined that is used to calculate the resource demand. At the current level of detail, we use the functions to calculate a mean value for each phase. This simplifies the calculation and we are able to combine all phases in one usage scenario. This might result in a slightly different load profile, but due the nature of the IRT-modus, in which the tasks usually have cycle times of 1 to 2 ms this is a reasonable deviation. For other PROFINET modes like *RT* new models and functions are needed.

An exemplary function to calculate the CPU resource demand for the receiving of messages and storing their values in the shared memory is shown in equation 1. The function *conDev* returns all pndevices for the PLC, *modules* returns a set of modules for a pndevice, *modulsize* retrieves the size of the given module, and *sendInterval* returns the millisecond between each message send to the PLC.

$$f_{PN_{receive}}(plc) = 10 + \sum_{conDev(plc)}^{d} \frac{(0.023 * \left(\sum_{modules(d)}^{m} modulsize(m) \right)) + 2{,}380}{sendInterval(d)} \quad (1)$$

OPC-Server: The last influence factor we like to present is the OPC-Server. During the measurements, we identified the previously mentioned parameters (see Section II) *variabletype*, *amount*, and the OPC-Clients request *intervals*. Table I shows the CPU utilization for varying amounts of String variables and different refresh times.

Table I
CPU utilization for string variables with different refresh intervals (excerpt).

#vars	0	1	10	50	100	200	1000
1 ms	16%	19,5%	19,5%	21%	24%	26%	30%
10 ms	16%	19,5%	19,5%	21%	24,5%	25%	30%
100 ms	16%	16,5%	16,5%	17%	17%	18%	18,5%

For the measurements we created an IEC program running in a low priority task, performing no operations. With the tool PC Worx the amount of initialized variables could easily be adjusted. The OPC-Client has been simulated with the tool OPC Test Client, which allows to configure different refresh rates. The variable type has a

non-negligible impact on the CPU utilization. When testing only a subset of all possible variable types (including self-defined STRUCTS), the type string used up more CPU resources than others.

The result of the measurements for a refresh interval of 1 ms is also visualized in figure 10. Starting at a base load of 16 % CPU utilization, an almost linear increase up to 2000 variables can be identified. Afterwards a saddle point is observable which could be an indicator for a bottleneck. Due to the limited sensors available, we can only assume that this bottleneck is caused by the network interface, pushing out messages to the OPC-Client. Therefore, only the linear section, marked with the dotted rectangle, is used to determine the CPU resource demand, allowing us to create the following function which is parameterized by the number of variables n:

$$f_{String}(n) = 0.0105 * n + 34895$$

Such a function must be created for each variable type. Due to the similarities of most types, we combined them to groups and thereby reducing the number of needed calculations. Using String and other variables in one OPC-Server request had the effect of using the lower CPU utilization curve instead of the higher. The reason for this behavior still needs to be investigated. Finally a usage scenario can be created which calls the OPC-Server SEFF that generate CPU resource demand. A single worker in an open workload is run in intervals specified by the OPC-Clients refresh interval.

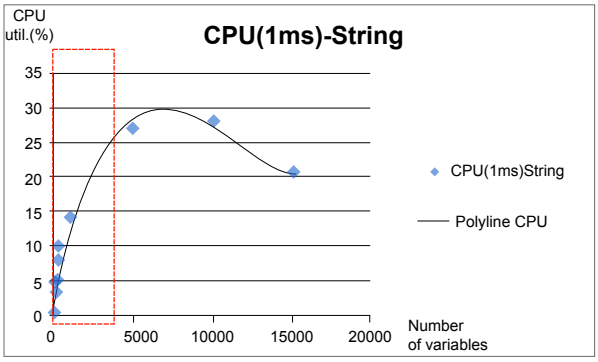

Figure 10. CPU utilization for the OPC-Server (String, 1 ms)

C. Transformation

Creating performance models with Palladio is a complex and time consuming task. For this reason, the automation model has been created, which hides the Palladio modeling from the user. This section will give some insights how the automation model is transformed to Palladio models and what advantages this approach offers.

The automation model and the five Palladio models (repository, usage, allocation, system, and resourceenvironment) are based on the EMF technology. This allows us to choose from a range of Model-To-Model approaches provided for Eclipse[5]. To select a fitting transformation technology we used the study conducted by Lehrig[9]. He provides a first guidance in form of a decision-tree. Based on the requirement to allow an easy modification of the transformation rules and a target audience of C/C++ developers, we choose QVT-Operational (QVTO)[10]. QVT-O is an imperative language designed for writing unidirectional transformations. Its imperative language is similar to programming languages like C++ and its text based specification allows an easy customization of its transformation rules. Additionally, QVTO supports QVT-BlackBox operations for invoking external code. This allows us to specify complex calculations or queries through the automation model in JAVA. Alternatives to QVT-O like ATL[11], TGG[12], or XTend[13] were not covered in Lehrig's work.

For the design of the transformation rules we followed the requirement to be as modular as possible. This lead us to the decision to create one transformation for each influence factor. A transformation contains mapping rules that specify how an object from the automation model is transformed to an object in the Palladio models. The top-level object in the automation model is of type *Automationsystem*. The root-transformation rule creates the five Palladio models based on this Automationsystem and sets up the hardware. Afterwards one or more follow-up transformations can be executed. This approach gives us the flexibility to build up the Palladio models incrementally, creating first model elements for cyclic tasks, OPC-Server, WindowsCE and so on. The functions used to calculate the different resource demands are implemented in the QVTO-Blackbox operations for each transformation. This modular approach allows us to combine different transformation rules to a product configuration. Such an exemplary configuration is shown in figure 11. The configuration named *RFC_470_Rev_2.34* is used for a specific firmware revision of the RFC 470. The firmware consists of several transformations, each creating a specific part of the Palladio models. These sub-transformations are called by the root-transformation. In case a FTP-Server firmware update enhances the performance of the service, a new sub-transformation is created and the old one is replaced. This allows a simpler creating and customizing of existing firmware configurations.

Another advantage of using QVTO-transformations to generate the Palladio models is the possibility to easy modify attributes and configurations. The listing 1 shows an excerpt from the transformation *WinCE5.0_Rev01*, which contains several global variables. Changing the priority of WindowsCE related processes can be done by just setting a new integer value. The same applies to the *osI-*

[5]Eclipse IDE - www.eclipse.org

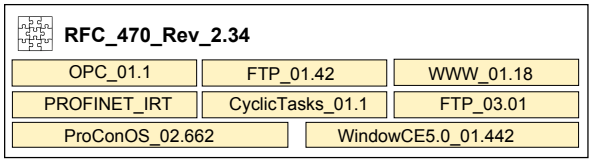

Figure 11. Product configuration for a specific RFC470 firmware version.

dlePercentage and *osDeviation*, which influences the creation of the ResourceDemand as mentioned in III-B.

Listing 1. Excerpt from the WinCE5.0_Rev01 transformation rule

```
// ------ Operatingsystem (Windows CE 5.0)
property osName : String = 'WinCE50';
property osPriority : Integer = 250;
property osIdlePercentage : Real = 0.16;
property osDeviation : Real = 0.03;
```

The validity of the resulting performance model holds for this level of detail and as long the influence factors only use the resource CPU and the scheduler manages all demands. As more fined grained actions are added to the model, interactions between the different modules (e.g. via passive resources like shared memory or FCFS queues at the network interface) should be considered, too. For this, the root-transformation, which creates all necessary models before executing the sub-transformations, could create the needed common resources as well.

IV. Evaluation

Performance tests with actual hardware in a networked environment are a time consuming and costly task. Therefore an sophisticated test covering all influence factors could not be conducted. Instead several smaller tests could be made which focus on one or two influence factors at once. These tests are based on measurements carried out in Phoenix Contact's dedicated testing facilities or in small, controlled environments.

Due to the nature of this contract research, most of the timing results and actual readouts are not approved for the public. Therefore only a few or just small excerpts from the simulation results can be shown here.

In the first test, different cyclic tasks have been simulated and compared with real executions on the PLC. For this, an IEC program has been created that allows the generation of load. The program performs stressing calculations such as cosine operations and watched its own execution time. This time is not the response time, which might be influenced by preemption, but the aggregated execution times. In Table II different tests are listed. Each test contains its simulated and real CPU utilization or run-time exception (RTX). In addition to this, the test name and a brief description of the setups are given. For example, stands *T(15 ms,5 ms),T(150 ms,5 ms)* for two tasks, in which the first produces a load of 5 ms and cycles with 15 ms, the second task with 5 ms load every 150 ms. These tests include the WinCE5.0 module and its base load. The average deviation between simulated and real values are just about 2 percent. This fault could be caused by internal caching effects that are not yet considered in our performance models.

Table II
CPU utilization of cyclic tasks with various configurations

Testname	Real(%)	Sim(%)	Description
1T	40-42	38	T(4ms,1ms)
3T	52	53	3x T(15ms,2ms)
3TSEMI6	60	63	3x T(6ms,1ms)
5T	61	63	5x T(10ms,1ms)
1TFAST	63	63	T(2ms,1ms)
T2-MIX1	48-50	50	T(15ms,5ms),T(150ms,5ms)
T3-MIX2	54	53	T(4ms,1ms), T(10ms,1ms),T(15ms,1ms)
T4-MIX3	RTX	61	T(4ms,2ms),T(15ms,2ms), T(30ms,4ms),T(150ms,6ms)
T4-MIX4	63-65	68	T(4ms,1ms),T(15ms,2ms), T(15ms,4ms),T(150ms,6ms)

Second, we present the comparison of measured and simulated CPU utilization of PROFINET running in IRT mode. Table III shows the number of pndevices, modulesize, size of the data sent (none or all), and the simulated and measured CPU utilization. Despite a constant offset of about two percentage points the values are almost identical. The measurements were collected with instrumented code that logged the execution times. This could also be a possible reason for this deviation, due to fact that logging also consumes CPU processing power.

Table III
CPU utilization comparison PROFINET real measurements with simulation (excerpt).

#Devices	3		12		16	
ModSize	1x32		1x128		1x64	
DataSize	0	32	0	128	0	64
Real(%)	27,5	32,5	33,2	50	35,5	57
Sim(%)	29,8	33,9	35,6	51,6	38,6	59,87

V. Conclusion and Outlook

Future production plants and automation systems need to cope with more complex tasks, including communications with each other and external systems. To cope with these requirements, their PLC have to handle their normal controlling tasks as well as new services. This complicates the early selection of a suitable PLC performance class by the developer. With our approach, we support this decision by providing a performance prediction of the PLC and its future environment and tasks. To simplify the simulation, we created an abstract automation model, containing all viable information about the automation system in a form that is familiar to the developer. This model is automatically transformed to the

PALLADIO models needed as input for the simulation. The transformations used for this step are created in a modular way, allowing the combination of different influence factors to a specific product configuration. Despite a full evaluation was not available at the time this paper was written, the results of smaller evaluation tests showed a promising start.

During the modeling of the internal and external influence factors on the PLC we also identified some PALLADIO features that were missing and could be used for detailing the model:

- Usage scenarios might use state based information to call different functions or use different parameters. This would allow us to model different amount of communication accesses for a system tick instead of calculating the mean value, leading to a more precise model.
- A (graphical) tool for easy definition of DoublePDM and DoublePDF functions would have helped to add various resource demands based on given load-curves.
- More sensors for easier analysis of for example minimum and maximum response times to detect run-time exceptions.
- Use of interpolated multidimensional look-up tables which are able to return a resource demand based on given parameters, like MatLab[14].

In the future, further influence factors should be incorporated into the simulation as well as refining existing ones. Using a modular system, some parts of the model can easily be exchanged by more precise ones. Also, new hardware modules should be added to the simulation like FPGA or DPM. Finally a full scale evaluation, which is already being prepared, based on an existing application will show the accuracy of the performance prediction.

REFERENCES

[1] J. Happe, "Performance Prediction for Embedded Systems," pp. 1–16, 2005.

[2] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg, "The Palladio Component Model," Karlsruhe, Tech. Rep., 2011.

[3] S. Becker, "Coupled model transformations for qos enabled component-based software design," Ph.D. dissertation, UniversitÃđt Oldenburg, Uhlhornsweg 49-55, 26129 Oldenburg, 2008.

[4] INTERNATIONAL ELECTROTECHNICAL COMMISSION, "IEC 61131-3: Programmable controllers - Part 3: Programming languages," 2003.

[5] Profibus and Profinet International (PI). [Online]. Available: http://www.profibus.com

[6] Profibus Nutzerorganisation e.V. (PNO). [Online]. Available: http://www.interbusclub.com/

[7] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[8] J. Happe, "Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments," Dissertation, University of Oldenburg, Germany, August 2008.

[9] S. Lehrig, "Assessing the quality of model-to-model transformations based on scenarios," Master's thesis, University of Paderborn, Zukunftsmeile 1, October 2012.

[10] Object Management Group. "Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)". [Online]. Available: http://www.omg.org/spec/QVT/

[11] E. M. Project, "Atl (version 3.2.1)." [Online]. Available: http://www.eclipse.org/atl/

[12] A. SchÃijrr, "Specification of graph translators with triple graph grammars," in *in Proc. of the 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG '94), Herrsching (D.* Springer, 1995.

[13] Eclipse.org, "Xtend (version 2.3.1)." [Online]. Available: http://www.eclipse.org/

[14] MathWorks, "Matlab (matrix laboratory) numerical computing environment." [Online]. Available: http://www.mathworks.de/products/matlab/