Deploying New Functionality to Manufacturing Resources Safely at Runtime

Julius Pfrommer, Miriam Schleipen
Fraunhofer IOSB
Fraunhoferstraße 1,
76131 Karlsruhe, Germany

Selma Azaiez, Michael Boc,
Loïc Cudennec, Selma Kchir,
Thibaud Tortech
CEA, LIST, Saclay, France

Xenia Klinge
DFKI Projektbüro Berlin
Alt-Moabit 91c
10559 Berlin, Germany

***Abstract*—Automated manufacturing systems are increasingly required to be flexible in order to support an increasing number of products and product variations, as well as shortening product life cycles. However, even though many manufacturing resources are multi-purpose, they are integrated in an automation infrastructure that may require significant effort to adapt. In this contribution, we show how new functionality can be deployed on manufacturing resources at runtime. For this, we make use of model verification for code analysis, Software-Defined Networking (SDN) to establish new communication pathways for collaboration, and a novel OPC UA-based scripting environment that supports runtime changes to information models and underlying functionalities.**

I. Introduction

The increasing number of product variants leads to multi-purpose production resources which could be used for more for different production scenarios [1]. However, much of the flexibility on the level of individual resources is lost in automated systems where the behavior and interaction between resources is hard coded.

In this contribution, make use of flexible object-oriented information models based on OPC UA to deploy new functionality in manufacturing settings at runtime of the system. In order to provide the required flexibility, we integrated OPC UA with the Lua scripting language. The safety of the deployed code artifacts is ensured by static code analysis. Furthermore, we use Software-Defined Networking to establish new communication pathways if the new functionality requires assets on the shopfloor to interact.

This paper is structured as follows. In Section II, we discuss the background of our work as well as related results from recent years. We then introduce the flexible manufacturing use case that motivated our work in Section III. The main body of the paper then describes the architecture and implementation of the verification and deployment infrastructure in Section IV and Section V. The paper concludes with a summary in Section VI.

II. Background and Related Work

A. Runtime Deployment of Functionality in Automation

Many software systems can change their behavior at runtime, ranging from mere configuration to scripting, loading of shared libraries, and finally more exotic approaches for Dynamic Software Updating (DSU) [2]. In automation, the differences between configuration, programming and scripting are often blurry, e.g., with operators loading G-code for numerical-control milling machines between part runs. The authors of [3] give an overview of the use of downtimeless system evolution of control applications based on IEC 61499. Vogel-Heuser et al. [4] discuss the evolution of software in automated production systems in general and touch upon runtime adaptation. In this work we however only consider the addition of new functionality, which avoids many problems that have to be addressed otherwise [5].

B. Software and Service Verification

The evolution of infrastructures towards more agile softwares where applications can be dynamically downloaded and deployed raise major issues related to the correctness and the safety of the newly deployed application as well as the correctness and the safety of the orchestration of services provided by applications [6]. Verification of software and services addresses these issues.

In a black box verification approach, only interfaces are visible and testable. In a white box approach, binaries or even source code are also accessible. Bozkurt et al. [7] give an overview on testing solutions in service-centric frameworks. Some of them are based on formal verification, i.e. based on model checking. Model-based test case generation has been proposed for the integration of services based on orchestration with BPEL [8], [9], [10]. A symbolic execution is also used. A service is executed symbolically using a set of possible input values for testing service composition using Web Service Choregraphy Description Language [11] or BPEL processes [12]. Model checking approaches were also used for BPEL testing and transformation [13], for verifying timed properties on service interoperabily [14], for verifying multibusiness interactions [15] or for verification using Petri nets for testing BPEL [16], [17].

C. Software Defined Networking

Software Defined Networking (SDN) aims at simplifying and securing the creation of new communication pathways via "programmable networks" [18]. In contrast to traditional network management solutions, SDN differentiates between the data-plane layer where network hardware is interacting, and the control-plane layer used for management of software

defined networks that overlay the physical connectivity. So-called south-bound protocols bridge the control- and data-plane. SDN is commonly associated with the OpenFlow south-bound protocol [19]. However, many vendors have developed custom extensions and protocols in recent years.

In this work, we make use of the NEON SDN protocol developed at CEA LIST [20]. NEON enables the fast configuration of new devices and services deployment within seconds in initially unconfigured infrastructure contexts, while remaining compatible with other configuration and management south-bound protocols such as OpenFlow.

Note that SDN is different, albeit related, to autoconfiguration of network connections and discovery mechansims. See for example [21] for such applications in a manufacturing setting.

III. Use Case Safe and Flexible Human-Machine Interaction

The use case considered in this work consists of a production cell containing a robotic manipulator and a camera (see Figure 1 on the left). The robotic manipulator, called Sybot, was designed for human-machine interactions without safety barriers. The Sybot has 2 main modes of operation: a learning mode where it follows the lead of an operator to learn how to perform the task. In this mode, the robot records all the information about the trajectory that must be followed and the actions that must be performed. In the second mode, the Sybot repetitively reproduces the learned actions and trajectories.

Most importantly, the Sybot has to ensure human safety. When the Sybot detects a contact collision while in the second mode (automatic mode), it stops the current trajectory and waits for directives from the human operator. However, this safety guard may not be sufficient, e.g. when the robot uses a sharp tool where a contact may be detected too late. Our goal within this use case is to ensure additional safety properties in the production cell by using the camera to create a safety zone around the Sybot.

The Sybot can be quickly moved between cells and adapted to a new task. This flexibility should not be lost by tightly integrating the Sybot with the camera. Instead, we want the possibility to connect the camera and the Sybot wirelessly and to deploy the following functional behavior at runtime: The camera streams a live video-feed to an algorithm that was configured to detect humans in the defined safety area. Whenever an intrusion is detected, the fail-safe of the Sybot is tripped. Static code analysis and Software-Defined Networking is used to ensure the functional correctness.

IV. System Architecture

The main components of the architecture are show in Figure 1. In the following, we describe their use as well as interaction patterns based on the use case scenario.

The scenario is described in RobotML (Robot Modeling Language) [22], a Domain Specific Modelling Language (DSML) from the robotics context. The OPC UA-based orchestration initially is agnostic to any specific scenario until the RobotML description of the production scenario is loaded.

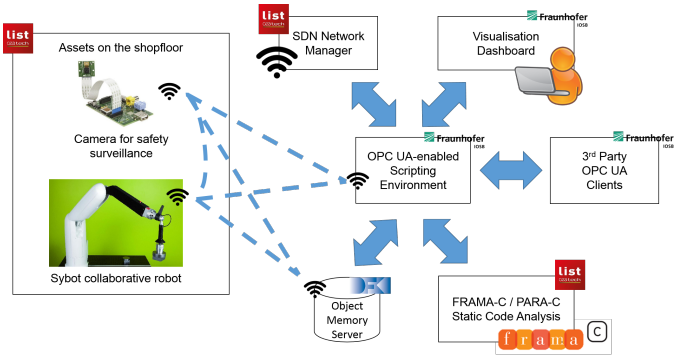

Fig. 1. Systems architecture for the runtime deployment of functionality. Dashed lines denote wireless connectivity based on the NEON SDN framework. Blue arrows denote either communication via OPC UA or local inter-process communication.

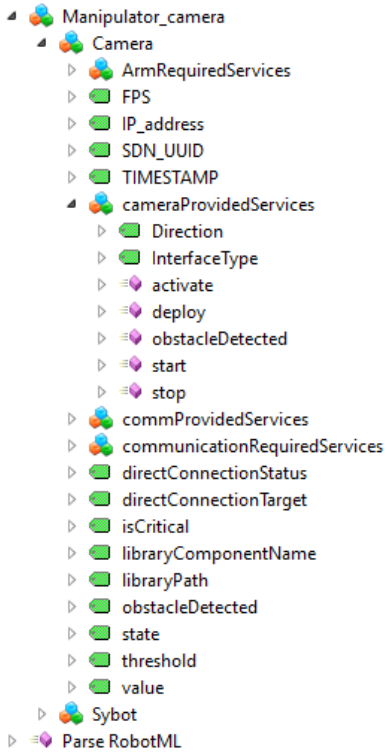

Fig. 2. The OPC UA interface of the newly loaded functionality. Also the structure of the interface, such as object types and method types, are loaded dynamically at runtime.

Furthermore, a description of possible new functionalities and reconfigurations of the production cell are loaded using the same description language. It also indicates the parts of critical applications that need to be verified before their deployment.

The current scenario, as well as the loadable functionalities are reflected in the OPC UA information model of the integration server (see Figure 2). Note that many aspects of an OPC UA information mode, such as object types and method types, are not hard-coded but all generated from the configuration that is supplied at runtime. When the deployment or change of functionality is requested, the following steps are

performed:

1) Verify code-fragments that were marked as critical with a static code analysis tool.
2) Establish eventually required communication path ways between components via the SDN infrastructure.
3) Copy (and compile) the program source code on the target device.
4) Adapt the information model in the Object-Memory Server that some components use to exchange runtime data.
5) Trigger the execution / integration of the new functionality on the target devices.

V. Technical Implementation

A. OPC UA Scripting Environment

The OPC Unified Architecture (OPC UA) [23] is an industrial machine to machine communication protocol based on a platform independent service-oriented architecture. The development of complex technical solutions based on OPC UA today often requires many iterations during development. In order to support faster iterations, it is necessary to minimize the friction induced by the integrating technology. In order to do so in manufacturing environments, Fraunhofer IOSB integrated an open source OPC UA SDK [24] written in C99 into the Lua programming language [25]. Given its nature, and in reference to the infamous VBScript Excel Macros driving many office-floor processes, the resulting scripting environment has been named "uascript". The idea behind this is to avoid hard-coding behavior that is not part of a core stack. This simplified development and allows easy loading of new scripted functionality at runtime.

Lua is famed for being a very compact and easy to learn language. It is notably used for scripting in C++-based game engines with soft-realtime requirements. Furthermore, many recent applications from embedded and Internet-of-Things contexts chose Lua due to its speed and small memory footprint. The integration of an OPC UA SDK with Lua brings the following advantages:

- Easy access to native OPC UA data types: OPC UA defines a type system to specify the data structures that are serialized into binary messages. In uascript, UA data types are represented in native C in the background. But the integration provides automatic conversion of Lua data and the standard syntax can be used for member-access of structures and array-indexing. Furthermore, uascript provides garbage-collection of the UA data types.
- Dynamic development and introspection: uascript can be used as a shell for dynamic development and introspection at runtime. This is especially useful for exploring the design space of technical solutions and as a pedagogical tool for teaching the UA standard since all UA messages (being instances of structured UA data types) can be directly introspected and edited.
- Embeddable: Not only can uascript be used as a standalone scripting environment, it can also be embedded into existing applications. As an example, Fraunhofer IOSB integrated the uascript interpreter into the V-Rep virtual robot experimentation platform [26]. In addition to the usual Lua-scripts driving V-Rep simulations, Fraunhofer IOSB is able to dynamically add UA servers and clients to the simulation that handle the communication with external components.
- Native speed: The underlying client and server are the unmodified C99-implementations provided by open62541 library and run in a separate thread. Blocking synchronizations are used only for callbacks from the native library back into the uascript environment. Examples for this are methods in the UA information model implemented as Lua functions or variables with an external data source implemented in Lua.

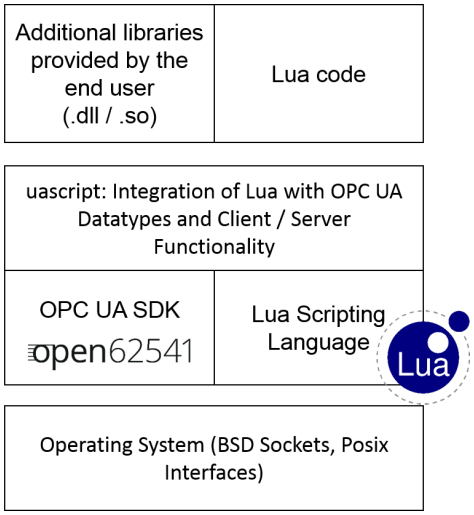

Fig. 3. Architecture of uascript-based OPC UA-enabled applications.

In conclusion, uascript provides a highly flexible integration platform that reduces the cognitive load and removes the usual write/compile/deploy cycle by the integration of OPC UA into a scripting language. See Figures 4 and 5 for some examples. uascript also simplifies the loading of new scripted code-fragments at runtime. For these advantages, it was used to build the orchestration server. It can run on a regular computer, as well as on mobile platforms and even on a low-end Raspberry Pi computer.

Besides using OPC UA for machine-to-machine interaction, a visual dashboard has been developed to show an overview of the scenario and to allow operators to interact with the system. The dashboard is based on an existing ProVis.Visu[1] visualization system based on the C++ UA Toolkit by Unified Automation.

B. Verification of the Camera Software

In this work, we assume that source code is available. Our approach is also based on verification. The system is first modeled as a composition of physical devices (e.g. robot

[1] http://www.iosb.fraunhofer.de/servlet/is/35793/

```
-- define a callback method
function hello_world(object_id, arg1)
    return "Hello " .. tostring(arg1.value)
end

-- initialize the server with a port number
server = ua.Server(48480)

-- add a method node
string_arg = ua.types.Argument()
string_arg.dataType = ua.types.String.typeId
string_arg.valueRank = -1
input_args = { string_arg }
output_args = { string_arg }
attr = ua.types.MethodAttributes()
attr.displayName = ua.types.LocalizedText("", "Hello World")
attr.executable = true
browsename = ua.types.QualifiedName(1, "hello_world")
nodeid = ua.types.NodeId(1, "helloworld")
parent = ua.nodeIds.Objects
parent_reference = ua.nodeIds.HasComponent
server:addMethodNode(nodeid, parent, parent_reference,
                     browsename, attr, hello_world,
                     input_args, output_args)

server:run()
```

Fig. 4. OPC UA server defined in uascript with method-callback back into the Lua-based scripting environment.

```
client = ua.Client()
client:connect("opc.tcp://127.0.0.1:4840")

cmr = ua.types.CallMethodRequest()
cmr.objectId = ua.nodeIds.Objects
cmr.methodId = ua.types.NodeId(1, "helloworld")
cmr.inputArguments = { ua.types.Variant("Peter") }
res = client:call({cmr})
print(res)

client:disconnect()
```

Fig. 5. Client in uascript calling the method in the server.

arm, camera, etc.) that provide functionalities which will be embedded within services. Each critical function is related to its source code. Such models will configure the service oriented framework, that will embed the source code and deploy it as a service that could be called and executed. The service oriented framework integrates an analysis framework that checks the correctness before allowing the deployment. Such operation is performed offline and allow in one hand to extract the application models and on the other hand to validate that the source code is bug free.

The validation approach we adopt does not require manually source code annotation. We validate code against generic properties such as such as absence of deadlock, assertion violation, memory leaks, buffer overflow, null-pointer dereference, non-termination. Code validation allows also to document the source code by automatic generation of abstract views such as control flow graphs or dependency graphs.

The human detection algorithm used in this scenario is a naive threshold filter that discriminates each pixel of the frame into black and white classes. This optimum threshold is automatically calculated for each frame thanks to the Otsu method that minimizes the intra-class variance after building the image histogram. On top of this method, and because we are dealing with a video stream, we take the mean threshold

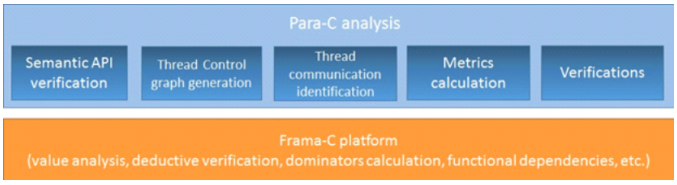

Fig. 6. The Para-C plugin for the Frama-C static code analysis framework.

value between the current frame and the last frame to avoid clipping effects. Then, a parallel section of code generates the new black-or-white image using a fixed number of threads. Finally, we calculate the percentage of black pixels in the picture and we decide whether an object is detected or not by comparing this percentage with a given threshold percentage. This threshold is later called the detection threshold.

The detection threshold is set by the user and has to be tightly adapted to the lightning properties of the scene and the expected detection sensitivity. The intrusion will increase the number of black pixels up to a given point after which the detection threshold will be met. Of course, the used algorithm is not the most performing one. Detection algorithms have been widely studied in the literature and ready-to-use implementations are available in proprietary and open source frameworks such as OpenCV. For example there exist complex algorithms that perform motion detection, tracking and 3-D scene reconstitution with IR or multi-camera matching. However, we use our own implementation of the Otsu filter for source code validation. Our main objective is to have a fully automated verification. This obviously excludes proprietary projects and impose certain constraints on source code.

We used a Frama-C/Para-C tool, dedicated to multithreaded source code verification. Frama-C [27] is a framework for modular analysis of C programs. It is composed by a set of interoperable program analyzers that perform static analysis, deductive verification, and testing for safety and security critical. At this step of development, Frama-C/Para-C requires regular ANSI C code that excludes most - if not all - open source image filter implementations relying on C++ libraries. Also, some actual limitations of the Frama-C/Para-C plugin require a tight management of parallelism (for example a static number of POSIX threads and a clear declaration of thread creation and destruction). We choose Frama-C/Para-C for two main reasons. The first one is to implement a fully automated source code verification on parallel code. Indeed, bugs due to parallelization introduce behavioral problems but also can induce security failures that can be exploited for cyber attacks [28].

Para-C was built on Frama-C layer (cf. Figure 6) and can use each of plug-ins provided by this later. Frama-C does not detect any parallelism. The first step performed by Para-C is a semantic interpretation of the parallel API used within the source code. Then information about the application parallelism is generated. The para-C plugin is an ongoing project. It is developed for two kinds of purpose. The first one is a source code documentation purpose, where the parallel code is parsed, then its software architecture is generated throughout different

graphs (i.e. thread control flow graph, data dependency graph etc.). Para-C also calculates metrics that provide information about the application such as the number of created threads, the number of shared resources, atomic and parallel regions within the source code. The second purpose of para-C is to validate parallel software applications and to check whether they are bug-free. It verifies some safety properties according to some parallel patterns such as absence of deadlocks, atomicity or race conditions.

The connection to OPC-UA was easy and Frama-C/Para-C service was connected to be used from the OPC-UA orchestration server (cf. Figure 1). It is considered as an internal validation service that is automatically called before each new application deployment. When bugs are detected, deployment is not allowed, connection between devices does not occur and the called service is not allowed to execute. Of course, it is impossible in the current state of the state of the art to make fully automated verification in different properties and different kind of source code. Our experimentation have been realized on our own simplified source code which allows to automate the analysis. On a more complicated and realistic source code, this task will be impossible. But having such internal validation framework remains justifiable. The dedicated validation framework within the OPC-UA orchestrator service will unified the validation process and make it easier to assess. It will also save and enhance the know-how by establishing a tool based validation procedure. Some of validation procedures can require manual annotations (for instance, specifying pre and post conditions to make precise verification), at that moment the Frama-C framework will help to build verification and validation patterns that may be re-used on different source code. Hence, having a unified and integrated platform is practical, specially when validation costs are still consumes a large part of development cost.

C. Software Defined Networks

The SDN architecture defines three interaction levels: the infrastructure level, the SDN controller, and high-level services. The infrastructure level comprises the physical communicating devices that support data traffic generation and exchange. These devices could be the robots, cameras, network switches and routers in the factory. To support the SDN architecture, we provide a SDN-software called NEON. NEON establishes and maintains the communication with the SDN controller, collects the network interfaces states and capabilities of all devices, and is able to interpret and activate commands coming from the SDN controller. The SDN controller is the link between high-level applications and the infrastructure. Its role is to collect all information from the architecture, to generate an abstracted view of the system to high-level services and to interpret high-level actions into concrete network actions to be pushed to the infrastructure. The high-level services layers has a direct access to the SDN controller through a Northbound API. This API provides a wide range of output information on different format (JSON-based REST API, binary, JSON-based CLI). The output information could be used to monitor the configuration of the network and traffic performance at near to real time. High-level services could also push commands to request a new configuration or advanced information such as the average signal strength on a specific wireless link. It is also possible for high-level services to push network actions directly on infrastructure devices. Such actions are not interpreted by the SDN controller and could target the realization of a specific task related to the device function, i.e., execution of a high-level method not related to network services. For the latter capability, a plugin system allows to enhance the SDN controller and/or the NEON software with new high-level functionalities.

D. Object Memory Server

Object Memories, as specified by the W3C Object Memory Modeling Incubator Group [29], are data repositories representing physical objects in a digital environment, allowing them to keep a history of their statuses, interactions and changes over time. The Object Memory Server (OMS) [30] serves as an access point to such memories, providing various interfaces, such as a web frontend, an OPC UA server or a representational state transfer (REST) interface. All access methods enable different agents, be it humans, machines or other artifacts, to manipulate memories utilizing a tractable memory block system that is open to many different types of content, from human-readable text over image formats to binary data. The focus of both the OMS and the Object Memories themselves lies on flexibility regarding formats, content and access.

In the context of the developed framework the OMS is used on the orchestration layer to distribute runtime information to support monitoring functions. In order to achieve this, memories for all production cells are created automatically on start-up, using the high-level RobotML description of the scenario. At the beginning, these memories contain mostly information relevant for the whole production environment, such as the linked object's identifier or network interface status. During runtime they are updated continually with current events, thus for example, when the camera detects an intrusion it is logged in its memory where it can be accessed later for trouble-shooting or information. Within the scenario, the OMS is reachable via an OPC UA, as well as a RESTful interface.

VI. Summary

In this paper, we show how recent results from static code analysis and Software-Defined Networking, as well as the ability to use flexible information models in automated systems with OPC UA, can be leveraged to deploy new functionality in manufacturing environments at runtime. We develop an overall systems architecture and give details on the technical implementation of its constituent components. The approach is applied and validated in a manufacturing use case based on a robot for close human-machine interaction and a camera for surveillance of safety areas. These—initially unrelated—assets are combined at runtime in order to increase the safety area surrounding the robot.

In future work, we intend to integrate the results for safe deployment of functionality at runtime in this paper

with the Plug & Produce paradigm. Then a description of the skills of manufacturing assets can be leveraged to derive executable automaton procedures that are combined in a planning system to achieve the manufacturing goals at hand [31]. That way, also topological changes in manufacturing scenarios can be supported by intelligent software systems to enable a more flexible use of companies' manufacturing assets. Furthermore, the description of the scenario and the newly available functionality is currently encoded in RobotML. To simplify the use in manufacturing-related environments, the container format could be exchanged with AutomationML.

REFERENCES

[1] A. Angerer, M. Vistein, A. Hoffmann, W. Reif, F. Krebs, and M. Schnheits, "Towards multi-functional robot-based automation systems," in *Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on*, vol. 2. IEEE, 2015, pp. 438–443.

[2] G. Stoyle, M. Hicks, G. Bierman, P. Sewell, and I. Neamtiu, "Mutatis mutandis: safe and predictable dynamic software updating," *ACM SIGPLAN Notices*, vol. 40, no. 1, pp. 183–194, 2005.

[3] M. N. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer, "Zero downtime reconfiguration of distributed automation systems: The epsiloncedac approach." Springer, 2007.

[4] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[5] C. Suender, V. Vyatkin, and A. Zoitl, "Formal validation of downtimeless system evolution in embedded automation controllers," *ACM Transactions on Embedded Control Systems*, 2011.

[6] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *Software Engineering*. Springer, 2009, pp. 78–105.

[7] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: a survey," *Software Testing, Verification and Reliability*, vol. 23, no. 4, pp. 261–313, 2013.

[8] A. Endo, A. D. S. Sim, S. Souza, and P. Souza, "Web services composition testing: a strategy based on structural testing of parallel programs," in *Practice and Research Techniques, 2008. TAIC PART'08. Testing: Academic & Industrial Conference*. IEEE, 2008, pp. 3–12.

[9] S.-S. Hou, L. Zhang, Q. Lan, H. Mei, and J.-S. Sun, "Generating effective test sequences for bpel testing," in *Quality Software, 2009. QSIC'09. 9th International Conference on*. IEEE, 2009, pp. 331–340.

[10] C. Ma, J. Wu, T. Zhang, Y. Zhang, and X. Cai, "Testing bpel with stream x-machine," in *Information Science and Engineering, 2008. ISISE'08. International Symposium on*, vol. 1. IEEE, 2008, pp. 578–582.

[11] L. Zhou, J. Ping, H. Xiao, Z. Wang, G. Pu, and Z. Ding, "Automatically testing web services choreography with assertions," in *Formal Methods and Software Engineering*. Springer, 2010, pp. 138–154.

[12] J. P. Escobedo, C. Gaston, P. Le Gall, and A. Cavalli, "Testing web service orchestrators in context: A symbolic approach," in *Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on*. IEEE, 2010, pp. 257–267.

[13] J. Garcıa-Fanjul, J. Tuya, and C. de la Riva, "Generating test cases specifications for compositions of web services," *Proc. of WS-MaTe2006*, pp. 83–94.

[14] N. Guermouche and C. Godart, "Timed model checking based approach for web services analysis," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 213–221.

[15] M. Yuan, Z. Huang, X. Li, and Y. Yan, "Towards a formal verification approach for business process coordination," in *2010 IEEE International Conference on Web Services*. IEEE, 2010, pp. 361–368.

[16] M. Felderer, P. Zech, F. Fiedler, and R. Breu, "A tool-based methodology for system testing of service-oriented systems," in *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*. IEEE, 2010, pp. 108–113.

[17] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini, "The plastic framework and tools for testing service-oriented applications," in *Software Engineering*. Springer, 2009, pp. 106–139.

[18] B. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[20] S. Decremps, S. Imadali, and M. Boc, "Fast deployment of services in sdn-based networks: The case of proxy mobile ipv6," *Procedia Computer Science*, vol. 40, pp. 100–107, 2014.

[21] L. Durkop, J. Imtiaz, H. Trsek, L. Wisniewski, and J. Jasperneite, "Using opc-ua for the autoconfiguration of real-time ethernet systems," in *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*. IEEE, 2013, pp. 248–253.

[22] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "Robotml, a domain-specific language to design, simulate and deploy robotic applications," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2012, pp. 149–160.

[23] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.

[24] F. Palm, S. Gruner, J. Pfrommer, M. Graube, and L. Urbas, "Open source as enabler for opc ua in industrial automation," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–6.

[25] R. Ierusalimschy, L. H. De Figueiredo, and W. Celes Filho, "Lua-an extensible extension language," *Softw., Pract. Exper.*, vol. 26, no. 6, pp. 635–652, 1996.

[26] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1321–1326.

[27] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-c," in *Software Engineering and Formal Methods*. Springer, 2012, pp. 233–247.

[28] J. Yang, A. Cui, S. Stolfo, and S. Sethumadhavan, "Concurrency attacks," in *Presented as part of the 4th USENIX Workshop on Hot Topics in Parallelism*, 2012.

[29] A. Kröner, J. Haupert, M. Seißler, B. Kiesel, B. Schennerlein, S. Horn, D. Schreiber, and R. Barthel, "Object memory modeling w3c incubator group report," Worldwide Web Consortium, Tech. Rep., 2011. [Online]. Available: http://www.w3.org/2005/Incubator/omm/XGR-omm

[30] J. Haupert and M. Schneider, "The object memory server for semantic product memories," in *SemProM*. Springer, 2013, pp. 175–189.

[31] J. Pfrommer, D. Stogl, K. Aleksandrov, S. Escaida Navarro, B. Hein, and J. Beyerer, "Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems," *at-Automatisierungstechnik*, vol. 63, no. 10, pp. 790–800, 2015.