Model-Based Systems Engineering for the Agile Life-Cycle Management of IIoT Based on DevOps

Amirali Amiri
Institute of Computer Engineering
TU Wien, Vienna, Austria
amirali.amiri@tuwien.ac.at

Gernot Steindl
Institute of Computer Engineering
TU Wien, Vienna, Austria
gernot.steindl@tuwien.ac.at

Wolfgang Kastner
Institute of Computer Engineering
TU Wien, Vienna, Austria
wolfgang.kastner@tuwien.ac.at

***Abstract*—The domain of Industrial Internet of Things (IIoT) contains a vast variety of standards, protocols, tool suites, coding languages, etc. The Reference Architecture Model Industry (RAMI) 4.0 provides an abstraction of the different views involved in designing IIoT systems. However, the life-cycle model of the reference architecture suggests a progression from the development into production. This progression does not conform well with the best practices of software engineering, e.g., DevOps, where runtime data is given back to the design-time. In this paper, we suggest an agile life-cycle management of the IIoT systems, and propose an approach of Model-Based Systems Engineering (MBSE) based on the System Modeling Language (SysML) 2.0. We define metadata in the domain of event-driven IIoT systems. Architects can create tagged model instances, which are automatically validated against system requirements. The validated models are passed to an artifact generator to create code, test cases, or documentation. Our approach assists in the development and commissioning of system assets. We feed the runtime data to design-time to improve the quality of service.**

***Index Terms*—Industrial IoT, MBSE, DevOps Life Cycle**

I. Introduction

Nowadays, there are many different standards, protocols and tools for the Industrial Internet of Things (IIoT) systems. Architects and system designers must have a very deep expertise to be able to cope with the complex scope of designing such systems that include hardware, software, information, communication, users, etc. The Reference Architecture Model Industry (RAMI) 4.0 [9] presents an abstraction to assist IIoT system designers. Despite being helpful, the life-cycle model of the reference architecture does not match well with the proven methods of the software-engineering domain. For example, the DevOps practice [8] suggests a circular life-cycle management that results in providing feedback from the runtime to the design-time of the industrial applications.

Moreover, proven approaches such as Model-Based Systems Engineering (MBSE) [11] make the design of IIoT systems easier. This is done by providing abstractions of system models and views. To the best of our knowledge, there exists no approach in the literature that applies the MBSE to the life-cycle management of the IIoT systems based on the DevOps practices. Thus, we set out to answer the research questions:

RQ1: *What is an agile life-cycle management of IIoT systems using DevOps practices?*

RQ2: *How can model-based systems engineering assist the agile management of IIoT systems during design- and runtime?*

Fig. 1 shows the RAMI 4.0 three-dimensional model [9]. These dimensions are *Interoperability Layers*, *Hierarchy Levels*, and *Life Cycle and Value Stream*. The *Interoperability Layers* provide different levels of abstractions from the business processes down to the assets integration in an IIoT system. The *Hierarchy Levels* consider an industrial application from the IIoT devices (including sensors, actuators and controllers) up to the cloud integration.

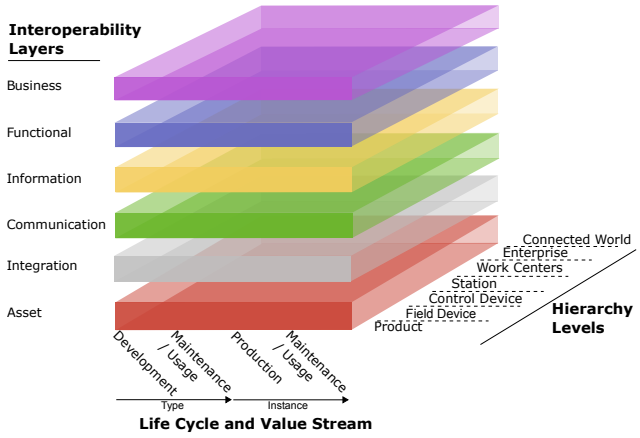

Fig. 1: Reference Architecture Model Industry 4.0

The *Life Cycle and Value Stream* is based on the IEC 62890:2020 [1] standard providing models for a *Product Type* life cycle, i.e., during development, and an *Product Instance* life time, i.e., when operating. However, the agile best practices of software engineering, e.g., DevOps [8], consider a continuous development not reflected by the RAMI 4.0 life-cycle model. Using DevOps, the runtime data is fed back to the design-time to improve the Quality of Service (QoS).

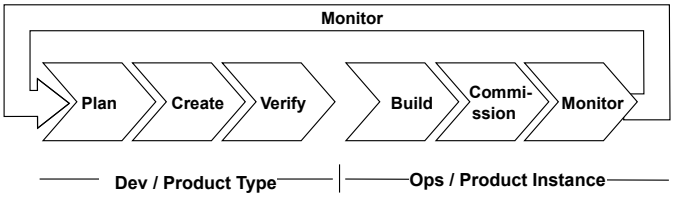

Fig. 2: Adapted DevOps Toolchain to RAMI 4.0 Life Cycle

II. Agile Life-Cycle Management

Fig. 2 shows the DevOps toolchain, adjusted to be comparable to the RAMI 4.0 and applicable to the hardware infrastructure of IIoT systems. The *Dev* phases refer to planning, creating and verification of software and hardware assets. The *Build* phase considers the construction of a hardware entity, or packaging of software artifacts. The *Commission* step brings a hardware asset into operation, or deploys a software component. The adapted DevOps matches with RAMI 4.0, except for the *Monitor* step going back to the *Plan* phase.

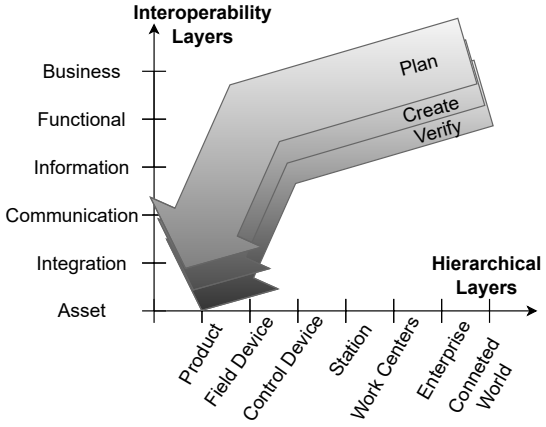

Fig. 3: *Dev* Practices Applied to RAMI 4.0

Dev Phase Fig. 3 shows the *Dev* practices applied to the RAMI 4.0 life-cycle model. The phases start with defining the *business processes*, which are usually high-level activities concerning the *Connected World* and *Enterprise* hierarchy levels. As we refine the processes into *Functional Blocks* and further down to the *System Assets*, we start to consider more software and hardware infrastructure of the hierarchy levels.

Fig. 4 shows the high-level activities of our approach regarding the *Dev* phase. Architects tag system components using our pre-defined metadata definitions in SysML 2.0 (see Section III). The tagging can be done on different levels, e.g., business processes or assets, that correspond to different RAMI 4.0 *interoperability layers*. Our approach validates requirements and informs the architect or system designer of the exact system part(s) violating the requirements. We convert the system model into graphs automatically, and performed graph-based validation of the requirements (see [3] and [4]).

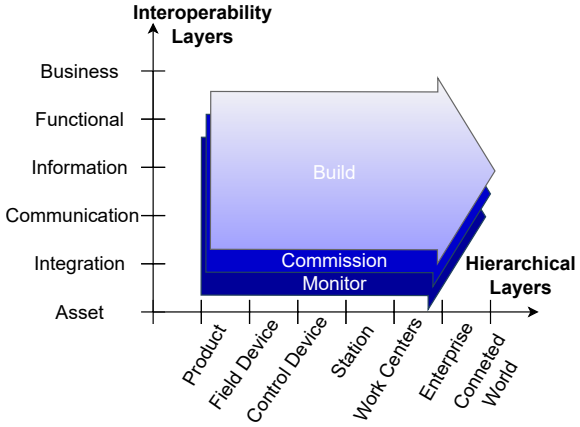

Fig. 5: *Ops* Practices Applied to RAMI 4.0

Ops Phase Fig. 5 shows the *Ops* practices applied to the RAMI 4.0 life-cycle model. We deploy our system model

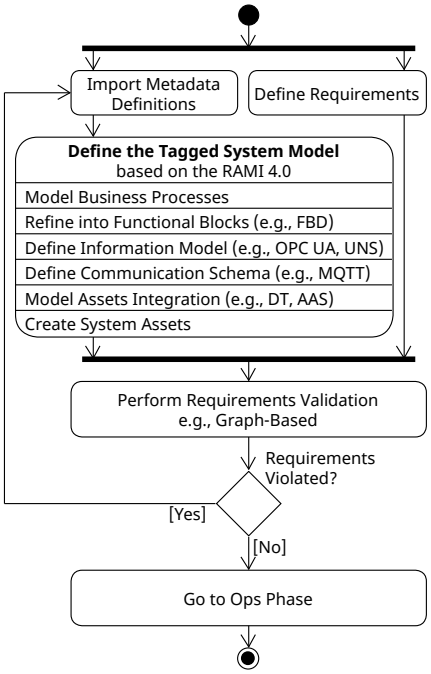

Fig. 4: The High-Level Activities regarding the *Dev* Phase

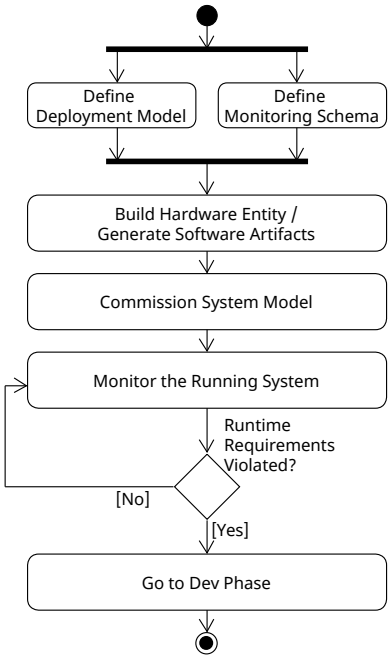

Fig. 6: The High-Level Activities regarding the *Ops* Phase

on the available infrastructure, covering the *Product* to the *Connected World*. We *monitor* the system assets, including software and hardware, for any violation of QoS metrics.

Fig. 6 shows the high-level activities. In this phase, we define a *deployment model* that maps the *functional blocks* to the underlying *execution environment*. These environments are modeled by the RAMI 4.0 *Hierarchy Levels*. We put all together and add the connection from the *Monitor* back to the *Plan* phase. This connection provides feedback from the runtime, e.g., collecting monitoring data of an IIoT system, back to the design-time to improve the QoS. Fig. 7 shows the proposed agile life-cycle management.

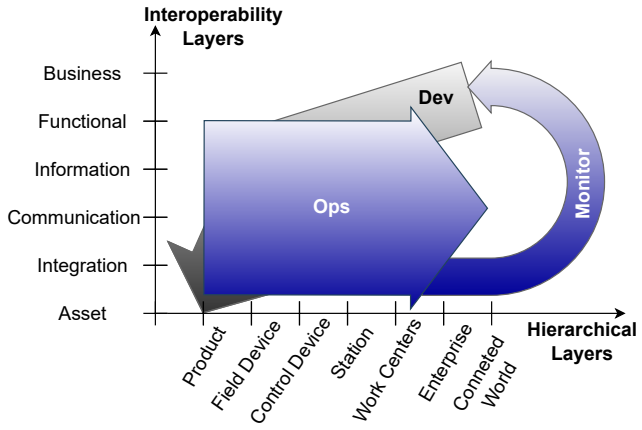

Fig. 7: Agile Life-Cycle Management of the RAMI 4.0

The agile practices are used in the software-engineering domain extensively. In the industrial-automation domain, these practices can be very beneficial to give feedback from runtime to the design-time of applications. In addition to improving QoS metrics, system architects can detect requirement-violations as early as possible at the design-time. It is costly to adjust violations after an IIoT system is in production and running. We present an MBSE approach, automating the use of our concepts to some extent.

III. Model-Based Systems Engineering

This section proposes an MBSE approach. Fig. 8 shows the MBSE workflow [11]. We use the SysML 2.0 textual representation[1] and define metadata in the IIoT domain. These metadata are used as tags when creating a *Model Instance*. A

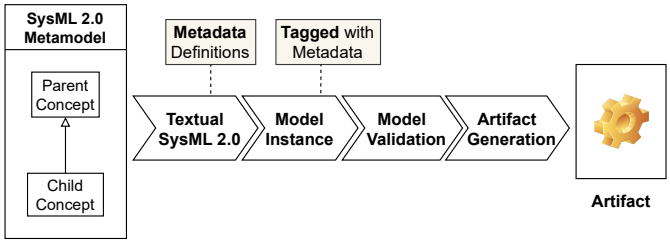

Fig. 8: The MBSE Workflow

[1]https://www.omg.org/spec/SysML/2.0/Beta1

Model Validator verifies system requirements. The validated models are passed to an *Artifact Generator* to generate artifacts, e.g., code, test cases, or documentation. We propose a layered system model based on the *interoperability layers* of the RAMI 4.0. This section details the layered model.

Metadata Definitions Our metadata definitions concerning event-driven IIoT systems are shown in Fig. 9. A *System under Consideration* consists of at least one *Asset* that can be *Information*, *Communication*, *User* with different *Roles*, or *Infrastructure*. A *Base Node* models *Software* and *Hardware*. A *Composite Node* consists of base nodes, and can be *Edge*, *Fog*, *Cloud*, *Message Broker*, *Device Gateway*, or *IIoT Device*. An *Execution Environment* models the virtual or physical environments, e.g., virtual machines, containers, or baremetal servers. A *Requirement* models systems requirements. *Counter Measures* are actions performed to fulfill the requirements. *Test Cases* assess the effectiveness of counter measures. Architects can easily extend our metadata, e.g., by defining user roles.

SysML 2.0 Metadata Listing 1 shows an excerpt of SysML 2.0 metadata. We perform the *Plan* and *Create* phases of the *Dev* practices by tagging the system model using the metadata.

```
package Meta_EventDrivenIIoT {
    enum def UserRole;

    metadata def Asset ;
    metadata def User :> Asset {
        import Meta_EventDrivenIIoT::UserRole::*;
        attribute Role : UserRole; }
    metadata def Communication :> Asset;
    metadata def Information :> Asset;
    metadata def Infrastructure :> Asset;
    metadata def BaseNode :> Infrastructure;
    metadata def Hardware :> BaseNode;
    metadata def Software :> BaseNode;
    metadata def CompositeNode :> Infrastructure {
        import Meta_EventDrivenIIoT::*;
        metadata BaseNode;  }
    metadata def Edge :> CompositeNode;
    metadata def Fog :> CompositeNode;
    metadata def Cloud :> CompositeNode;
    metadata def MessageBroker :> CompositeNode;
    metadata def DeviceGateway :> CompositeNode;
    metadata def IIoTDevice :> CompositeNode;
        ...
}
```

Listing 1: Exceprt of Metadata Definitions in SysML 2.0

Layered System Model We follow the *interoperability layers* of the RAMI 4.0 to refine our system model iteratively. Our proposed steps are as follows, also visualized by Fig. 4. First, we model high-level *business processes* based on system requirements using SysML 2.0 activity diagrams. Second, we refine each business process into *functional blocks* to realize the business processes. Third, we consider functional blocks that can be code-generated. Also, we consider black-box functional blocks that need to be manually completed by the developers. Next, we define *information models* to provide semantics for the data exchange between functional blocks. Then, we define the *communication schema* between functional blocks. Also, we consider the *digital integration* of each asset to run specific functional blocks and the communication schema. Last, we define all system *assets*, including the cloud, fog and edge nodes. Note that the order of the steps above is

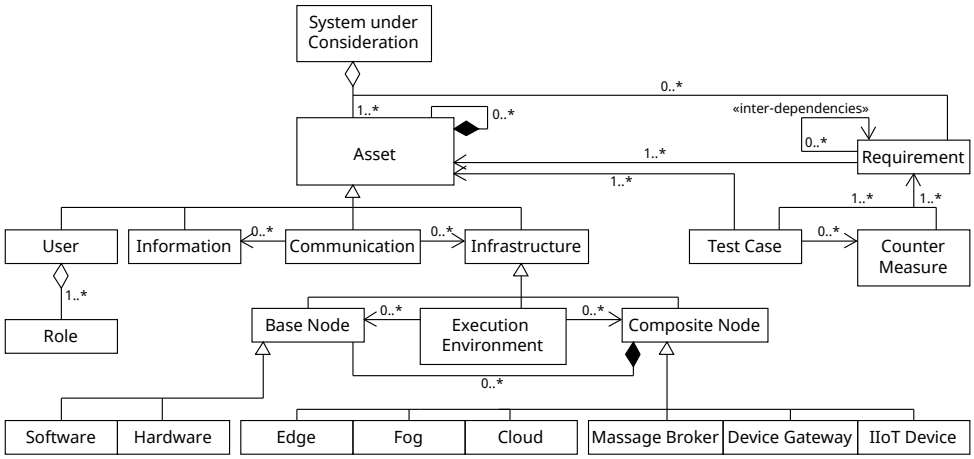

Fig. 9: Metadata of Event-Driven IIoT Systems

arbitrary. A validator verifies the model against requirements automatically, e.g., by performing traversal algorithms on the converted graph of the system model [3], [4].

Ops Phase Having validated the models, we focus on the *Ops* practices, also visualized by Fig. 6. We define a monitoring schema to check runtime QoS metrics, e.g., performance, accuracy, etc. At this point, we have created different views of the system model, i.e., structural, behavioral, and runtime models. We can now generate artifacts based on pre-defined templates. These templates map our metadata definitions to code blocks or documentation. However, in some cases, the application-specific code must be completed by developers to have a fully-functional system.

We deploy the software artifacts, commission hardware infrastructure into operation, and monitor the runtime metrics continuously. In case of violation, we go to the *Dev* phase, and adjust our models. This step is performed manually by an architect or a system designer based on the recommendations of our approach. Also, a self-adaptive system can automatically update the system model using the Monitor, Analyse, Plan, Execute and Knowledge (MAPE-K) loops [6], [7], [10].

IV. Conclusions and Work-In-Progress

In this paper, we set out to answer what an agile life-cycle management of IIoT systems using DevOps practices is (**RQ1**), and how model-based systems engineering can assist the agile management of IIoT systems during design- and runtime (**RQ2**). For **RQ1**, we proposed a continuous life-cycle management based on DevOps practices [8], where runtime data is fed back to the design-time to improve the quality of service. For **RQ2**, we suggested an MBSE approach based on SysML 2.0 textual representation. We defined metadata concerning the event-driven IIoT systems that can be used to created tagged system models.

Our work-in-progress includes generating code based on the standard IEC 61131-3:2013 [2], and deploying our generated artifacts on an experimental setting. We plan to evaluate our concepts using empirical data on local and public cloud infrastructures. Our evaluation scenario is as follows: Architects model an event-driven industrial system that includes message brokers. The generated industrial control code based on IEC 61131-3:2013 runs on Programmable Logic Controllers (PLCs). These PLCs send data to the brokers for the edge-cloud integration. We measure the response time of requests and include a monitoring schema to adapt the system model based on the runtime data. As mentioned, this adaptation can be performed manually or automatically using MAPE-K loops [6], [7], [10]. The self-adaptation usually requires optimization (see, e.g., [5]).

References

[1] Industrial-process measurement, control and automation - life-cycle-management for systems and components, 2020-07-22, https://webstore.iec.ch/en/publication/30583.

[2] Programmable controllers - part 3: Programming languages, 2013-02-20, https://webstore.iec.ch/en/publication/4552.

[3] A. Amiri, G. Steindl, I. Gorton, S. Hollerer, W. Kastner, and T. Sauter. Integrated safety and security by design in the IT/OT convergence of industrial systems: A graph-based approach. In *IEEE International Conference on Software Services Engineering*, 2024.

[4] A. Amiri, M. Thoma, G. Steindl, C. Klaassen, and W. Kastner. Model-based systems engineering of the event-driven industrial internet of things. In *Annual Conference of the Industrial Electronics Society*, 2024.

[5] A. Amiri and U. Zdun. Tool support for the adaptation of quality of service trade-offs in service- and cloud-based dynamic routing architectures. In *European Conference on Software Architecture*, 2023.

[6] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23, 2015.

[7] P. Arcaini, E. Riccobene, and P. Scandurra. Formal design and verification of self-adaptive systems with decentralized control. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2017.

[8] N. Azad and S. Hyrynsalmi. Devops critical success factors — a systematic literature review. *Information and Software Technology*, 2023.

[9] R. Heidel. *Industrie 4.0: The reference architecture model RAMI 4.0 and the Industrie 4.0 component*. Beuth Verlag GmbH, 2019.

[10] D. G. D. L. Iglesia and D. Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):1–31, 2015.

[11] A. L. Ramos, J. V. Ferreira, and J. Barceló. Model-based systems engineering: An emerging approach for modern systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 42(1):101–111, 2012.