SIMULATING CYBER-PHYSICAL SYSTEMS USING A BROKER-BASED SYSML TOOLBOX

Sylvia Melzer, Jan Speichert, Oliver C. Eichmann, Ralf God

Institute of Aircraft Cabin Systems, Hamburg University of Technology
Hein-Saß-Weg 22, 21129 Hamburg, Germany

sylvia.melzer@tuhh.de

Abstract
The trends of Industry 4.0 and Internet of Things (IoT) are motivating companies to digitise their processes and products in order to offer new data-driven services to customers. Digitally supported services typically require a combination of a business in our physical world with flows of information in the virtual world, i.e. the cyberspace. Thus, design and implementation of so-called Cyber-Physical Systems (CPS) demand simple and configurable messaging architectures.

Designing CPS is a challenging task since the distribution of system components and information, multiple system interfaces, and connections as well as ancillary system security aspects must be considered. Using methods and tools of Model-Based Systems Engineering (MBSE) is an established approach for the development of complex and software-intensive systems. The methods and tools enable executable specifications, simulation, verification, and system security analyses in early stages of the systems engineering process. Hardware integration into models allows rapid prototyping and fosters a deep understanding of service systems already in the design phase. This reduces misconceptions, design flaws, and consequently error induced costs. However, recent methods and tools do not support a simple and configurable integration of information exchange. Utilising Information and Communication Technology (ICT) in the design phase, i.e. hardware for rapid prototyping, supports the simulation and verification process.

This paper presents a broker-based Systems Modeling Language Toolbox (SysML Toolbox) using the open source message broker software RabbitMQ. The applicability of the SysML Toolbox is demonstrated in the design of a CPS by means of modelling, rapid prototyping, and simulating a baggage tracking system which fulfils the requirements of IATA Resolution 753 for accurate baggage inventories in air travel.
1 INTRODUCTION

Former system development was characterised by the design of an isolated system taking into account only a limited amount of interfaces to other systems as illustrated in the lower part of Figure 1. Today, companies must evolve from system suppliers to service providers in order to increase and maintain market shares. As displayed in the upper part of Figure 1, today’s services are almost always built upon several interacting systems and their communication [1].

So-called platforms offer services to their customers by integrating several reciprocal systems into a network. As an example the platform “booking.com” offers advice, planning, and booking of holiday and business trips. In the past, the travel agency and the systems for hotel booking and billing had loose interfaces with a low degree of coupling and automation. The current platform integrates these systems with close coupling in order to satisfy a new level of customer demands [2].

Whenever many different systems interact, the communication architecture plays an important role. Message broker software offers architecture with the freedom for message routing, transformation, and validation and is therefore predestined for the connection of systems in a diverse environment like the Internet of Things (IoT) [3].

In the development process of service enabling Cyber-Physical Systems (CPS) model based approaches are established. A model is defined as an abstract description of reality [4] avoiding cost intensive design flaws at an early stage in the development process for two reasons. First, models enable a consistent system understanding by all involved stakeholders avoiding misconceptions. Second, models support early verification of development results by simulation [5].

Simulation is also applicable to message broker software and hardware. This paper presents a broker-based Systems Modeling Language Toolbox (SysML Toolbox). It uses the open source message broker software RabbitMQ allowing the early simulation and test of hardware combined with broker software and applies it to
the simulation of a baggage tracking system. Since IATA Resolution 753 demands tracking of baggage during check-in, baggage loading, baggage transfer, and arrival [6] a baggage tracking system contains many sensing points for baggage detection operated by different stakeholders, i.e. airports, airlines and ground handling staff. Thus, the communication architecture development is an important aspect in the improvement of the baggage transport service supported by the broker-based SysML Toolbox.

2 BROKERAGE

Today’s increasing demand for flexible and configurable communication networks drives the development of message broker technologies. A message broker is a software translating messages between at least two applications. Those applications do not necessarily communicate in the same messaging protocol and may not even know each other. The broker itself is software-configurable, message-oriented, and able to route, transform, and validate messages. Different routing schemes are supported, including common messaging paradigms like (1) send and receive messages, (2) create work queues for distributed tasks among multiple workers, (3) publish and subscribe messages, (4) routing messages, (5) topic-based messaging, and (6) Remote Procedure Call (RPC). The broker middleware enables an effective decoupling of participants in the network. Due to the high configurability, in addition, a broker can provide different levels of Quality of Service (QoS), data retention, and data replication to match the specific requirements for an application. In terms of applications, used protocols, structure, and architecture there is a variety of broker technologies on the market. This paper focuses on message brokers supporting communication within contemporary Internet of Things (IoT) networks. As applications in these IoT networks typically range from small sensors up to cloud-based High Performance Computing (HPC) entities, the requirements on bandwidth, latency, and message length are on both ends of the scale. Multiple approaches and concepts match these requirements and have been developed as well as promoted by different organisations and companies.

2.1 MQTT as a Message Protocol for IoT

Amongst the most popular protocols for IoT networks is the Message Queuing Telemetry Transport (MQTT) protocol. MQTT is an ISO standard [7] initially developed by IBM and submitted to the Organization for the Advancement of Structured Information Standards (OASIS). It is designed for lightweight applications such as embedded systems due to a small size in code and message length. Even though the “MQ” in MQTT suggests a message queuing this is not a mandatory part for implementation. Therefore, the OASIS no longer expands the acronym. Clients send or publish their messages to a tree-like structure with each end-point being a so-called topic where messages are stored. In order to receive messages a client needs to subscribe to a topic. The usage of wildcards enables a client to subscribe to multiple topics. In addition, multiple clients can subscribe to one topic, resulting in a many-to-many relationship between topics and clients. This so-called
publish-subscribe pattern is a common technique in message-oriented systems. Apart from MQTT, many other protocols are based on or are supporting a publish-subscribe pattern, e.g., Extensible Messaging and Presence Protocol (XMPP), Data Distribution Service (DDS), Amazon Simple Notification Service (SNS), or Advanced Message Queuing Protocol (AMQP).

2.2 AMQP as a General-Purpose Message Protocol and Model for CPS

One of the very popular and most advanced protocols in Cyber-Physical Systems (CPS) is the Advanced Message Queuing Protocol (AMQP). In document [8], AMQP (version 0-9-1) is specified with messaging capabilities called the Advanced Message Queuing Model (AMQ Model). While the protocol specification defines the actual wire-level protocol needed for communication between clients and servers, the AMQ Model defines the semantics for the server-side components. The AMQ Model consists of three major components, namely exchanges, message queues, and bindings. Messages sent to the broker are received by the exchange which routes messages to message queues, based on previously defined parameters. These parameters are defined in the binding component, a representation of the relationship between exchange and message queue. Messages are stored in the message queue and can be consumed by one or multiple clients. Arbitrary complex communication architectures, i.e., an airport-to-airport communication network architecture, can be built using solely these three components. Different exchange and message queue settings can be easily combined in order to match almost every requirement on communication of participating clients. The AMQ Model defines four different exchange types, namely direct, fanout, topic, and headers exchange. Direct exchange is the default. Each exchange type allows different rulesets to be applied on bindings between exchange and message queues. E.g., a fanout queue simply forwards every message to every queue bound to the defined exchange, the direct exchange relies on a routing key in order to decide to which queues messages shall be forwarded. While MQTT is a specific lightweight design for low-energy consuming and small footprint applications, AMQP can be considered as a powerful and high-end general-purpose message protocol. Both protocols are often compared to each other, although there considerable differences in scope and limitations. During the design phase of a system the basic communication requirements should be very carefully examined in order to choose the most suitable protocol for an implementation. As a tool for developing CPS the general-purpose protocol AMQP offers more possibilities than MQTT and therefore AMQP was chosen for the design of the broker-based SysML Toolbox.

2.3 Broker Federation

In IoT communication networks, there are often multiple entities, distributed clients, and brokers with special requirements w.r.t. the communication paradigms and protocols. Figure 2 presents a simplified European airport communication architecture consisting of the four airports at Hamburg (HAM), London (LHR), Paris (CDG), and Madrid (MAD). All airports will have to exchange information in order
to perform transport of passengers, their baggage, and cargo from airport A to airport B. In the example of a message-driven airport environment every airport hosts at least one AMQP broker. A popular AMQP broker is RabbitMQ [9, 10], a full-featured implementation written in programming language Erlang [11]. The many stakeholders at each airport can utilize such RabbitMQ broker to communicate with each other on different communication paths. On the one hand such communication serves to digitize current operational processes at an airport. On the other hand communication with passengers can improve customer services enhancing the information level of all stakeholders. Most of the locally required communication paths could be handled by only one local airport broker instance. But when an inter-airport communication is targeted there is a strong demand for software-configurable broker-to-broker communication. A simple high-availability clustering and mirroring of all locally available airport information would not comply with the idea of a streamlined and efficient architecture. In order to achieve such configurable distributed communication architecture, the information exchange between brokers should be kept at the minimum level required.

Figure 2 – An exemplary European airport communication architecture consisting of four airports at Hamburg (HAM), London (LHR), Paris (CDG), and Madrid (MAD). One AMQP broker is planned for each airport.

Considering for example passenger baggage for a flight from HAM to MAD a ground handling service provider generates a lot of data (passenger id, baggage id, gate number, on- and off-block time, destination, etc.). As the data might be very detailed and specific at HAM airport, i.e. the gate information and block time, only a subset of this data may be needed at MAD airport. A simple way of accomplishing this task is to establish a client which consumes and processes data from the broker in HAM and sends the data to the broker in MAD. Such distribution of multiple RabbitMQ brokers on different locations is called Broker Federation.
In the following the broker-based SysML Toolbox for a model-based development of CPS is presented, and, subsequently the application of this toolbox for implementing a broker federation system is described.

3 MODEL-BASED DEVELOPMENT OF CYBER-PHYSICAL SYSTEMS

The development of Cyber-Physical Systems (CPS) is challenging due to their complexity and a novel data-driven functionality, i.e. coupling digitised processes and products with Information and Communication Technology (ICT). In order to support the development, standardization efforts for CPS are strived for [12]. Currently, many different software tools exist in aviation industry. For example Original Equipment Manufacturers (OEMs) use more than a thousand of tools throughout the development process in order to develop a new aircraft [12]. However, the different tools are not designed to work throughout the whole development process or to develop a specific CPS. Some researchers propose architecture [13], design language [14], meta-model [15], or methods for developing a specific CPS [16]. These approaches require the implementation of new modelling languages, new tools, new developing skills, and new strategies for connecting systems to other systems.

In the following a methodology for developing a CPS with established methods, tools, and languages is presented. More in detail, developers are supported in model-based development of CPS using the well-known semi-formal modelling language SysML, the MBSE tool Cameo Systems Modeler, and process models such as the V-model or the recurrent and refining agile method. In addition a simple integration of hardware for rapid prototyping during the design phase is supported by the broker-based SysML Toolbox which is described in the following.

3.1 The Broker-Based SysML Toolbox

The broker-based SysML Toolbox is an implementation of the six different messaging paradigms of the open source software RabbitMQ in Cameo Systems Modeler (version 18.5), i.e. (1) send and receive messages, (2) create work queues for distributed tasks among multiple workers, (3) publish and subscribe messages, (4) routing messages, (5) topic-based messaging, and (6) Remote Procedure Call (RPC). For creating messaging applications for each paradigm in SysML the model element opaque behaviour is used. The opaque behaviour is an implementation-specific model element for specifying executable behaviours in SysML [16]. Languages of an executable opaque behavior in Cameo Systems Modeler can be Object Constraint Language (OCL) 2.0, binary, BeanShell, Groovy, JRuby, JavaScript, Jython, or StructuredExpression. The RabbitMQ community offers a Java implementation of a RabbitMQ client (rabbitmq-client.jar) as well as the basics for creating messaging applications using RabbitMQ. For creating messaging applications using the RabbitMQ as a message broker, the java-like scripting language BeanShell is chosen as the language of the opaque behavior. Thus, for the broker-based SysML Toolbox the basics of the RabbitMQ java source code have been translated to BeanShell. Developers can use the implementation specific
model element via drag-and-drop in their SysML model as **opaque actions** in order to create a required communication network easily.

![Diagram](image.png)

**Figure 3 – Opaque actions for sending and receiving messages from the RabbitMQ server as well as the BeanShell source code for the messaging applications**

In Figure 3 the opaque actions `sendMessage.bsh` and `receiveMessage.bsh` are presented which are parts of the broker-based SysML Toolbox. Figure 3 shows the messaging applications which are written in BeanShell for sending (Figure 3 left) and receiving (Figure 3 right) messages to and from the RabbitMQ server. The opaque action `sendMessage.bsh` needs the message, the broker configuration, and the queue configuration as input parameters while the `opaque action receiveMessage.bsh` requires the broker and queue configurations and returns a message as output value.

The SysML Toolbox has two SysML blocks for modelling the configuration elements: `MessageBroker`, `MessageQueue`, and `MessageExchange`. The `MessageBroker` has the properties `host`, `port`, `username`, `password`, and `virtualhost`. The `MessageQueue` has the properties `queueSend` and `queueReceive`. The `MessageExchange` has the properties `exchangeName` and `routingKey`. Developers can create instances of these SysML blocks in order to set individual configurations of the RabbitMQ server. In order to simulate modelled communication networks, the RabbitMQ java client `rabbitmq-client.jar` has to be imported into the tool Cameo Systems Modeler. Overall, the integration of hardware and software into the SysML model is very simple if the hardware and software also use the RabbitMQ API with the same configurations, i.e. host, virtual host, and queue names, as defined in the SysML model. With the Toolbox a simulation and validation of a modelled CPS are supported during the design phase and modelled components can be easily replaced by real software and hardware components.

Figure 4 represents the broker-based SysML Toolbox with the six messaging paradigms (cf. Figure 4, top left), visualizes the first paradigm with the SysML elements **opaque actions** (`:sendMessage`, `:receiveMessage`) for sending and receiving messages (top right), shows the sent and received messages in the console of Cameo Systems Modeler (bottom right) as well as the simulation environment (bottom left). The other paradigms were implemented as **opaque behavior** elements analogously. Note that due to tool-specific restrictions the implementation of the publish and subscribe messages paradigm (3) had to be solved by programming a subscribe
message application in Java which has to be imported into the tool Cameo Systems Modeler.

Broker-based SysML Toolbox

Figure 4 – The broker-based SysML Toolbox as part of the Cameo Systems Modeler

4 APPLICATION AND RESULTS

As a part of the project KomKab (Kommunizierende Kabine - Digitaler Ramp Agent) [18] concepts for an improved and more current information situation for the ramp agent is pursued. One concept is the realization of a baggage tracking system meeting the requirements of IATA Resolution 753. In 2018 the IATA Resolution 753 on baggage tracking became effective and is intended to encourage airlines to further reduce mishandling by implementing cross-system tracking for each baggage journey [6]. In Figure 5 left bottom the baggage journey is illustrated with the four mandatory tracking points (A) CHECK-IN, (B) LOAD, (C) TRANSFER, and (D) ARRIVAL. For the development of the desired baggage tracking system, European airport communication architecture is specified and a broker federation system using the broker-based SysML Toolbox is modelled (cf. Figure 5).

In Figure 5 the four airports HAM, MAD, CDG, and LHR are presented in so-called swimlanes. Each airport has a broker with its individual broker configuration. It is shown for the journey from HAM to MAD (cf. Figure 5 top left) that the processes (A) CHECK-IN and (B) LOAD are located in HAM. During the check-in process an individual dataset with all relevant flight information, including the baggage id, is generated. In HAM the dataset has the name BagEventNotif_HAM (according to the IATA’s BaggageXML definition [19]). In this case, the standardized, user-friendly, and lightweight data-interchange format JavaScript Object Notation (JSON) is used. The dataset BagEventNotif_HAM.json is sent to
MAD and MAD receives the dataset before or after the passenger has arrived at MAD (process step D). More technically, the opaque behavior element *EmitLogDirect.bsh*, c.f. routing paradigm (4), is used. The HAM broker has the dataset *BagEventNotif_HAM.json* as well as the broker and exchange configurations *brokerConfigHAM* and *routingExchangeConfigHAM* as input. The MAD broker has the broker configuration *brokerConfigHAM* and the queue configurations *queueMAD* as input parameters. The opaque behavior element *receiveMessage.bsh*, c.f. send and receive paradigm (1), is used for receiving the dataset *BagEventNotif_HAM.json*.

![Diagram](image)

**Figure 5 – A model-based implementation of a baggage tracking system w.r.t. IATA Resolution 753 with the four mandatory tracking points (A) CHECK-IN, (B) LOAD, (C) TRANSFER, and (D) ARRIVAL**

In Figure 5 on the right a journey from LHR (process step A and B) via CDG (process step B and C) and MAD (process step D) is presented. The process steps (A to D) are implemented analogously. Therefore the same paradigms are used for sending, receiving, and routing messages. In addition, CDG is a transfer airport so that the transfer baggage (process step C) has to be tracked in CDG. In Figure 5 the airport CDG has a broker for receiving the dataset *BagEventNotif_LHR.json* from LHR and sending a new dataset *BagEventNotif_CDG.json* to MAD while the broker configurations are differing. For receiving the dataset the configurations of LHR and for sending data the configurations of MAD are used. This example shows how model elements of the SysML Toolbox can be easily used for implementing and simulating a broker federation system.

In this paper the application of the broker-based SysML Toolbox was demonstrated for an exemplary European airport communication architecture using...
the broker federation principle. Opaque behavior elements in the SysML Toolbox enable modelling of message broker paradigms by drag-and-drop avoiding coding efforts. Architecting and modelling of broker-based messaging applications are supported by predefined modelling elements solving the communication needs for developing new services in a cyber-physical System of Systems.

5 ACKNOWLEDGEMENTS

We thank Ulrike Wittke and Christian Witte for their valuable contribution to this work for a broker-based SysML Toolbox. This work was supported by the LuFo V-2 project “Kommunizierende Kabine - Digitaler Ramp Agent”, funded by the Federal Ministry for Economic Affairs and Energy based on the decision by the German Bundestag.

6 REFERENCES

[16] Rainer Müller, Leenhard Hörauf, Matthias Vette, and Christoph Speicher, Planning and developing cyber-physical assembly systems by connecting virtual and real worlds. (ScienceDirect, 2016).