Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2018) 000–000

www.elsevier.com/locate/procedia

51st CIRP Conference on Manufacturing Systems

Component based Verification of Distributed Automation Systems based on Model Composition

"Andreas Zeller, Michael Weyrich" *

"University of Stuttgart, Institute of Industrial Automation and Software Engineering, Germany"

* Corresponding author. Tel.: +49-711-685-67291 ; fax: +49-711-685-67302. *E-mail address:* andreas.zeller@ias.uni-stuttgart.de

Abstract

Challenges on safeguarding distributed automation systems arise due to their increasing complexity and changeability. Functional changes in automation systems are mainly conducted by software modifications. Especially in distributed automation systems, the impacts of software modifications are difficult to estimate. If behaviour models of the automation systems are available, model-based techniques are suitable to estimate the impacts of software modifications on other system components. In fact, behaviour models of distributed automation systems are seldom available or maintained, due to the high complexity of the overall system and the changing structure caused by reconfigurations or software modifications. This often prevents the application of model-based techniques.
This contribution presents a model-based approach with which the impacts of software modifications can be recognized and affected subsystems can be safeguarded efficiently by model-based verification methods. To achieve this an impact analysis is performed, identifying requirements which are affected by software modifications. As the behaviour models that are necessary to verify the identified requirements are seldom available, the necessary models are generated automatically.

Keywords: "Modelling, Software Modification, Verification, Smart Factory"

1. Introduction

Usually the life cycle of automated systems are longer than the innovation cycles of system functionalities, which are often realized by software. This leads to software modifications that are increasingly relevant during the operation time of automated systems. Software modifications, realized by software updates, are already a relevant topic for consumer products. In the context of the industrial internet of things it’s very likely that software modifications at runtime will also gain in importance for industrial automation systems to increase their flexibility [1].

To ensure the necessary flexibility, industrial automation systems control is increasingly decentralized [2] [3]. Service oriented architectures can be a suitable approach to coordinate distributed automation systems due their modular design and the potential for scalability, reusability and ad-hoc networking. Ad-hoc networking requires semantic described interfaces of the services. The services are software components which coordinate decentralized a control task. This eases software modifications of the systems, by adapting the control code of a component or integrate new components into the existing automation system.

These easily modifiable distributed systems also create substantial challenges to safeguard the impacts of software modifications on other components [4]. That is due to the fact that ad-hoc networking and software modifications can lead to changing dependencies between the system components and that dependencies of decentralized systems are harder to determine than those of centralized systems. This leads to difficulty in estimating impacts of software modifications. This poses a significant barrier, in the field of industrial automation, due to high safety requirements.

Model based techniques can support the estimation of the impacts of software modifications and safeguard the subsystem which is affecting by this. The application of model based techniques often fails due the unavailability of suitable formal behaviour models of the automation system. This is firstly due to the fact that models of automation systems are becoming increasingly complex. Furthermore, for each functional modification of the automation system, the behaviour model of the system has to be adapted. Often the operators of an automation system are not capable of maintaining the overall behaviour model.

To support the safeguarding process of distributed automation systems there is a need for systematic test processes and suitable modelling techniques. Some of them are presented in the following.

2. State of the Art

Modifications to automation systems can be distinguished in four categories [5]. By this differentiation, it can be decided which parts of a simulation (context, platform, simulation of the technical process) must be adapted after modifications. According to this, when software modifications occur, just the software has to be changed. The design of the other parts isn't affected.

To meet the challenges for testing automation system there are several approaches in the field of model based testing and test automation. The research project "DoMain" focuses on software evolution of automation systems. In the framework of this project, [6] describes how software evolution can be safeguarded by verifying interface models under consideration of hardware, software and electric. Additionally a concept is presented to safeguard the evolution of variant-rich by incremental model checking [7]. Within the scope of another project "FOCUS" a methodology has been presented how distributed, reactive automation system can be developed on a formal way [8] [9]. This enables the usage of verification methods in different development phases on different granularity levels. An approach how to automate the identification and classification of software modifications is presented in [10] on the basis of Programmable Logic Controller.

Model based verification techniques can be used to examine if a test object meets its requirements. With these techniques a functional behaviour model, that e.g. contains the control logic of an automation component, can be verified against formalized requirements with help of mathematic methods. Thus model based verification constitutes a method of static testing. As a result the real system is not executed. This enables the testing of the behaviour of a system before going into operation and decreases the risk of hazards when commissioning.

With some formal verification methods like the model checking the state space explosion poses challenges, especially for system with a high degree of parallelism. There are several approaches how to handle these challenges [11]. This shall not be regarded further in the following.

For the success of verification methods the choice of a suitable modelling technique plays an important role. For this reason, several modelling techniques for control logic have been developed in the field of automation (e.g. NCES [12], Modellica, UPPAAL [13], SIPN) as well as internet technologies (e.g. open nets [14]).

These modelling techniques which are suitable for verification methods are listed and compared in table 1 by criteria which are relevant for the concept presented.

Modelling Technique	Environ-ment	Parallelism	Suitability for modular systems	Specialism – Modelling of Logic	Time Behaviour
Modellica	No	Yes	Yes	Low	Yes
UPPAAL	Yes	No	No	Medium	Yes
NCES	No	Yes	Yes	Medium	Yes
Open Nets	No	Yes	Yes	High	Yes
SIPN	No	Yes	Nein	High	Yes

Table 1: Comparison of modelling techniques and environments that are suitable for model based verification

3. Concept of component based verification

The goal of the concept is to efficiently safeguard distributed automation systems after software modifications. For this a customized input for a verification tool is generated. The tailored input contains the requirements which are affected by the software modifications as well as the behaviour models that are needed to verify the affected requirements. The three steps of the concept structure is pictured in Figure 1. The puzzle pieces illustrate control services. The control services are software components of the distributed automation system. In the following just called components.

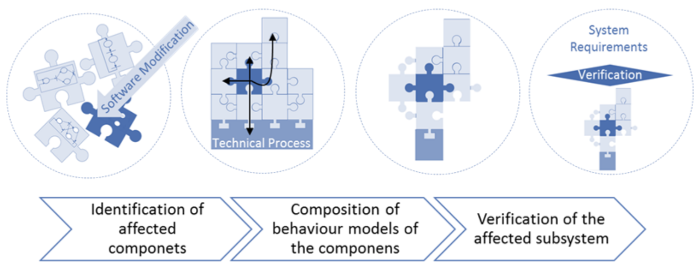
Software Modification
Technical Process
System Requirements
Verification
Identification of affected componets
Composition of behaviour models of the componens
Verification of the affected subsystem

Figure 1: Overview of the concept of component based verification

Within the first step an impact analysis is done. Originating from the component which was modified, the system requirements of the components affected are identified, by dissolving dependencies between the model interfaces. Within the second step, the behaviour models of the components which are necessary to verify the affected requirements, are composed to a single behaviour model. Within the third step, the requirements as well as the corresponding behaviour models are verified by a model checking tool.

From the figure 1 it becomes evident, that modularization is an essential part of the concept. In the following the **modularization approach** and the requirements on the modelling technique for the behaviour models are described and a **modelling technique** that meets the requirements is presented. This is the basis for **the impact analysis of software**

modifications as well as the composition of behaviour models that is subsequently presented.

3.1. Modularization Approach

Behaviour models of automation system can become complex very quickly. Modularization is a suitable approach to reduce this complexity. Every automation component represents a module that contains an outer and an inner view. The inner view describes the behaviour of the component and the outer view describes the interaction between the components of the automation system. These views are just connected via the interfaces of the components. This allows an encapsulated view on the behaviour of a component.

The outer view of a component is pictured in figure 2. It is defined by the requirements which must be met by the component by, its software interfaces as well as its interfaces to the technical process. Due to a semantic interface description, the mapping between the components can be done by the interface description.

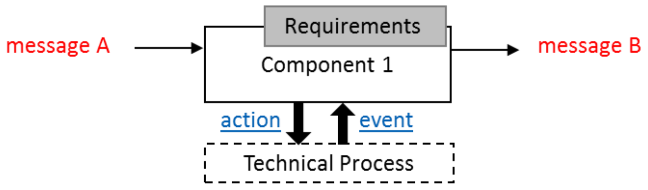

Figure 2: Outer view of a component

Every component contains the requirements, which have to be fulfilled to ensure its functionality. The requirements do not only depend on the behaviour of this component but also to the behaviour of components on which the component is dependent. The dependencies result by service calls. If a component calls the services of other components it depends on them. Hence similar to the call hierarchy between the components a hierarchical structure between the requirements exist. Due to the property of services the component is not depending on the components by which its functionality is called. The requirements can be modelled in a formal language that is suitable as input for a verification tool, such as CTL (Computation Tree Logic) or LTL (Linear temporal logic). The dependencies between components and requirements will be described within the impact analysis chapter in greater detail.

3.2. Modelling the Behaviour of a Component

As described in the state of the art, there are several modelling techniques that are suitable to model the behaviour of automation system. The concept poses additional requirements which must be fulfilled by the modelling technique: representation of parallelism, representation of interfaces for asynchronous communication as well as interfaces to the technical process, modularity, composability. Next to these properties, the model must be formal and widely used, such that it is a suitable input for existing verification tools.

The inner view describes the behaviour of a component. Open-Petri Nets are very suitable to model the behaviour due to their suitability for event-controlled systems and their easy representation of parallel processes and the wide usage of petri-net similar models in the field of automation. The open nets are extended by attributes from signal interpreted petri nets (SIPNs) to enable the representation of interface to and from the technical process. In the following the attributes of the open nets are described as the composition of the components to a behaviour model of the overall system. The open net is described by the following 9-tuple:

$$N =< P, T, F, M_0, \Omega, I, A, S, L, >$$

- ***P***: finite set of spaces.
- ***T***: finite set of transitions.
- ***F***: finite set of of flow relations between transitions and places and reverse$F \subseteq (P \times T) \cup (T \times P)$.
    - set of predecessor places of transitions described via $\cdot t = \{p|(p,t) \in F\}$
    - set of successor places of transitions described via $t \cdot= \{p|(t,p) \in F\}$
    - set of predecessor transitions of places described via$\cdot p = \{t|(t,p) \in F\}$
    - set of successor places of transitions described via $p \cdot= \{t|(p,t) \in F\}$.
- $\boldsymbol{M_0}$: initial marking, represented via a vector with the cardinality|P|.
- **Ω**: finite set of markings where N is allowed to terminate.
- ***I***: interface places $I \subseteq P$. Places to exchange message with other open nets. The description of the places corresponds with the messages being exchanged. For minimal open nets apply:
    - input places $I1$ which receive messages don't have predecessor transitions: $\cdot p = \emptyset$.
    - output places $I2$ which send messages don't have predecessor transitions: $p \cdot= \emptyset$.
    - an input place can't also be an output place and reverse: $I1 \cap I2 = \emptyset$.
- **A**: interface to the technical process. Actions which are executed to the technical process when a place is occupied. A vector with the cardinality: $|A| \equiv |P|$.
- **S**: interface from the technical process. Events from the technical process trigger transitions. A vector with the cardinality: $|S| \equiv |T|$.
- ***L***: Latency of the arcs. A vector with the cardinality: $|L| \equiv |F|$.

A generic graphical representation of the open net is illustrated in figure 3.

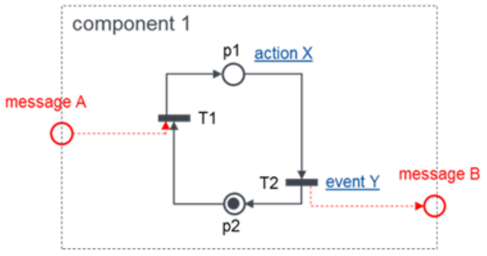

Figure 3: Generic illustration of an open net.

The interface places are marked red on the system boundaries. The functionality offered by component 1 can be requested with the service call “message A”. “Message B” illustrates a service call to a component whose functionality is requested by component 1. Similar to SIPN the interfaces to the technical process are mapped to the places (action X) and the interfaces from the technical process are mapped to the transitions (event Y).

3.3. Composing Behaviour Models

The descriptions of the interface places correspond with the interface descriptions of the outer view. If open petri nets of several related component are available, a common open net of the system can be composed by following calculation rules:

$P_{ges} = P_{Komp.1} \cup P_{Komp.2} \cup \dots \cup P_{Komp.n}$
$T_{ges} = T_{Komp.1} \cup T_{Komp.2} \cup \dots \cup T_{Komp.n}$
$F(s,t) = F_{Komp.1}(s,t) \oplus F_{Komp.2}(s,t) \oplus \dots \oplus F_{Komp.n}(s,t)$
$F_{ges}(t,s) = F_{Komp.1}(t,s) \oplus F_{Komp.2}(t,s) \oplus \dots \oplus F_{Komp.n}(t,s)$
$M_{0,ges} = M_{0,Komp.1} \cup M_{0,Komp.2} \cup \dots \cup M_{0,Komp.n}$
$\Omega_{ges} = \Omega_{Komp.1} \cup \Omega_{Komp.2} \cup \dots \cup \Omega_{Komp.n}$
$I_{ges} = I_{Komp.1} \cup I_{Komp.2} \cup \dots \cup I_{Komp.n}$
$A_{ges} = A_{Komp.1} \cup A_{Komp.2} \cup \dots \cup A_{Komp.n}$
$S_{ges} = S_{Komp.1} \cup S_{Komp.2} \cup \dots \cup S_{Komp.n}$
$L_{ges} = L_{Komp.1} \cup L_{Komp.2} \cup \dots \cup L_{Komp.n}$

The graphical representation of the composition is illustrated in figure 4.
When the composed petri-net is available, it is possible to verify it against its system requirements. As automation systems can become very big and many parallelisms and system requirements exist, it is very CPU-intensive to verify the overall system. As not to need to safeguard the overall system in the event of software modifications, and impact analysis can be useful. An impact analysis approach that is suitable for distributed automation system will be introduced in the next chapter.

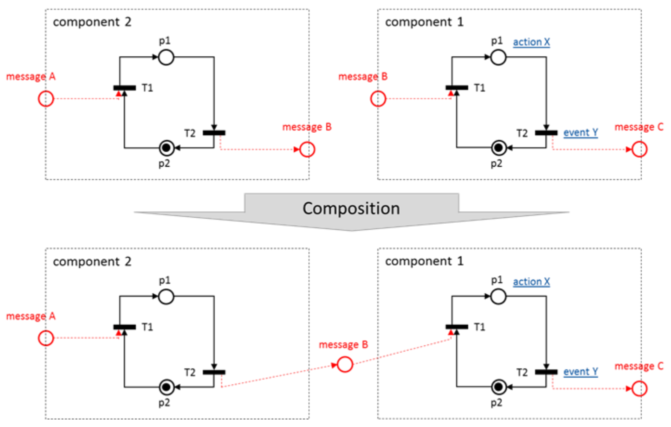

Figure 4: Graphical representation of the composition

3.4. Impact Analysis of Software Modifications

To analyse the impacts of software modifications, the dependencies within the systems must be known. With increasing size of the net, it is hard to recognize the dependencies within an open net, because of its meshed structure. In contrast structure diagrams are very suitable to visualize dependencies between components. To improve the comprehensibility, the call relations between the components are extracted and a structure diagram is generated. For this the structure diagram “block definition diagram” has been chosen due to its standardization in UML and its high degree of recognition.

The block definition diagram of a small example is illustrated in figure 5. The blocks correspond with the components of the outer view. The requirements which must be met by the components are mapped on the respective block. In this example component 2 is dependent on component 1 as well as component 3. Component 2 could be a control and component 1 and 3 sensors or actuators. The impact of software modifications of a component can be analysed by following the arrows backwards from the component which was modified. As the functionality of these components depend on the functionality of the modified component and as a result the validity of their requirements cannot be ensured anymore and must be verified again. For example when component 1 is modified, the requirements of component 1 and 2 have to be secured.
For the affected requirements of each component a customized open petri net must be composed. Because the requirements of a component relate not only to this component but also to all components on which the component depends, the petri nets of all components that are accessible in the direction of the arrows must be considered. In the example of figure 5, to verify the requirements of component 2, an open petri net must be composed by the open petri nets of component 1, 2 and 3 as well as the behaviour model of the technical process. The technical process which represents the physical dependencies between actuator and sensor can also be modelled with open nets and composed with the same calculation rules. The composed open petri nets as well as their requirements are the tailored inputs for the model checker which delivers statements if the requirements are fulfilled by the models.

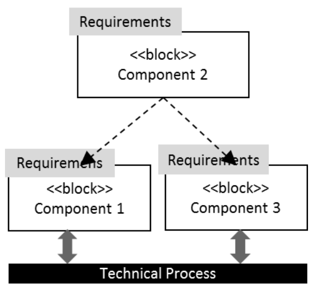

Figure 5: Structure of a block definition diagram

4. Realization and Evaluation of the Conception

The concept has been realized with the test component "TestIAS". TestIAS can be ad-hoc integrated into a distributed automation system. It scans the automation system for software modifications. When a software modification is detected, TestIAS safeguards the affected subsystem as described within the conception. A model pool administers the models of the components as well as their system requirements. The build-up, consisting of the distributed automation system, TestIAS as well as a model pool, is illustrated in figure 6.

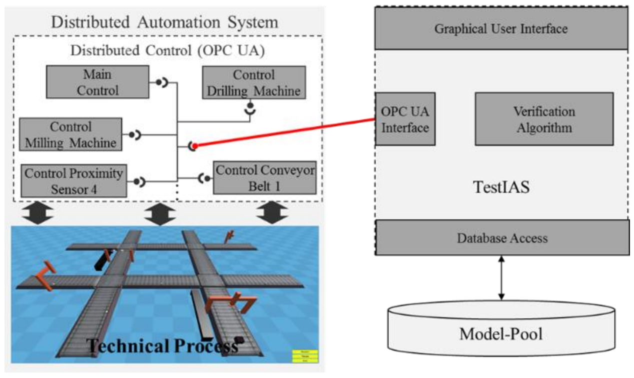

Figure 6: Structure of the realized demonstrator

The distributed automation system meets important requirements of future automation systems like ad-hoc networking, configurability, reusability and scalability.

An OPC-UA based control network interacts with a technical process. The control network consists of 122 services / components running on 6 controllers. The services contain the control functionality and coordinate the production process. Five services are illustrated as examples within the distributed control in figure 6. A configuration interface allows modification of services at run-time. The technical process that represents a discrete production is realized by a game engine. This virtual reality simulation allows the technical process to scale with ease. A configuration interface allows modification of services at run-time.

TestIAS contains a graphical user interface (GUI), an OPC-UA Client, an interface to the model pool database as well as an algorithm that implements the concept. After starting the safeguarding process via the GUI, the OPC-UA Client scans the distributed automation system for software modifications. This is done by scanning which components are available and comparing their version with the version of a previous scan. When a modification is detected, the models of components of the automation systems as well as their requirements are obtained from the model pool. Using the interface of the components an impact analysis is executed, the tailored input for the verification tools is generated and a model checker (ITS-Tool) executed. The verification results are displayed on the GUI.

The successful applicability of the conception has been evaluated by five software modification scenarios. Different types of defects could be allocated, caused by errors in time behaviour or by errors in the logical behaviour.

5. Conclusion

By means of the concept it was demonstrated how software modifications can be safeguarded on an efficient way. The requirements of affected components could be identified by an impact analysis. It was shown how the behaviour models that are necessary to verify the requirements can be composed automatically. For this a suitable modelling technique as well as calculation rules for the composition have been presented.

The applicability of the concept has been evaluated with the help of a test component that comprises the concept. This executes the concept described automatically when software modifications are detected and returns the verification results. The software modifications have been carried out on a distributed automation system that consists of 122 control services. All five tested software modifications, including logical and temporal behaviour, could be verified correctly by the test component.

References

[1] Vogel-Heuser Birgit, Fay Alexander, Schaefer Ina, Tichy Matthias. Evolution of software in automated production systems: Challenges and research directions, The Journal of Systems and Software 110, 2015.

[2] Forschungsunion, Acatech. Recommendations for implementing the strategic initiative INDUSTRIE 4.0, 2013.

[3] Fay Alexander, Vogel-Heuser Birgit, et al. Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns, The Journal of Systems and Software 101 (2015) 221-235.

[4] Zeller Andreas, Weyrich Michael. Challenges for Functional Testing of reconfigurable Production Systems, 21st IEEE International Conference on Emerging Technologies and Factory Automation ETFA, 2016.

[5] Legat Christoph, Steden Frank, Feldmann Stefan, Weyrich Michael, Vogel-Heuser Birgit. Co-Evolution and Reuse of Automation Control and Simulation Software, IECON 40th Annual Conference of the IEEE, 2014.

[6] Legat Christoph, Mund Jakob, Campetelli Alarico, Hackenberg Georg et al. Interface Behavior Modeling for Automatic Verification of Industrial Automation Systems' Functional Conformance, Automatisierungstechnik (at), 62(11):815—825, 2015.

[7] Lochau Malte, Mennicke Stephan, Baller Hauke, Ribbeck Lars. Incremental model checking of delta-oriented software product lines, Journal of Logical and Algebraic Methods in Programming 85, 2016.

[8] Broy Manfred, Fox Jorge et al.Service-oriented Modeling of CoCoME with Focus and AutoFocus, The Common Component Modeling Example, Springer, 2008.

[9] Spichkova Maria. Focus on Isabelle: From specification to verification, Technical Report Department of Electrical and Computer Engineering, Concordia University, 2008.

[10] Ulewicz Sebastian, Schütz Daniel, Vogel-Heuser Birgit. Software Changes in Factory Automation, IECON 2014-40th Annual Conference of the IEEE, 2014.

[11] Schlich Bastian, Brauer Jörg, Wernerus Jörg, Kowalewski Stefan. Direct Model Checking of PLC Programs in IL, 2nd IFAC Workshop on Dependable Control of Discrete Systems DCDS'09.

[12]Khalgui Mohamed. NCES-based modelling and CTL-based verification of reconfigurable embedded control systems, Computers in Industry 61, 2010.

[13] Behrmann Gerd, David Alexandre, Larsen Kim. A Tutorial on UPPAAL, in: Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science , Springer-Verlag 2004.

[14] van der Aaslst Wil, Lohmann Niels, Massuthe Peter, Stahl Christian, Wolf Karsten. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. The Computer Journal 53 (1)90-106, 2010