Automatic Deployment of IEC 61499 Function Blocks onto Interconnected Devices

M.Sc. Jens Frieben, Fraunhofer Project Group Mechatronic Systems Design
Dr. Matthias Tichy, Software Engineering Group, University of Paderborn

Automatic Deployment of IEC 61499 Function Blocks onto Interconnected Devices

Manually deploying IEC 61499 function blocks onto distributed devices can be a time consuming and error prone task for large systems. In this paper we present an approach to automatically compute a feasible deployment. To do this, we add new attributes and restrictions to the IEC system. Based on this information, an automated constraint solver is able to find feasible deployments, which satisfy the given system requirements like end-to-end deadlines. By generating additional routing functionality, this approach can also speed up the development process enormously.

IEC61499, automatic deployment, optimization, linear program

1. Introduction

Modern applications for programmable logic controllers (PLC) have to fulfill a broader range of functionality and more complex tasks than it was necessary a few years ago. The IEC 61499 standard has been developed to simplify PLC programming. It supports reusability, interoperability and offers an object oriented approach for easy extension of existing applications. One way to achieve this extensibility is to split the application onto several devices, for which the standard provides a generic model for distributed systems.

Currently, this mapping of software components, called function blocks, onto devices has to be done manually. End-to-end deadlines or other properties, like communication overhead or bandwidth restrictions, have to be checked via extensive testing or simulation. This is a time consuming and error prone task for the engineer.

In our work, we extended the IEC 61499 standard to allow an automated mapping of function blocks to devices. Additional attributes, like memory consumption, worst-case-execution-time or network delay, are added to the IEC system. Via restrictions, like end-to-end deadlines or maximum bandwidth usage, the engineer is able to ensure system properties according to the requirements. All data is afterwards

converted to a linear program which is executed by a solver to find a feasible deployment. This approach yields three important advantages: First, all function blocks of the application are mapped automatically to devices. Depending on the size and complexity of the application, this can speed up the development and prevent the selection of infeasible solutions, even before testing. Second, is the engineer is able to define end-to-end response times, which are strictly adhered when computing a deployment - an important feature when designing safety-critical systems. Third, a linear program to find feasible solutions also allows the search for optimal solutions regarding given objectives e.g. execution times or communication overhead. Finally, our approach automatically creates the routing functionality that is necessary for distributing communicating function blocks across different network segments. Depending on the used modeling tool, these elements must otherwise be created manually.
To evaluate our concepts, a prototype has been developed which allows the engineer to specify additional attributes and restrictions. It uses a constraint solver to find feasible or optimal deployments from which IEC 61499 compatible mappings and the according communication function blocks for routing purposes are generated.

This paper is structured as follows: In Section 2 we give a short overview of the IEC 61499 standard and it's most important elements. Afterwards, current research topics in the field of automated deployment are presented. In Section 3 we detail the concepts of our approach and explain the additions that are necessary to automatically compute a deployment. In the last two sections we present some implementation details and test results (Section 4) followed by a short conclusion and possible future work (Section 5).

2. Fundamentals and current research

In this section we give a short introduction to the important parts of the IEC 61499 standard which are most relevant to this paper. Afterwards a short overview to the similar or related current approaches is given.

2.1 IEC 61499 Standard for programmable logic controller

The IEC 61499 standard incorporates several new concepts and approaches for programming PLC. One of these new features is its object oriented approach that allows the distribution of functionality onto several devices in a network. The three main parts of new IEC Systems are the applications, the hardware (devices and communication segments) and a mapping as shown in figure 1.

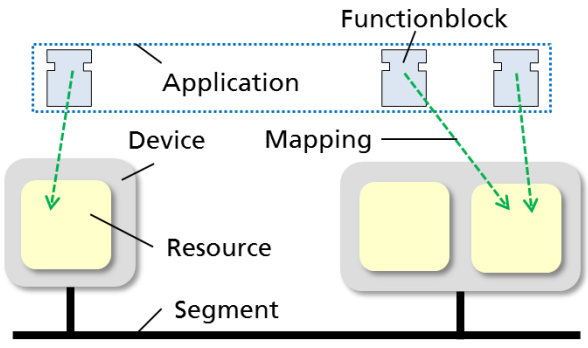

Figure 1 Important parts of the IEC 61499 standard

An application consists of one or many function blocks that send events and data to communicate with each other. A function block (short FB) is an abstract representation of functionality and has to be instantiated for use. Figure 2 shows

an instance of the voter function block named *RedundancyVoter1*. If the event *vote* is triggered, the three data inputs *in1*, *in2*, *in3* are used in the function blocks internal algorithms to compute a value that will be set to the output port *result*. Other function blocks use this data the moment they get the output event *resultReady*. This way, complex applications can be created by interconnecting different function blocks. There are also different types of function blocks, like the Service Interface Function Block (SIFB). The SIFB represents an interface to underlying hardware like actuator or sensors. This abstraction makes it easy to access hardware functionality by just reading the output and sending events to the FB.

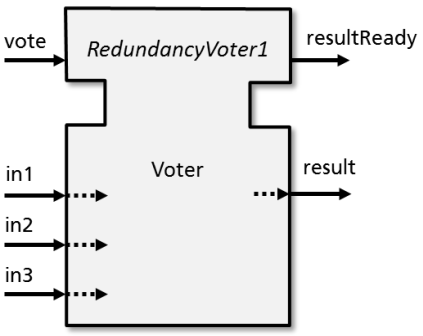

Figure 2 A RedundancyVoter1 instance of the Voter FB

The second part, the hardware of an IEC 61499 system consists of several devices connected by segments and links. A segment represents a communication medium like Ethernet. A link attaches a device to a segment. Each device has resources with runtime environments in which the function blocks are instantiated and executed. They also provide interfaces to the communication systems and to the device specific services.
The third part of an IEC system is the mapping which is used for allocating the function blocks defined in the applications onto the resources in the network. It is mandatory, that every FB is mapped to exactly one resource.

2.2 Related work

The deployment of software components on hardware components is a field of high interest and used in different domains like microprocessor design, multitier application development or web engineering. Different techniques and algorithms based on heuristics or linear programming are used to find a good or best solution, while trying to cope with multiple, contrary objectives (e.g. reliability vs. performance vs. cost).

The work of Tichy, Giese and Schilling [TSG04] for example, deals with the modeling and dynamic distribution of software components. They focus particularly on the availability of individual components and its effect on service oriented system with self-repair capabilities. To compute a feasible and optimized deployment, they use linear programs that are restricted by patterns to ensure fault tolerance.
In [BTD08] a similar approach has been used. A linear program optimizes the two contrary goals performance and reliability of web applications. Routing or reaction times were omitted in their work.
In [AGMM09], the authors used ant colony optimization algorithms to allocate software functions onto different nodes of connected embedded systems. Their flexible approach allowed them to swap to a genetic algorithm, which they afterwards compared against the ACO approach.
Some other approaches which were investigated during the creation of this work, cannot be introduced due to the restricted page count, are [THK07],[MBAG10] and [KK04], which all focus on different deployment techniques and aims, but do not

provide the required features like routing functionality, optimization, deployment constraints or repeatability. Due to this fact, we developed an approach based on linear programming that fulfills the requirements and detail it in the following sections.

3. Concepts

In our work we extended the IEC 61499 standard to allow an automated mapping of function blocks to devices. In this section, a short introduction to the changes and additions are given, as well as details to the deployment process and the advantages this approach offers.

A function block has one or more internal algorithms that are used to fulfill a certain task. Depending on the input and current state of the function block, this task takes some time to be executed and the results propagated to the next FB.
For our work, we had to make some simplifications how the function blocks are executed and how long this will take. The IEC standard does not define the execution semantics of the function block on a resource, so vendors can implement alternative scheduling functions. In this paper, we assume a first in/first out strategy and pessimistically assume that each FB will be executed last. This assumption guarantees that regardless of the execution order, the required constraints on the end-to-end-deadlines hold. In addition to this, the execution time depends in most cases on the given input and algorithm. To work with a fixed value, we require worst case execution times (WCET) for each FB. These two simplifications allow us to handle the tasks addressed in this work more easily.

3.1 Additional attributes

The first step towards an automated deployment of FBs onto resources is to quantify the properties that influence the system. For example resources have limited memory or a specific performance that must be considered in a deployment. To cope with this, we added nine new attributes to different elements of the IEC System which are sufficient for our approach. Function blocks need to specify how much memory they will consume (*memUse*) and how often they will communicate with each other (*comFreq*). The size of the data (*dataSize*) that is send affects the deployment due to possible bandwidth restrictions. A value for the *WCET* (*execTime*) is important to be able to check if the reaction time restrictions can be adhered. The communication segments were augmented by the attributes *delay*, to specify the time needed to transmit the events and data, and the maximum *bandwidth* which can be used to prevent lags in Ethernet based networks by restricting the traffic on these segments. To the resources, three attributes were added. *Performance* to represent the processing power (e.g. 500MHz), *devicetype* to restrict the types of FBs that can be instantiated on this particular hardware and *memCap* as an upper bound for the memory that can be allocated.

3.2 Restrictions and routing

Storing additional information in the model of the IEC system is a prerequisite for an automated deployment approach. However, to restrict all possible distributions of function blocks to resources, more computational restrictions must be set up. An example for such a restriction is the *MaximumMemoryUse* rule which states the simple fact: The sum of all FBs mapped onto a resource must be less or equal the resources memory capacity. A second example is a similar rule that forbids an algorithm to place incompatible FBs on resources based on the *devicetype* attribute. When using SIFB these restrictions are also very important. Due to the fact that SIFB represent kind of access points to the hardware, they must be mapped to a specific resource. These predefined mappings can be enforced by a restriction.

To go one step we added additional concepts to adhere predefined end-to-end deadlines. In many application domains is it necessary to know exactly how long after a triggering event it will take until a certain reaction can be expected. Figure 3 shows an exemplary setting in which a *flowsensor* passes its data to an *input/output compare*. The comparer FB notifies the *pumpcontrol* immediately, if the values do not check out correctly. In that case, the pump control must send a stop-signal to the *pump*. The resulting reaction time can be calculated by adding each FBs WCET, the time to wait for other FBs on the same resource, the communication overhead and the delay on the network. To cope with this problem, we added *executionchains*. The engineer knows the critical path (from the sensor to the pump) and adds the FBs in this order to a list. Afterwards a *maximum reaction time* is set which must not be exceeded by the calculated reaction time.

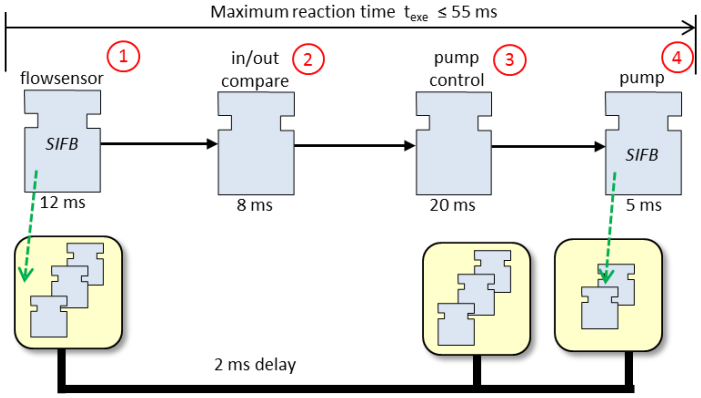

Figure 3 An execution chain of four FBs and it max reaction time

A welcome side effect of computing a feasible deployment of function blocks onto hardware is the automatic generation of routing functionality. Currently most tools offer simple, automated functions that create *CommunicationSIFB* to handle the communication of function blocks between resources and devices. As soon as two communicating FBs are separated by a third segment, the engineer has to create them manually. By doing so, the additional FBs must be again considered in the timing analysis. This situation is shown in figure 4. The devices *router1* and *router2* relay the events and data from the *sender* to the *receiver* over a third segment,

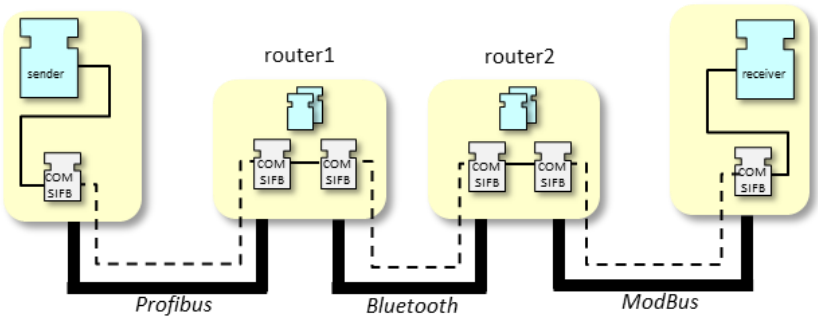

Figure 4 Routing through two segments

using Bluetooth as a medium. This scenario is not far-fetched, due to the recent developments towards smart factories [SMRFCR].
With the attributes and restriction already in use to compute a deployment, only a few constraints had to be added to extend the linear program to support routing. All ComSIFB that are needed for a communication over multiple segments can now be automatically generated.

3.3 Computing the deployment

There are various ways to compute a deployment and up to this point, we developed generalizable concepts that could be used with any of them. The two most common categories are heuristics or linear programming approaches. Heuristics usually use probability functions or randomness in different kinds of degree. Well known heuristic approaches are Ant Colony Optimization [CHS02], Genetic Algorithms [Gol89] and Simulated Annealing [ZK10]. In our work, we aimed towards a fully repeatability technique to calculate a deployment. This led us to the use of a linear program which can read, preprocessed and interpreted by an off-the-shelf constraint solver. As long as the input remains the same, a solver outputs won't differ from previous computations. Therefore, the IEC System, all its elements, attributes and restrictions had to be described in such a linear program before the actual deployment can be computed.

Linear programs are formal, textual descriptions of model and data of a problem that shall be solved. There are many languages for different solvers tailored for specific purposes. In our approach we used the *optimization constraint language* (OPL) of IBM's commercial ILOG CPLEX [IBM] tool, because it is very close to a plain mathematical formalization.
The first step is to convert the IEC system into a format that the solver can process. Every FB, resource, connection between FBs and segments are represented as an ID and its attributes that are needed to compute the deployment.
The next step is to define the constraints that must be adhered. An example of such a constraint is the already introduced *MaximumMemoryUse,* which ensures that the memory capacity of a resource is never exceeded. In our work, every constraint has been specified in a solver independent, mathematical form and in OPL syntax. This allows us to easily create linear programs in different languages or even go back and specify appropriate formulas for the heuristic approaches. First, the *MaximumMemoryUse* constraint is shown in formal syntax. The constraint uses functions to access attributes, like *memUse(f)* to get the memory use of the function block named *f*. The *FB2ResMaps(f,r)* is a true-false variable that is true if FB *f* is mapped onto resource *r*. This way it is easy to sum up the already used memory on

$$\forall\, r \in Res : \sum_{f \in FB} FB2ResMap(f, r) * memUse(f) \leqslant memCap(r)$$

the resource. The constraint written in OPL (listing 1) is quite similar to the mathematical definition. Attributes can be accessed via the dot notation, e.g. the *memCap* value from the resource named *r*. Overall 18 constraints had to be implemented to force a correct deployment.

```
constraints {       /* Constraint:  MaximumMemoryUse */
    forall(r in resources)
        sum(f in functionblocks) FB2ResMap[f,r] * f.memUse <= r.memCap
}
```

Listing 1 MaximumMemoryUse constraint in OPL syntax

3.4 Optimization

An important feature of using a linear program to compute a deployment, is that it provides the possibility to not find just one feasible, but an optimal deployment for a specific objective. In our work we used primarily the *executionChains* to minimize reaction times on predefined end-to-end deadlines or to minimize the bandwidth on segments. In a linear program only one objective function can be defined, but there are often ways to combine different goals into one formula. These formulas are quite similar to the already introduced restrictions, but are supplemented by a *minimize* or *maximize* statement. Listing 2 shows the objective function for minimizing the reaction times. The sum adds the values for the different chain times consisting of reaction times, delays and the ComSIFBs. With the use of the minimize statement, the solver tries to find the smallest value of this sum.

```
minimize
    sum(c in chains )
          (sum ( r in resources )
            (resInChain [r,c]∗sum(f in functionblocks) FB2ResMap[ f,r]∗ f.execTime)
       + (sum(l1 in functionblockLinks , rescon in resourceConnections ) linkInChain [l1,c]
                                                ∗ FBLink2ConMap[l1,rescon] ∗ rescon.delay )
       + (sum(l2 in functionblockLinks , r2 in resources) linkInChain [l2,c]
                                                ∗ router [l2,r2] ∗ sumResExecTime [r2]) ) ;
```

Listing 2: Objective function to minimize the reaction time of execution chains

Minimizing the reaction time is useful to reduce for example the cycle time of processing steps, which can lead to improved throughput in a factory. A major drawback of using an objective function is the increased computation time. Despite many well developed techniques to solve linear programs, searching for an optimal solution is still a very time consuming task and might take up multiple times longer than any other feasible solution.
There are various optimization goals that can be implemented. It is for example possible to bypass unreliable communication media or in some cases, a load balancing might be required to use low-performance devices which are only battery powered and have to safe energy.

4. Evaluation

To evaluate the concepts presented in this work, a prototype has been implemented and used to compute deployments. The goal was to model an IEC system, define the additional attributes, restrictions, optimizations and calculate an optimized deployment that fulfills all requirements and restrictions.
The IEC System including all its elements like FBs, resources and segments were created and modeled with a commercial tool. As defined by the standard, all information must be stored as xml files, which could be read and preprocessed by our tool. As a result, we are able to generate a formal, machine readable model of the system. During this process, several shortcomings of the current IEC 61499 standard had to be dealt with like redundant information or identical names for type definitions.

We created a prototype that allows the import of the system, set the values for the new attributes and create the chains in a graphical editor. Afterwards, the linear program is generated and executed by the solver. The resulting output is analyzed and the function blocks for the routing are added to the system.

Based on this prototype, we executed a series of tests to gather information about the runtime of the solver. To get representative data, we randomly generated IEC models in multiple versions and specific setups. These setups included, among other factors, different numbers of function blocks or resources, varying sizes of chains and with or without an objective function. Afterwards, we measured the time it took the solver to compute a deployment. All tests showed an exponential runtime depending on the size of the model. First, simple scaling tests were run, in which no chains, routing or optimizations were set up. Scaling only one factor, function blocks or resources, resulted in faster computations in comparison to setups in which both factors were scaled simultaneously. This showed how the complexity of the input model influenced the overall computation time. Adding more communication segments to increase the routing complexity resulted only in a minor raise in the runtime of the solver. The use of execution chains and an objective function had a much more severe impact. The tests showed that adding multiple execution chains to the linear program resulted in twice as long computation times. The determination of an optimal solution is by far the most time-consuming setup and usually takes three to five times longer to compute than a model with routing, multiple segments and execution chains.

The prototype proves that the concepts we developed can be applied and used to deploy function blocks onto resources. However, there still a great potential to of performance improvements to speed up the computation process which are partially addressed in the next section.

5. Conclusion and future work

Manually mapping function blocks onto resources can be a time consuming and error prone task. Main reasons for this are usually the sheer amount of elements that have to be managed or the restrictions implied by the requirements. In our work we showed an approach to augment an existing IEC system with new attributes and restrictions. Based on this information, a constraint solver is used to create feasible deployments, which satisfy the given requirements like end-to-end deadlines. In addition to this we are able to find optimal mappings with respect to a specified optimization goal. Due to the preprocessing and formalization of the IEC system we are still flexible enough to later on exchange the linear program by a heuristic approach. Finally, an automated routing functionality has been added by making simple additions to the restrictions used by the solver. The concepts we showed help to make another step towards a more user friendly and computer aided development environment for IEC 61499 systems.

But there are still some issues left open to investigate. For example must the simplifications that had been made to better cope with the WCET be analyzed in more detail. Timing analysis in general is an interesting topic that several research groups are currently investigating. Some problems caused by the non-formal or

incomplete definition of the IEC 61499 standard complicate the creation of tools or the external processing of the IEC systems. Workarounds are used vendor specific and therefore cancel out the aim of an independent programming standard.
The exponential runtime of the solver is still a problem for large systems. Here, more techniques should be incorporated to speed up the process. Possible solutions to handle the execution time problems might be heuristic approaches, to hierarchically structure the system or to use partial mapping techniques in combination with clustering approaches.

References

[AGMM09] Aldeida Aleti, Lars Grunske, Indika Meedeniya, and Irene Moser. Let the ants deploy your software - an aco based deployment optimisation strategy. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE '09, pages 505–509, Washington, DC, USA, 2009. IEEE Computer Society.

[BTD08] Bas Boone, Filip De Turck, and Bart Dhoedt. Automated deployment of distributed software components with fault tolerance guarantees. In Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications, pages 21–27, Washington, DC, USA, 2008. IEEE Computer Society.

[CHS02] Oscar Cordon, Francisco Herrera, and Thomas Stützle. A review on the ant colony optimization metaheuristic: Basis, models and new trends. Mathware & Soft Computing, 9:141–175, 2002.

[Gol89] David E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[IBM] IBM ILOG CPLEX Optimization Studio, http://ibm.com/software/products/

[KK04] Tatiana Kichkaylo and Vijay Karamcheti. Optimal resource-aware deployment planning for component-based distributed applications. In Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, pages 150–159, Washington, DC, USA, 2004. IEEE Computer Society.

[MBAG10] Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Architecture-driven reliability and energy optimization for complex embedded systems. In George Heineman, Jan Kofron, and Frantisek Plasil, editors, Research into Practice – Reality and Gaps, volume 6093 of LCNS, pages 52–67. Springer Berlin / Heidelberg, 2010.

[SMRFCR] SmartFactory KL e.V. http://www.smartfactory-kl.de/

[TSG04] M. Tichy, D. Schilling, and H. Giese. Design of self-managing dependable systems with uml and fault tolerance patterns. In Proc. of the Workshop on Self-Managed Systems (WOSS) 2004, FSE 2004 Workshop, Newport Beach, USA, 2004.

[THK07] Chouki Tibermacine, Didier Hoareau, and Reda Kadri. Enforcing architecture and deployment constraints of distributed component-based software. In Matthew Dwyer and Antónia Lopes, editors, Fundamental Approaches to Software Engineering, volume 4422 of LCNS

[ZK10] Albert Y. Zomaya and Rick Kazman. Algorithms and theory of computation handbook. chapter Simulated annealing techniques,pages 33–33. Chapman & Hall/CRC, 2010