Chapter 4
Introduction to Case Studies

Safa Bougouffa, Kiana Busch, Robert Heinrich, Christopher Haubeck, Suhyun Cha, Ralf Reussner, and Birgit Vogel-Heuser

This chapter introduces the case studies used in the DFG Priority Programme *Design For Future – Managed Software Evolution* (SPP 1593). Section 4.1 gives an overview of evolution in information systems and automated production systems. An open community case study for software architecture modelling and evolution, the Common Component Modeling Example, is introduced in Sect. 4.2. An open demonstrator for automated production systems, the Pick and Place Unit (PPU) and its extension (xPPU), is introduced in Sect. 4.3. Finally, both case studies are integrated as Industry 4.0 demonstrator and introduced in Sect. 4.4.

4.1 Evolution of Long-Living Systems to an Industry 4.0 Case Study

Many industrial information systems are operated over decades. During operation, they face various modifications, for example due to emerging requirements, bug fixes, and environmental changes, such as legal constraints or technology stack updates. In consequence, the systems change and evolve continually.

S. Bougouffa (✉) · S. Cha · B. Vogel-Heuser
Technische Universität München, Lehrstuhl für Automatisierung und Informationssysteme, Garching, Germany
e-mail: safa.bougouffa@tum.de; suhyun.cha@tum.de; vogel-heuser@tum.de

K. Busch · R. Heinrich (✉) · R. Reussner
Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
e-mail: kiana.busch@kit.edu; robert.heinrich@kit.edu; reussner@kit.edu

C. Haubeck
Universität Hamburg, MIN-Fakultät, Fachbereich Informatik, Hamburg, Germany
e-mail: haubeck@informatik.uni-hamburg.de

R. Reussner et al. (eds.), *Managed Software Evolution*,
https://doi.org/10.1007/978-3-030-13499-0_4

Supporting software evolution is a competitive advantage in software engineering. A variety of methods aim at supporting different aspects of software evolution. However, it is hard to assess their effectiveness and to compare them due to divergent characteristics [Hei+15a]. Empirical research in terms of case studies and controlled experiments is useful to validate these methods. However, empirical studies on software evolution are rarely comprehensive.

To study evolution comprehensively, we believe it is important to collaborate by joint research. Joint research supports sharing of knowledge and resources [SDJ07]. In particular, this allows replicating studies, which in general is important to confirm and to strengthen the results of empirical research [JG12] and thus enhance evidence. Our goal is to support joint research by collaboration and replication in empirical studies based on common evolution scenarios and artefacts. Currently, empirical studies on software evolution are seldom comparable as they vary in analysed subjects and execution process. Furthermore, these studies are rarely reusable as important artefacts (e.g. requirements, design decisions, architectural knowledge, or context knowledge) are often not provided to the community. A common basis for study collaboration and replication is missing. To overcome these shortcomings in the SPP1593, two case studies are used: the Common Component Modeling Example (CoCoME) as a community case study for software architecture modelling and evolution and the Pick and Place Unit (PPU) and its extension (xPPU) as a community case study for automated production systems' evolution. CoCoME represents a knowledge base for collaborative empirical research on information system evolution [Hei+15a]. The knowledge base for the evaluation process can be exploited and extended by researchers with different backgrounds and research interests. It provides assistance on diverse characteristics that are important for software evolution, like artefacts in different revisions, comprehensive evolution scenarios, and coverage of different life-cycle phases (development-level evolution and operation-level adaptation). The xPPU represents a lab-size demonstrator for investigating research on evolution in machine and plant automation [LFV13]. The original PPU featuring 13 evolution scenarios is limited in size and complexity, but it has been extended with its functionality, together with the additional structure with over 10 evolution scenarios. Different evolution scenarios are provided [LFV13] to demonstrate its various change reasons, for example changing requirements, fixing of failures, and unanticipated situations on site. The xPPU evolution scenarios are provided to meet research requirements of the community, together with their artefacts such as system architecture, models, runtime data and code.

The community case studies aim at providing several benefits to researchers:

- By building upon existing specifications and settings, less effort in scenario definition, study setup, and execution is required.
- A common case study increases the comparability of evaluation results with those of other researchers and leads to increased evaluation confidence.
- A common case study also increases community acceptance by interaction with other researchers.

Unlike information systems, automated Production Systems (aPS) consist of artefacts of multi-disciplines and are all closely interwoven; the software for an aPS is strongly influenced by the hardware, which is implemented by mechanical and electrical/electronic components. Usually the complexity of the software and the system itself is very high; therefore, it is not obvious how a change in one discipline is affecting the software, context or platform of the system. [Jäg+11] even though maintainability is an important aspect for a long-living system.

The three disciplines involved in aPS are regarded as three different aspects of context, platform, and software [Leg+14]. Context includes the mechanical aspects of the system, such as pure mechanical components, sensors, and actuators. Platform represents electric/electronic aspects, which manage signal flows from or to the interfaces of the Programmable Logic Controller (PLC). Software reflects software engineering aspects, which consist of data-processing functions using the flow-in information to generate the flow-out information. The software of the aPS has been implemented in IEC 61131-3 [Int09], to be run on PLCs.

Modularity, which is one of the key aspects to enable evolution of software-intensive systems [PCW84], is still rarely fully applied in aPS [Vog+17b]. Moreover, fundamental methods such as variability modelling and tracing, which enable software evolution, are still limited to the software domain of the cyber-physical systems. However, aPS impose special requirements on the development and maintenance process. For instance, mechatronic components are designed to function for several years. However, it is predictable that their development and maintenance will change over their utility lifetime. To allow for later adaptions to the functionality of these components, suitable means should be considered during the development. As software can be adapted more easily than mechanical or electrical parts, changing the control software of aPS may solve adaption requirements. However, these changes may result in code smells, as they are usually conducted quickly on site by technicians.

aPS are supposed to operate for decades. During operation, they are ageing. For instance, as a result of physical effects like wear, tear, and corrosion, life expectations of mechanical components are affected. These components have to be maintained after some years, known as re-engineering and modernisation (cp. Fig. 4.1). There are other reasons for ageing such as changing requirements and system specifications, market requirements, new technologies or legal requirements. Many of those changes can be realised by adapting the control software, which is done more frequent and even during runtime.

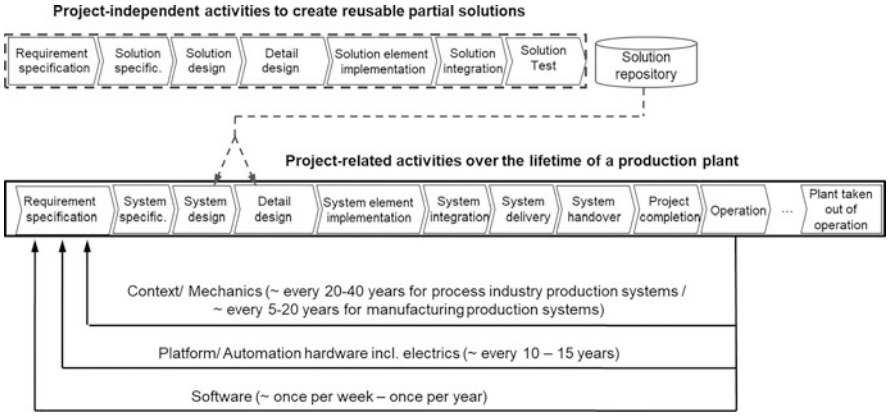

Fig. 4.1 V-Model XT integrated into the life cycle of different disciplines in aPS distinguishing between project-independent activities (top) and project-related activities (bottom) [Vog+15c]

Scenario category: / Causal Order of changes and change criteria		Ia	Ib	Ic	Id	IIa	IIb	IIc	III	IVa	IVb	IVc	Va	Vb	Vc	Vd	VIa	VIb	VIc
PPU scenario (cp. Table 2)		Sc11, Sc12 (*)	Sc12d (#)	Sc1 (*)	Sc12c (#)	Sc13 (*)	Sc1 (*)	Sc2, 4,4b, 6 (*)	Sc6 (*)	Sc4b (*)	Sc1 (*)	Sc12c (#)	Sc11, Sc12	Sc12d (#)	Sc1 (*)	Sc12c (#)	Sc13 (*)	Sc1 (*)	Sc12c (#)
(a) Requirements of the plant's management (informal)		1.											unchanged						
(b) Semi-formal system requirements specification (SRS)		2.							omitted										
Artifact	(c) Software of the aPS	3.				4.			2.	3.			1.				2.		
	(d) electrical parts of the aPS		3.		3.	3.		3.		2.		2.		1.		1.	1.		1.
	(e) mechanical parts of the aPS			3.			3.				2.				1.			1.	
Anticipation of Change (Buckley et al., 2005)		yes							no										
Time of change		offline							online										

Fig. 4.2 Categories of evolution scenarios with references to PPU case study examples in Section 3 [Vog+15c]

Evolution of aPS can be initiated by different reasons for change, which can affect the software, mechanical, and/or electrical and electronic parts. A classification of evolution was introduced in [Vog+15c], which distinguishes causal orders of change by which the aPS is affected (see Fig. 4.2). The evolution categories can be related to the history of change and anticipation of change [Buc+05, KVF04]. Anticipated changes include changes during the development of the system and also during operation in case of a model-based approach (i.e. offline changes). Moreover, changes during commissioning and operation are categorised as unanticipated changes (i.e. online changes), as they are implemented directly in the aPS during runtime.

In order to blur the boundaries between pure information systems and automated production systems, recent trends in industrial digitalization, known as Industry 4.0, were established. According to Vogel-Heuser et al. [Vog+15c], the proportion of software in automated production systems is increasing and the demand for

highly customizable production systems will require higher involvement of multiple engineering disciplines. Consequently, research demonstrators in Industry 4.0 are required to study evolution cycles in these heterogeneous environments.

There exist demonstrator systems for Industry 4.0 environments targeting specific problems where the automation aspect is dominant [VH16]. The software systems interacting with the physical parts of these prototypes do not comprehend the complexity of information system in real-world scenarios. Additionally, a community case study is supposed to be a standardised or at least widely used reference for projects with the same research topics. This requires a demonstrator that not only is easily accessible and expendable but also comprehends the most significant aspects of evolution in Industry 4.0 scenarios. Therefore, both case studiesCoCoME and xPPU are integrated as Industry 4.0 case study.

4.2 Introduction of the CoCoME Case Study

CoCoME represents a trading system as it can be observed in a supermarket chain handling sales. This includes processing sales at a single store of the chain, like scanning products or paying, as well as enterprise-wide administrative tasks, like inventory management or reporting. Each store of the CoCoME supermarket chain contains several cash desks, whereas the set of cash desks is called cash desk line (visualised by the dashed line) (Fig. 4.3). The cash desk is the place where the

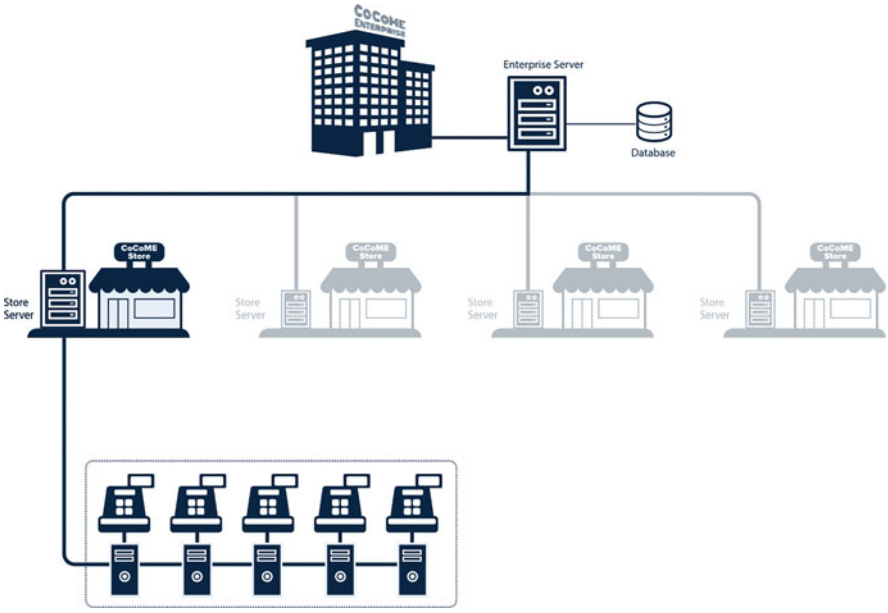

Fig. 4.3 Overview of the CoCoME structure

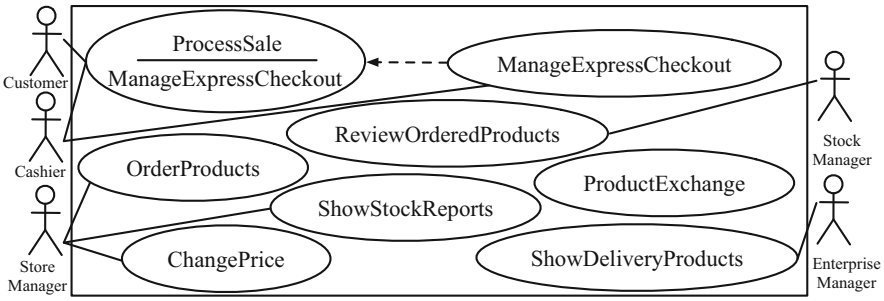

Fig. 4.4 Basic use cases of CoCoME [Her+08b]

cashier scans the goods that a customer wants to buy. The central unit of each cash desk is the cash desk PC. The cash desk line is connected to a store server. A set of stores is organised in the CoCoME enterprise where an enterprise server exists to which all stores are connected.

A detailed description of the basic use cases supported by CoCoME is given in [Her+08b]. In the *ProcessSale* use case, the cashier detects the products that a customer wants to buy and payment is performed at the cash desk (Fig. 4.4). If the conditions for express checkout [Her+08b] are fulfilled, a cash desk automatically switches to express mode in the *ManageExpressCheckout* use case. Product items can be ordered by the store manager in the *OrderProducts* use case. In the *ReceiveOrderedProducts* use case, products that arrive at the store are checked and inventoried by the stock manager. The store manager generates stock-related reports in the *ShowStockReport* use case. The *ShowDeliveryReports* use case provides the mean times a delivery from each supplier to a considered enterprise takes to the enterprise manager. The store manager can change the sales price of a product in the *ChangePrice* use case. In the *ProductExchange* use case, products are shipped from one store to another if a store runs out of a certain product. In this use case, no human actor is involved. Only the system is involved.

CoCoME uses Java SE in combination with Java Database Connectivity (JDBC), Java Persistence API (JPA), and Java Message Service (JMS) (Fig. 4.5). JMS is used to provide a way for communication between the components. The main component is the TradingSystem component. It consists of the TradingSystem::CashDeskLine component and the TradingSystem::Inventory component. The TradingSystem::CashDeskLine in turn consists of several CashDesk components representing the physical cash desks in a store with their corresponding components like the CashBox, BarcodeScanner, and CardReader. There is one Coordinator component per store, which receives sales events from the cash desks and changes the express mode state if needed. The TradingSystem::Inventory consists of the Console component, which provides a user interface for store-related operations through its Store component. The Console::Reporting component provides the user interface to retrieve enterprise or store reports. The central component of the TradingSystem::Inventory is the Application component. It provides the cash desk

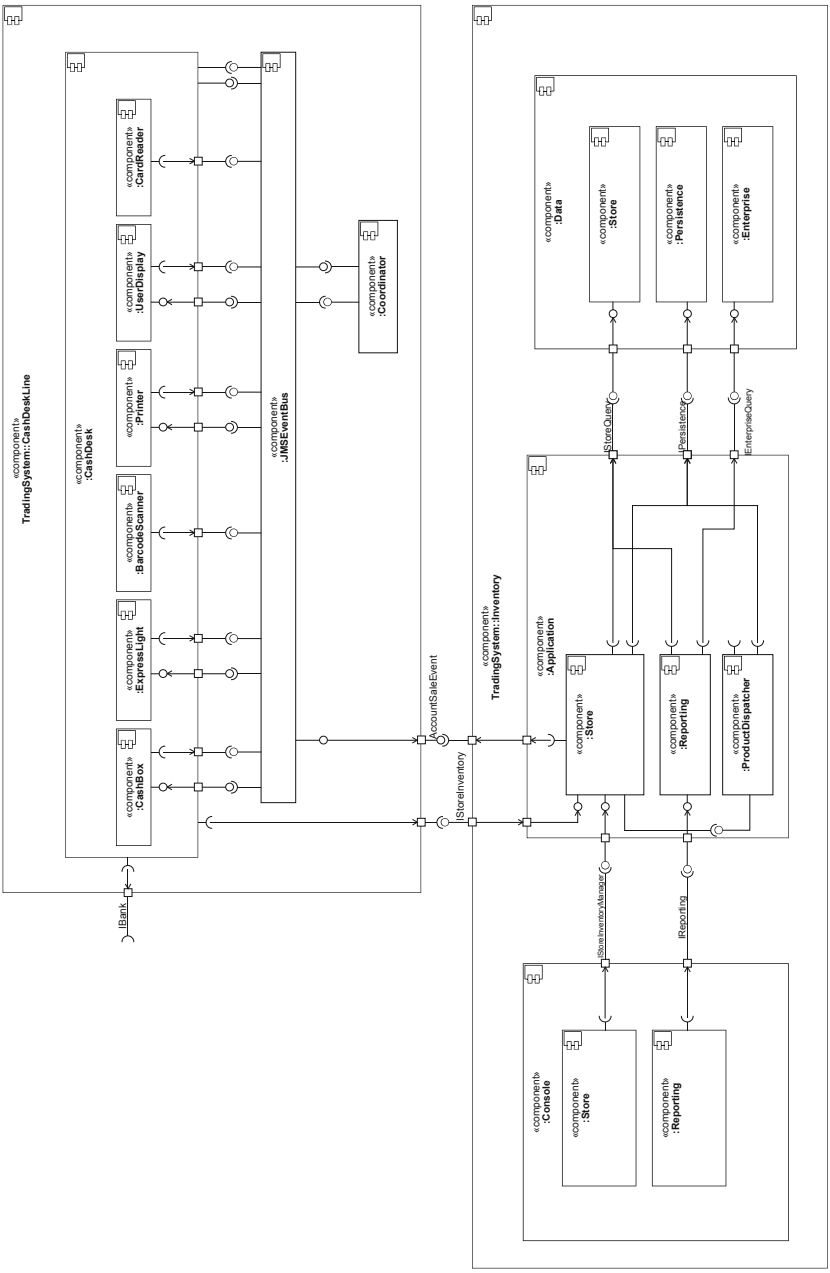

Fig. 4.5 Initial software architecture of CoCoME [HRR16]

and the store user interface and the operations to retrieve data and to book sales. The data are transferred in the form of Transfer Objects to provide an abstraction layer between the database and the other components. To retrieve the reporting information for the presentation layer, the Application::Reporting component provides the needed interface. There is also a ProductDispatcher component available to dispatch needed stocks from one store to another if necessary. A connection to the underlying database is realised by the Data component, which relies on JDBC and JPA to persist and retrieve data. It is divided into three sub-components, Store, Enterprise, and Persistence. The Store and Enterprise components are only used to query store or enterprise data, whereas the Persistence component writes objects to the database.

A detailed description of the initial requirements, architecture, and system behaviour in form of sequence diagrams is given in [Her+08b]. In the course of the DFG Priority Programme 1593, CoCoME faces changes by various evolution scenarios. Detailed description of changes to requirements, architecture, and system behaviour is given in [HRR16].

Since CoCoME has been applied and evolved successfully in various research projects, like SLA@SOI and Q-Impress funded by the European research council, several variants exist that span different platforms and technologies. Furthermore, various development artefacts are available, such as requirement specification or design documentation, that changed over time. CoCoME is well suited to serve as a study subject because the supermarket context is commonly comprehensible and the complexity of the system is appropriate. As CoCoME is a distributed system, several quality properties are affected by evolution.

In SPP 1593, a hybrid cloud-based variant of CoCoME has been developed based on the initial CoCoME specification [Her+08b] by implementing various evolution scenarios. The frontend of the hybrid cloud-based variant of CoCoME uses Java Server Faces (JSF) to implement the user interface (Fig. 4.6). In the WebFrontend::UseCases component, the presentation logic is implemented, which uses the components in the TradingSystem component to store the data retrieved from the ServiceAdapter. The ServiceAdapter component defines and implements an interface for database access and internally uses JDBC and JPA to access the underlying database. To query the database, the ServiceAdapter provides a Representational State Transfer (REST) style interface over Hypertext Transfer Protocol (HTTP).

Additional abstraction layers are introduced for the communication between the presentation layer and the business logic. These layers are located in the WebService component. The inner structure of the TradingSystem was nearly left unchanged. One exception is the event bus. Instead of the JMS event bus, the Context and Dependency Injection (CDI) event bus is used. Another change is that the components in Data now use the ServiceAdapter instead of the database directly. This allows for more flexibility in the cloud context. The newly introduced WebService::CashDesk component provides the frontend with a way to access the cash desk components. It is designed as a wrapper around the business logic so the method of accessing the business logic can be exchanged just by exchanging the wrapper classes. This is also the purpose of the WebService::Inventory component.

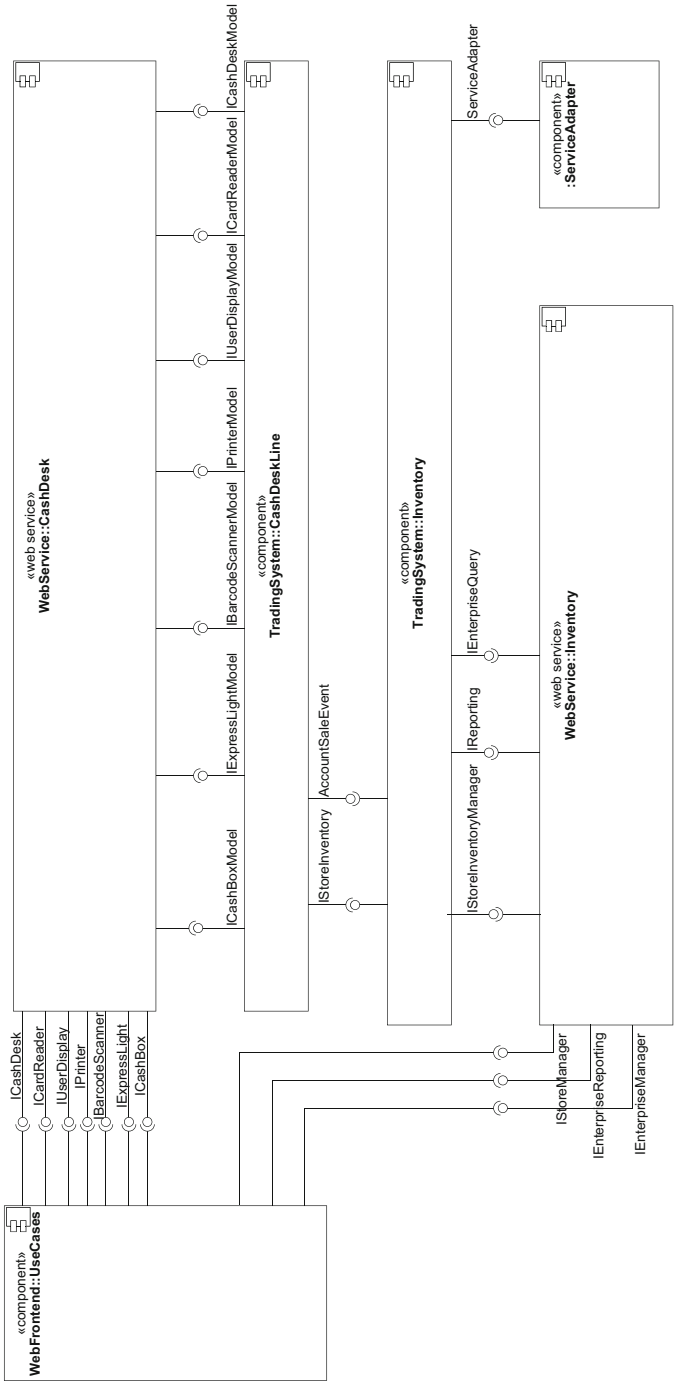

Fig. 4.6 Architecture overview of the hybrid cloud-based variant of CoCoME (coarse structure) [HRR16]

The WebService::Inventory contains the Enterprise component to enable the frontend to access enterprise-related information. This is necessary to enable several tasks needed for database administration like the listing of all stores in a specific enterprise. Design details are given in the technical reports [HRR16] and [HKR18].

We specified and implemented distinct evolution scenarios covering the categories adaptive and perfective evolution. Corrective evolution is not considered in the scenarios as this merely refers to fixing design or implementation issues. An adaptive evolution of the hybrid cloud-based variant of CoCoME is reflected in the scenarios Platform Migration, Microservice Architecture and Container Virtualization due to evolving technology. Perfective evolution is represented in the scenarios Adding a Pick-Up Shop, Adding a Mobile App and Adding Payment Methods by emerging user requirements. Furthermore, in order to accommodate the self-adaptiveness of modern software architectures, reconfiguration during system operation is addressed in the scenario Database Migration.

4.2.1 Platform Migration

The CoCoME enterprise must reduce operating costs of the resources and, therefore, migrates some resources to the cloud. The enterprise server and its connected database are now running in the cloud. The introduction of the cloud enables flexible adaptation and reconfiguration of the system. However, putting the system into the cloud causes new challenges regarding quality properties that must be considered in development and operation. For example, a look back in the recent past shows that privacy is one of the most important quality properties for cloud-based systems.

The evolution scenario Platform Migration transfers the initial variant of CoCoME to the hybrid cloud-based variant. As mentioned before, for the design of the hybrid cloud-based variant, additional abstraction layers are introduced for the communication between the presentation layer and the business logic. These layers are located in the WebService::CashDesk and WebService::Inventory components. The WebService::CashDesk component provides the frontend with a way to access the cash desk components. The WebService::Inventory component enables the frontend to access enterprise-related information. Wrappers are designed around the business logic, so the method of accessing the business logic can be exchanged just by exchanging the wrapper classes.

4.2.2 Adding a Pick-Up Shop

In this scenario, an online shop is added where the customers can order online and pick up the goods at a chosen store. This design-time modification includes adding new use cases and modifying existing design models.

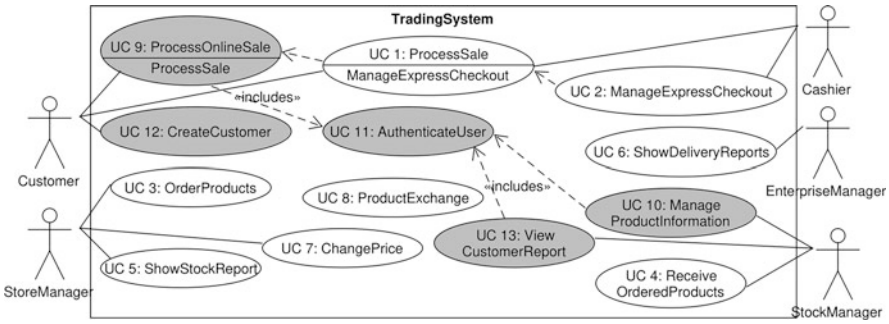

Fig. 4.7 Use cases of the pick-up shop [HRR16]

The CoCoME enterprise is in competition with online shop vendors (such as Amazon and Taobao). In order to increase its market share, the CoCoME enterprise management decides to offer a pick-up service for goods to address emerging customer requirements. The customers can order and pay online. The goods are delivered to a pick-up place (i.e. a store) of her/his choice. If the order has not been paid online, the goods have to be paid at the pick-up place (either per credit card or cash).

Existing use cases are extended, and new use cases are added to cover the pick-up shop's functionality (Fig. 4.7). In the *CreateCustomer* use case, a customer creates a new customer account for the pick-up shop. Users can be authenticated at the pick-up shop by the use case *AuthenticateUser*. The use case *ProcessOnlineSale* extends the existing use case *ProcessSale* by enabling a customer of the shop to select the products he/she wants to buy and to perform payment via credit card. Product information stored in the system can be changed by the stock manager in the *ManageProductInformation* use case.

For implementing the pick-up shop, the hybrid cloud-based variant had to be modified and extended to fit the needs arising from an online shop (Fig. 4.8). The first extension is to implement a service for customers to register and log in. This functionality requires the ServiceAdapter to store the login information and additional data like credit card data and the customer's preferred store in the data store.

The second modification is to include the services for the creation, modification, and authentication of customers into the business logic tier. To this end, the Inventory component is extended by a new UserManager component. This component implements the communication with the ServiceAdapter to retrieve, modify, or create the user and customer data.

The ShoppingCart component keeps track of all items that the customer wants to buy and is responsible for calculating the total price of these items. When the customer is done adding items to the ShoppingCart and proceeds, the sale is persisted by the CheckOut component.

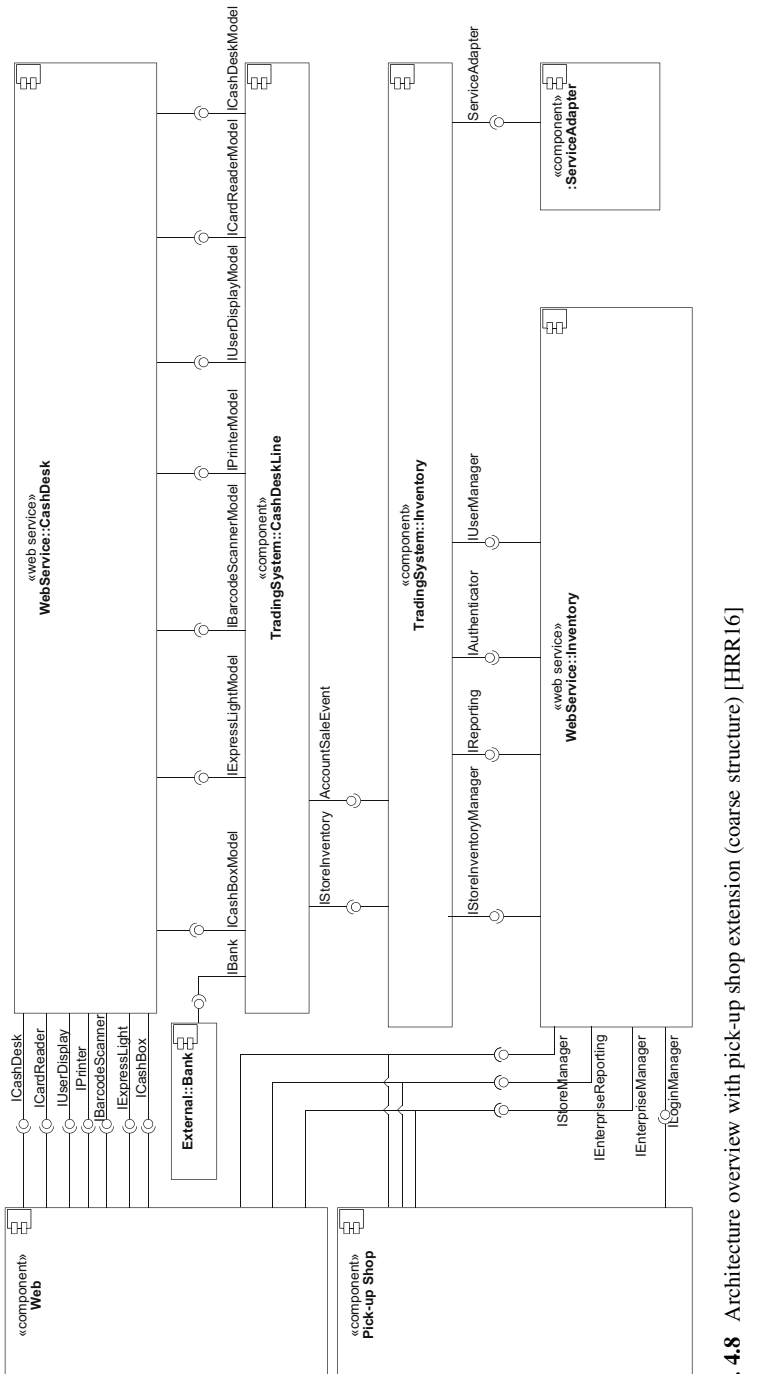

Fig. 4.8 Architecture overview with pick-up shop extension (coarse structure) [HRR16]

By introducing the pick-up shop as web application, the CoCoME system changes from a closed system (only employees can access but not the customers, and access depends on the location, e.g. a store) to an open system (all customers can accessed via the Internet). This raises certain consequences such that the number of users is not restricted any longer. Hence, various quality properties are affected, for example privacy, security, performance, and reliability.

4.2.3 Database Migration

After a while, the CoCoME enterprise starts a big advertisement campaign. Advertisements lead to an increased amount of sales. Thus, the performance of the system may suffer due to limited capacities of the cloud provider currently hosting the enterprise database. Migrating the database from one cloud provider to another may solve the scalability issues.

Especially in the cloud, the application usage, performance, pricing, and privacy are closely interrelated. The application usage impacts on the application's performance and pricing. Continuously appraised elasticity rules trigger the migration and replication of a cloud application's software components among geographically distributed data centers. Both migration and replication may lead to violation of privacy policies that prescribe certain geo-locations. Furthermore, a cloud application may also face performance/availability trade-offs as replication is often done for improving the system's overall availability, not just performance, which again might violate privacy policies.

This scenario represents a reconfiguration at runtime. Migrating the database may cause a privacy issue due to violations of privacy constraints. According to a privacy constraint[1] of the European Union (EU), sensitive data must not leave the EU. Since the CoCoME enterprise is located within the EU, its databases containing customer data must be hosted on data centers within the EU. This scenario is about the dynamic analysis of cloud applications at runtime to identify upcoming quality flaws. It includes model-based observation and prediction techniques in flexible environments.

4.2.4 Adding a Mobile App Client

In order to outperform its competitors and expand its market share, the CoCoME enterprise decides to offer a mobile app client. In this perfective evolution scenario, a mobile client is added where the customers can order through their mobile phones and pick up the goods at a chosen store. Figure 4.9 depicts the mod-

[1] http://eur-lex.europa.eu.

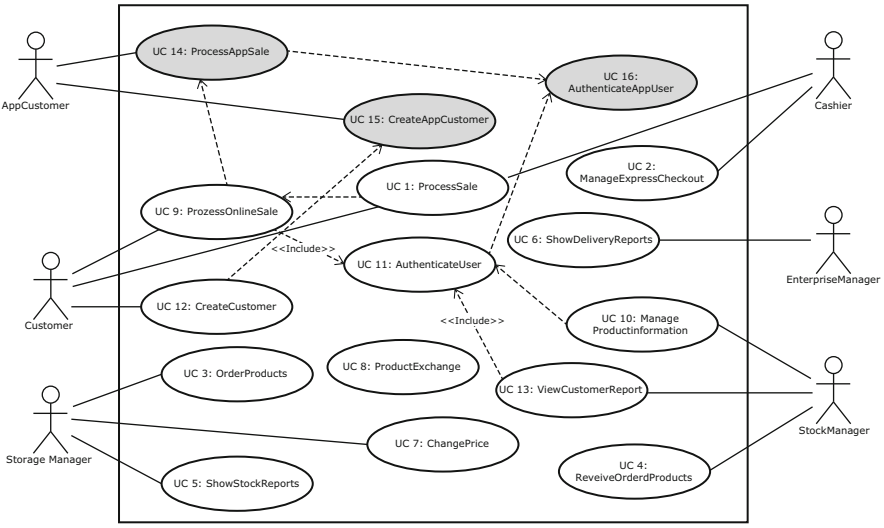

Fig. 4.9 Use cases of the mobile app [HKR18]

ifications regarding the use cases of CoCoME. In the *CreateAppCustomer* use case, a customer creates a new customer account for the mobile app. Customers can be authenticated on the app by the use case *AuthenticateAppUser*. The use case *ProcessAppSale* extends the existing use case *ProcessSale* by enabling the customer to buy products using a mobile app. This scenario introduces mobile communications to the CoCoME system, which may affect quality properties like privacy, security, performance, and reliability.

This design-time modification is based on the pick-up shop scenario but implements a mobile frontend (Fig. 4.10). The backend of CoCoME does not face any changes. An AppShopAdapter is introduced to bridge the technology gap between the web services of CoCoME and the technology used by the mobile app client. The AppShopAdapter consumes the three web services WebService::Inventory::LoginManager, WebService::Inventory::Store, and WebService::Inventory::Enterprise and provides a Rest API, which is used by the AppShop. The Rest API contains the service endpoints. Further design details are described in the technical report [HKR18].

4.2.5 Microservice Architecture

The architectural style of CoCoME is changed to microservices for reducing the coupling between the services of CoCoME and enable independent deployment, as well as reuse. In this adaptive evolution scenario, the functionality of CoCoME remains the same while changing the architectural paradigm of the system. This

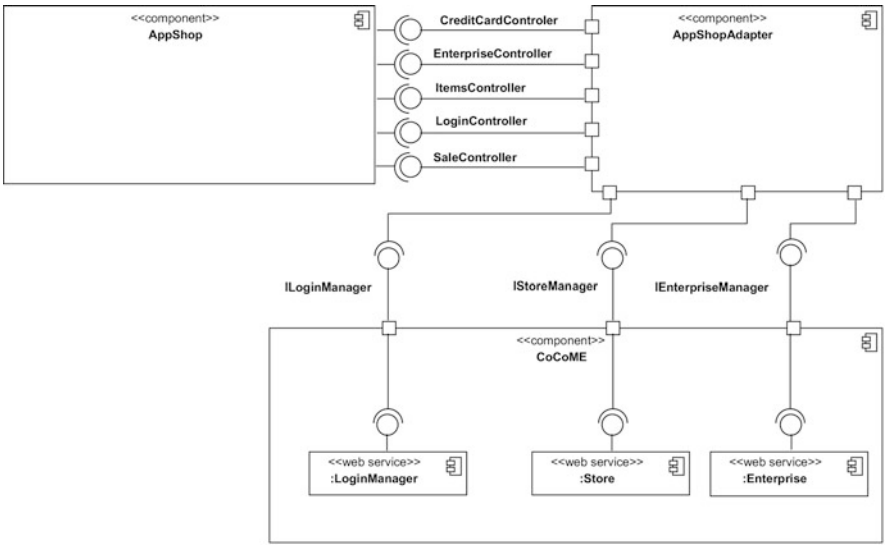

Fig. 4.10 Architecture overview with mobile app extension (coarse structure) [HKR18]

design-time modification introduces a collection of loosely coupled microservices where each microservice is internally structured in a layered fashion. Each microservice has its own graphical user interface (GUI) and business logic. Each microservice, except for Reports, has its own database. The CoCoME system (before structured by technical layers) is decomposed into four microservices for managing the orders, reports, stores, and products (Fig. 4.11).

In addition to the four microservices, a Frontend service is introduced. The Frontend service is required to provide a unique GUI and entry point for users. When a user requests a service, for example by clicking a button on the GUI, the request is delegated by the Frontend service to the corresponding microservice. Thus, the Frontend service handles the orchestration of the microservices. Furthermore, the Frontend service is responsible for identity and access management.

The evolution of the architectural style shifts complexity from software design into system operations. While the individual complexity of a microservice is reduced compared to intermeshed services, additional complexity is introduced for the orchestration of the single microservies. Moreover, quality properties like performance and privacy are affected by this evolution scenario.

4.2.6 Container Virtualization

In this scenario, the deployment and operation of the CoCoME system is facilitated by introducing container-based virtualization with Docker. Docker eases the integration of CoCoME into build and deployment pipelines. In this adaptive evolution

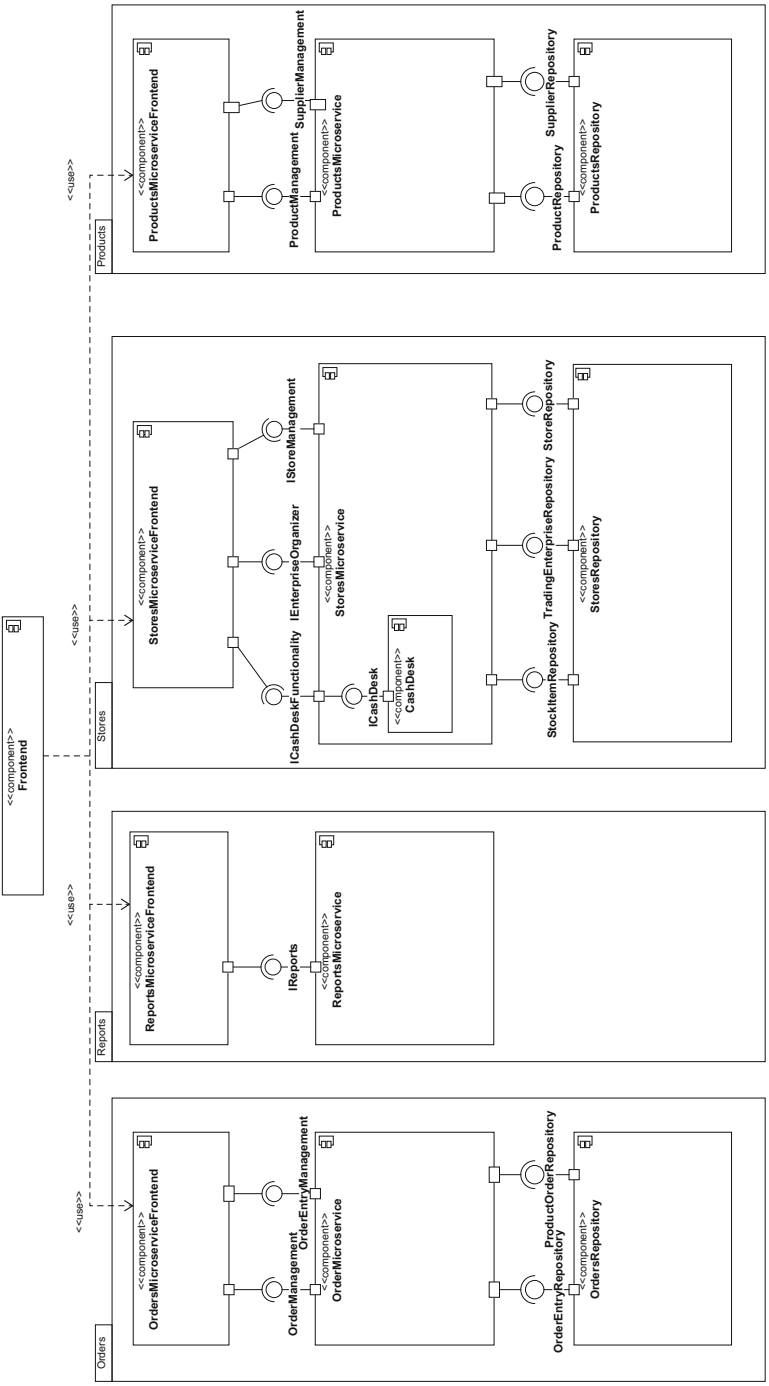

Fig. 4.11 Overview of the microservice architecture of CoCoME [HKR18]

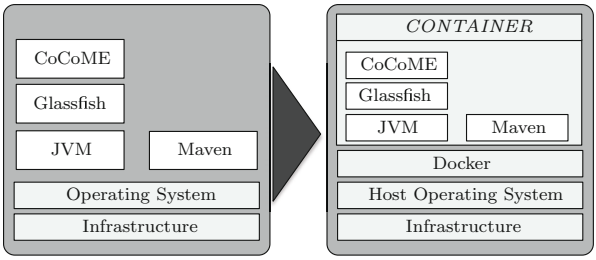

Fig. 4.12 Extended technology stack CoCoME [HKR18]

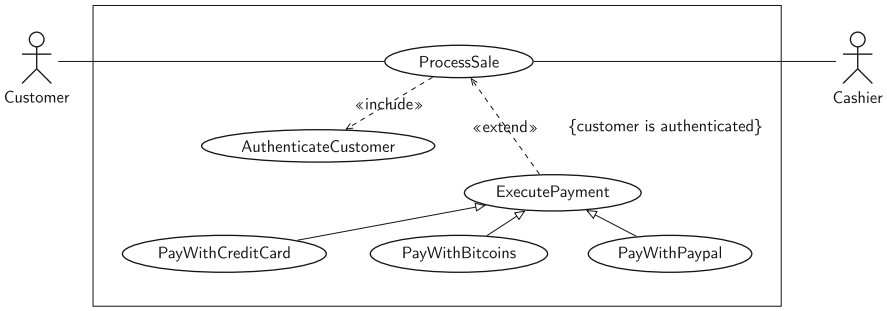

Fig. 4.13 Use cases of CoCoME payment possibilities

scenario, the functionality of CoCoME remains the same, while the technology stack is extended (see Fig. 4.12). The given CoCoME stack is moved into the Docker Daemon, which runs a Linux distribution. This evolution scenario provides a platform independent CoCoME that does not require any preconditions like installing or updating software. By using Docker, a version of CoCoME can be instantiated on any device without installing additional software. Furthermore, the building and deployment of CoCoME can be automated and be sped up.

4.2.7 Adding Payment Methods

Currently, customers can only pay via credit card. In this scenario, the CoCoME sales systems is extended with new payment possibilities such as PayPal and Bitcoins (Fig. 4.13).

Customers are then enabled to select between various payment options. Payments are initiated by the TradingSystem::CashDeskLine component (Fig. 4.8). This component communicates with an external bank (External::Bank::TrivialBankServer component) via the IBank interface. The IBank interface defines the methods *validateCard* and *debitCard*. In this scenario, a generic IPayment interface is introduced

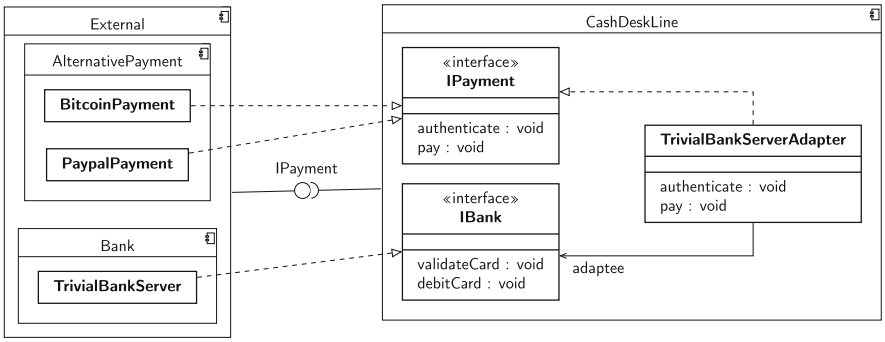

Fig. 4.14 Excerpt of the CoCoME architecture after adding new payment possibilities

that defines the *authentication* and *payment* methods. The IPayment interface is implemented by the PaypalPayment and the BitcoinPayment components (as part of the External::AlternativePayment component). In order to still provide the customer with the possibility to pay via credit card, the IPayment interface needs to be mapped to the IBank interface. For this purpose, the adapter design pattern is chosen. Figure 4.14 shows the resulting architectural structure.

4.3 Introduction of the PPU and xPPU Case Studies

The PPU and xPPU represent a laboratory plant for automated production system. The case studies handle and manipulate workpieces (WP) of different material (Fig. 4.15). An order for WPs is initially processed at a material storage. Afterwards, the PPU and xPPU distribute and manipulate the WPs that are detected by many different kinds of control hardware. Finally, WPs are sorted based on their material in product storage and delivered.

The original PPU consists of four equipment modules: stack, crane, stamp, and conveyor. WPs, which are the target of the process of the plant, are stored at the stack. These WPs are processed differently depending on their type in which the WP is either directly transported to the conveyor by the crane or moved to the stamping unit followed by transported to the conveyor. Sorting ramps possibly differ also depending on the WP type. The xPPU have additional features, such as a reordering module for logistic flexibility, so-called picalpha; reinforced security and safety; product recognition using radio-frequency identification (RFID); manual operating mode; and Industry 4.0 interface (Fig. 4.16).

In the xPPU, production, material, and product managers are responsible for controlling the plant status, material status, as well as the resulting product status respectively (Fig. 4.17). When the order is initialised, the production manager control the processes the plant conduct regarding the material and resulting product. The processes (i.e. basic control functionality) can be manipulating WP (i.e. stamping)

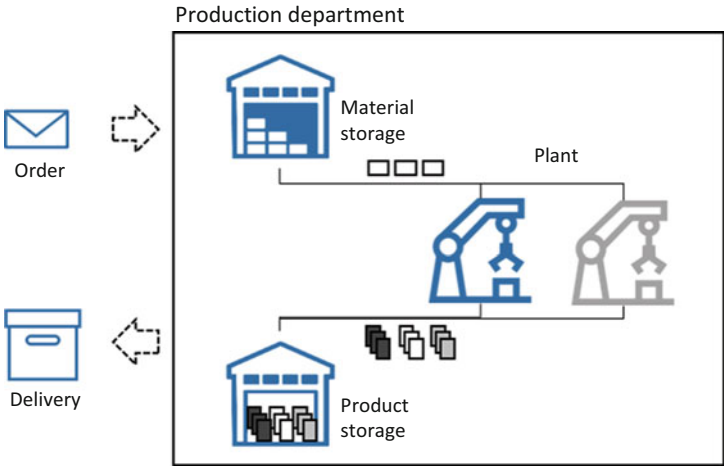

Fig. 4.15 Overview of the PPU and xPPU production chain

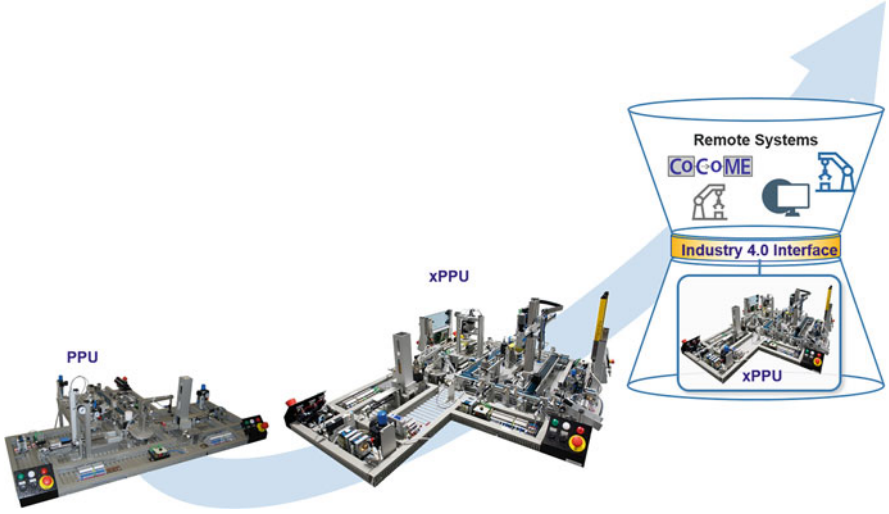

Fig. 4.16 Overview of the PPU, xPPU, and Industry 4.0 case studies

or sorting WP in different ramps according to their material. The plant operator is responsible for selecting the mode of operation for the plant (i.e. manual or automatic), as well as monitoring the status of the plant regarding fault and emergency handling. By enabling Industry 4.0 interface to the xPPU, remote operator and system can access the plant over the web or mobile application and execute processes. The use case for the Industry 4.0 interface is further detailed in Sect. 4.4.1.

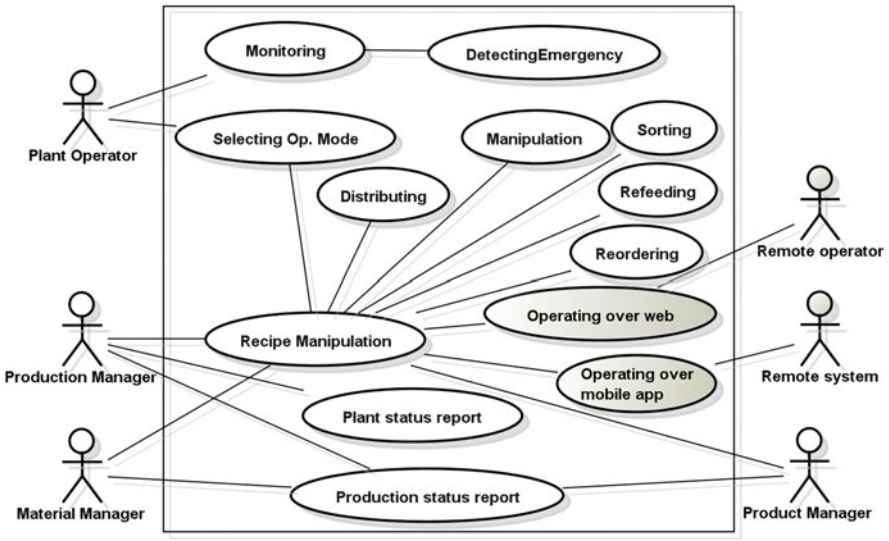

Fig. 4.17 Basic use cases of xPPU (highlighted use cases are from enabling Industry 4.0 interface)

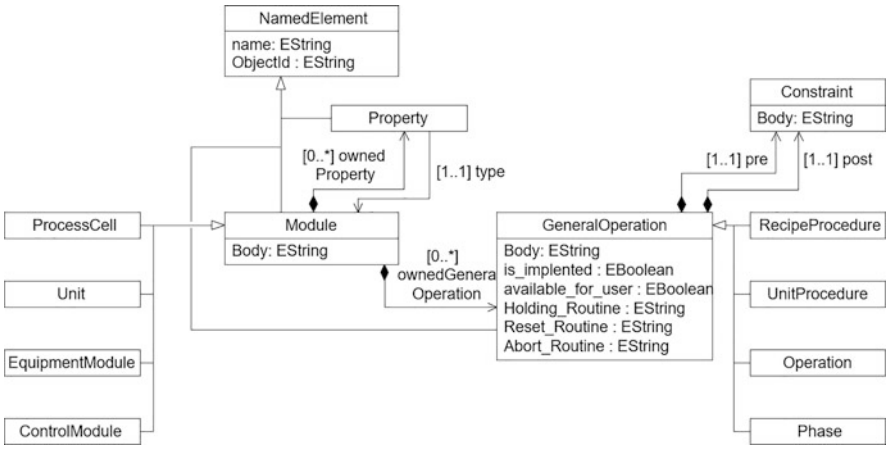

Fig. 4.18 Excerpt of the meta-model developed based on ISA88 [Com+95]

The meta-model used for modelling the system architecture of the xPPU is based on ISA88 standard [Com+95] (Fig. 4.18), which is a reference model for providing the essential fundamentals for batch process control (Fig. 4.19). The two central classes are the Module and the GeneralOperation. The Module corresponds to the physical assets of an enterprise. The GeneralOperation represents the procedural elements from the ISA88 standard. Both classes inherit the name and ObjectId from the NamedElement class. The Module can have multiple relations to the class

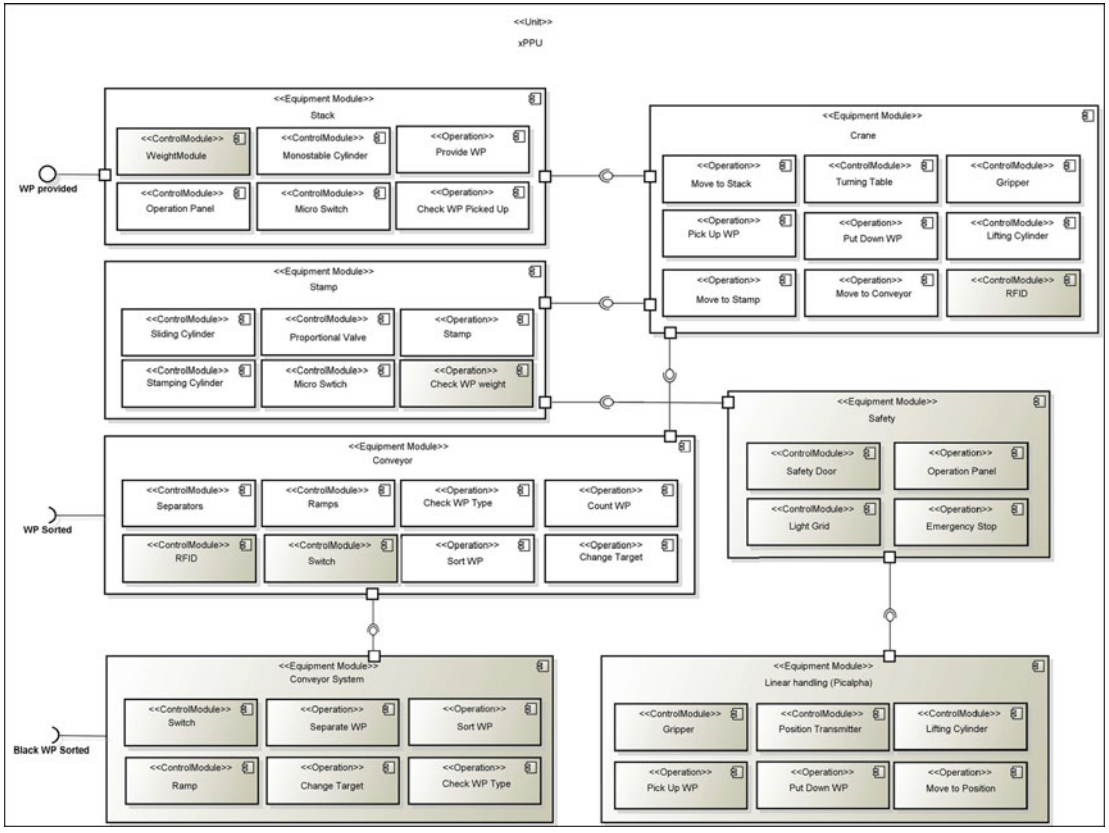

Fig. 4.19 Functional and structural view on the system architecture of PPU and xPPU (highlighted are the extended parts)

Property, which refers through a type relation to an existing Module. Moreover, a Module, which is already available and implemented in the overall model, can thus be referenced and used several times, which allows the creation of a repository (i.e. area repository). In addition, the Module is also related to GeneralOperation, which allows the procedural elements to be assigned to the physical plant components. GeneralOperation has two compositional relations to Constraint via a pre and post relation, which respectively express pre-conditions and post-conditions of procedural elements to be checked before and after the execution of a GeneralOperation. Furthermore, the physical assets in terms of process cells, units, equipment modules, and control modules inherit from the class Module. In the same manner, procedural elements in terms of recipe procedure, unit procedure, operation, and phase inherit from the class GeneralOperation.

In order to provide use cases and allow the comparison of different solutions, 24 evolution scenarios have been developed during the SPP 1593. Detailed documentation of the xPPU evolution scenarios are documented with structural and behavioural models, PLC control code, Matlab/Simulink simulation projects, and mechanical CAD files [Vog+14b] and are publicly available to the community on github.[2] The evolution scenarios were extended regarding more sophisticated requirement modelling, as well as fault handling functionality Chap. 12.

The xPPU is also used by the research community outside the SPP1593 to identify inconsistency [Fel+16], control parameter optimization [Zou+18], and in-place traceability [Ale+17]. The xPPU is connected to a PLC through EtherCAT, allowing the process of signals from the xPPU and the control of actuators accordingly (Fig. 4.20). The control software runs on a PC with a particular environment (e.g. CODESYS or TwinCAT). Furthermore, the PLC is connected to an Open Platform Communications (OPC) server that allows accessing the plant by OPC clients such as the Industry 4.0 interface.

4.3.1 Evolution Scenarios of the PPU

In Table 4.1, 13 sequential evolution scenarios of the PPU are depicted, which cover different combinations of platform, context, and software changes.

- Scenario Sc0: The initial scenario is the evolution scenario Sc0 where the stack, the crane, and a slide exist. The stack pushes a single black plastic WP out of the stack into the crane's pick-up position. At the pick-up position, the crane picks up single WPs by moving the crane down and by using a vacuum gripper to suck the separated WP. Upon rotation of 90°, the crane reaches the slide's position, where the WP has to be placed. After moving down, the vacuum gripper releases the WP, which then glides down the slide.

[2] https://github.com/x-PPU.

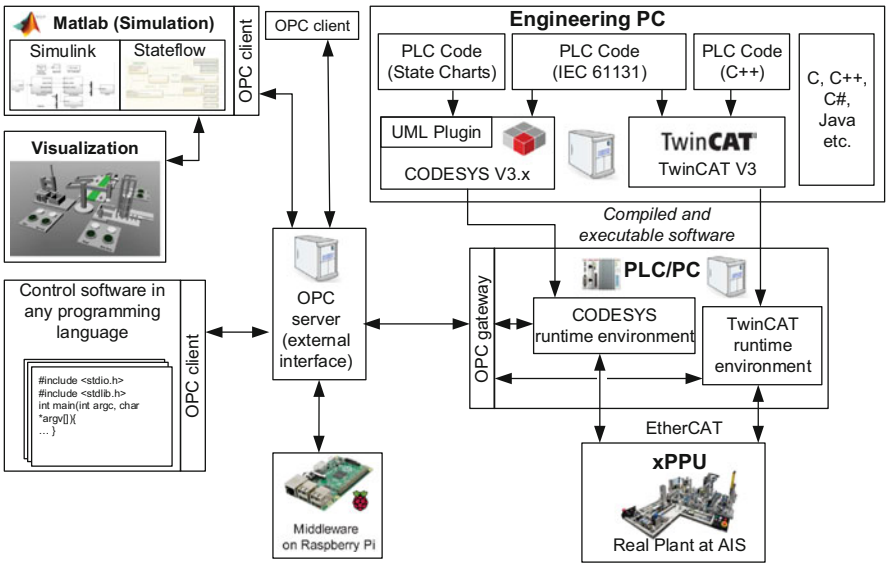

Fig. 4.20 Environment frame of engineering and access to xPPU

- Scenario Sc1: Within this scenario, the slide was replaced with a Y-shaped slide to increase the capacity of the slide to five WPs. the evolution in this scenario affect only the context as solely mechanical component was added.
- Scenario Sc2: Within this scenario, the PPU processes black plastic WPs, as well as metallic WPs. In order to distinguish between the processed WPs, an additional inductive sensor was installed at the stack.
- Scenario Sc3: For tractability reasons, a stamp module was added within this scenario to allow the labelling of the WPs. The stamp is located at position 180° of the crane. Once the WP is detected at the stack, the crane picks up the WP and turns to the stamp position to place it at the magazine, which then retracts to position the WPs under the stamp. The stamp moves down to press the WPs for a while and retracts. The magazine extrudes, and the crane then picks the WPs and place them at the slide. The evolution in this scenario results in modification of all dimensions of the crane.
- Scenario Sc4a: To increase availability, inductive sensors are installed for crane positioning replacing micro switches as they are more robust against pollution. The inductive sensors provide the same signals as the old position sensors. Therefore, the software is not affected; only the crane platform is modified.
- Scenario Sc4b: To increase the reliability of crane positioning, micro switches were installed in addition to the existing inductive sensors. As a result, each inductive sensor has a redundant micro switch.
- Scenario Sc5: Within this scenario, the crane behaviour is changed to allow the processing of more than one WP at a time. As soon as the crane places a metallic WP at the stamp, the stack checks if a plastic WP is available for pickup. In this

Table 4.1 Evolution scenarios of the PPU

Sc	Cause of evolution	Stack			Crane			Stamp			Ramp			Conveyor			Realization
		C	P	S	C	P	S	C	P	S	C	P	S	C	P	S	
0	Increasing throughput of workpiece	*A*	*A*	*A*	*A*	*A*	*A*	–	–	–	*A*	*A*	*A*	–	–	–	New development of the machine
1	Increasing capacity of the output storage	o	o	o	o	o	o	–	–	–	*M*	o	o	–	–	–	Y-shape of the ramp
2	Additional processing of metallic workpieces	*A*	*A*	*A*	o	o	o	–	–	–	o	o	o	–	–	–	Inductive sensor
3	Labelling of a workpiece	o	o	o	*M*	*M*	*M*	*A*	*A*	*A*	o	o	o	–	–	–	Addition of a stamp unit
4a	Decreasing failure due to sensor pollution	o	o	o	o	*M*	o	o	o	o	o	o	o	–	–	–	Replacement of crane sensors
4b	Increasing reliability of crane positioning	o	o	o	*M*	*M*	*M*	o	o	o	o	o	o	–	–	–	Sensor redundancy through inductive sensor and micro switch
5	Increasing throughput of workpiece	o	o	o	o	o	*M*	o	o	o	o	o	o	–	–	–	Optimised crane behaviour through the use of the stamp as a buffer
6	Further increasing in throughput of workpiece	o	o	o	*M*	*M*	*M*	*A*	o	o	o	o	o	–	–	–	Additional mechanical buffer
7	Recognition of additional workpiece	*A*	*A*	*A*	o	o	o	o	o	o	o	o	o	–	–	–	Installing of a light sensor
8	Different processes for different workpieces	o	o	o	o	o	o	*A*	*A*	*M*	o	o	o	–	–	–	Different pressure profiles at the stamp unit

9	Logistics optimisation	o	o	o	o	o	o	o	o	o	*M*	*M*	*M*	*A*	*A*	*A*	Replacement of the Y-slide by a conveyor belt with individual ramp
10	Increasing output storage	o	o	o	o	o	o	o	o	o	o	o	o	*A*	*A*	*A*	Two additional ramps on the conveyor belt
11	Different storage locations	o	o	o	o	o	o	o	o	o	o	o	o	*M*	*M*	*M*	Successive filling of ramps
12	Correct sorting of workpieces in ramps	o	o	o	o	o	o	o	o	o	o	o	o	o	o	*M*	Specified sorting of the workpiece
13	Increasing the precision of the crane	o	o	o	*M*	*A*	*M*	o	o	o	o	o	o	o	o	o	Analogue sensor

Sc Scenario, *A* Added, *M* Modified, *o* no changes, *C* Context, *P* Platform, *S* Software

case, the crane uses the stamping time to transport the plastic WP to the slide. The realised evolution only affects the software of the crane.

- Scenario Sc6: Within this scenario, a mechanical buffer was mounted next to the stamp, which allows another metallic WP to be placed next to the stamp even if the stamp is processing metallic WP. The behaviour of the crane is similar to Sc5.
- Scenario Sc7: Within this scenario, the PPU processes additional white plastic WP. Therefore, the stack was modified with additional optical digital sensor to detect the brightness of the WP. Combining the already existing inductive sensor with the new sensor, the stack is able to differentiate all kinds of WPs. The white plastic WPs are stamped like metallic ones. Black WPs are transported directly to the slide.
- Scenario Sc8: Due to the fragility of white plastic WPs compared to metallic ones, the stamp was modified with two different pressure profiles each for specific types of WP. Therefore, the present two-way valve was replaced by a proportional valve that handle analogue values.
- Scenario Sc9: Within this scenario, a conveyor was installed in the place of the slide. The crane now places the WPs directly on the conveyor, which transports WPs to a slide mounted at the end of the conveyor.
- Scenario Sc10: Additional two output slides were added in this scenario at the side of the conveyor. Therefore, to separate the WPs, two pneumatic pushers are mounted at the opposite side of the conveyor, facing towards the two slides. Right before each pusher, an optical sensor is attached to detect whether a WP is available. The slide mounted at the end of the conveyor is filled first, then the mid slide, and finally the slide at the beginning of the conveyor (first slide).
- Scenario Sc11: Within this scenario, only one type of WPs is separated into one slide. Therefore, two inductive sensors are installed in front of the optical sensors right before the two slides on the side. In the first slide, white WPs are separated; on the mid slide, metallic WPs are separated; on the last slide (at the end if the conveyor) black WPs are sorted.
- Scenario Sc12: In this scenario, the sorting order of WPs is changed at slides. Now WPs have to be mixed in all slides. The change in this scenario only affects the software.
- Scenario Sc13: Until this scenario, the positioning of the crane is done using digital position sensors. To increase the accuracy and to avoid spending cables and terminal blocks, the digital sensors are replaced by analogue sensors (potentiometer).

4.3.2 Incremental Evolution Scenarios

Within scenario 11, the sorting of WPs at the conveyor is targeted. The scenario follows a specific sorting regarding the WP types. The first ramp collects white WPs, the second ramp collects metal WPs, and the third ramp collects black WPs. The change is implemented by checking the type of WP with a diffuse and inductive

sensor after the WP is placed on the conveyor belt. When a white WP is identified, the first pneumatic cylinder pushes the WP in the first ramp, and after another sensor check the metal WPs are pushed in the second ramp by the next cylinder. Black WPs pass both the lateral ramps and are transferred to the ramp at the end of the conveyor. According to the categories of changes, scenario 11 is a change of category 5 without adaption of the requirement and specification.

Because of the various possibilities to sort WPs, this scenario is used to implement so-called mini-scenarios. Mini-scenarios are evolution scenarios of the PPU that only have a very limited impact on the whole system. The mini-scenarios were introduced because some technologies such as verification are not feasible for large change. The mini-scenarios reflect ad hoc changes that are often instantaneously performed to quickly react to avoid standstills of the production system. The mini-scenarios are simplifications in the material flow of scenario 11 and are implemented as simple code adaptations of the software code. Table 4.2 shows the implemented mini-scenarios. Scenario 11a and 11b simplify the material flow of the PPU by exclusively using only one ramp. In scenario 11c, two ramps are used by an alternating pattern that arise a very unique material flow. Scenario 11d and 11e are preliminary stages to the original scenario 11 by sorting just one or two WPs in a specific ramp. The mini-scenarios can be used to investigate approaches that consider undocumented or unknown changes during operation. The platform and context are not affected by any mini-scenario, and the changes in the behaviour arising out of the software modification are much smaller than in the other evolution scenarios. Therefore, these scenarios can, for example, be used to evaluate approaches that try to ensure consistency between specifications, models, and the running system and consider transformation of models or focus on atomic modification steps.

Table 4.2 Mini-scenarios: limited software modifications of the sorting of conveyor belt

Scenario	Cause of evolution	Conveyor belt			Realization
		C	P	S	
11	Specific sorting regarding workpiece type	o	o	*M*	White workpieces are stored in Ramp 1, metal in Ramp 2, and black in Ramp 3
11a	Exclusive use of Ramp 1	o	o	*M*	All workpieces are stored in Ramp 1
11b	Exclusive use of Ramp 2	o	o	*M*	All workpieces are stored in Ramp 2
11c	Use of two ramps	o	o	*M*	All workpieces are alternatively stored in Ramp 1 and Ramp 2
11d	Sorting of one workpiece type	o	o	*M*	White workpieces are stored in Ramp 1, and the others in Ramp 2
11e	Sorting of two workpiece types	o	o	*M*	White and black products are stored in Ramp 1 and the others in Ramp 2

M Modified, *o* no changes, *C* Context, *P* Platform, *S* Software

4.3.3 Evolution Scenarios of the xPPU

In Table 4.3, extended sequential evolution scenarios of the xPPU are depicted.

- Scenario Sc14: Within this scenario, the xPPU processes additional metallic WPs of different weights. Therefore, the stack was modified with a **weighting module** to distinguish between the processed WPs based on their weight. The introduction of new WPs will also affect the crane's behaviour. During the transportation of different WPs by the crane to the stamp or the conveyor, the heavier WPs need more time to stop oscillating after the crane's rotation. This latter modification can be adapted by modifying the software. Furthermore, the stamp is modified with different stamping pressures (e.g. heavy, medium, and light pressure).
- Scenario Sc15: To allow re-feeding of WPs that are detected as being faulty, the xPPU was extended with a **conveyor system** containing three additional conveyors.
- Scenario Sc16: Within this scenario, the xPPU allows the processing of WPs in priority at the conveyor system. A **picalpha module** was mounted on the first conveyor of the conveyor system, which has a handling module for reordering WPs by picking and placing the WP ahead in a different position.
- Scenario Sc17 and Sc18: With the evolution in these scenarios, the xPPU is extended with a **safety door** for the prevention of accidents at the stamp, as well as a **light grid** to prevent accidents at the picalpha module. The mounted hardware incorporates emergency stop buttons, as well as additional control elements.
- Scenario Sc19: The xPPU in this scenario has an additional control button to switch between automatic operating mode and the additional **manual mode**. Within the manual mode, the operator is allowed to control the xPPU in any required function sequence.
- Scenario Sc20: Within this scenario, the xPPU was extended with **energy monitoring hardware**. This hardware allow to measure the energy consumed by the different clamps. Therefore, three Wattmeter were installed. Not only does it allow to measure the electric energy consumed by the plant, but with flow sensor it is also possible to measure the air pressure and the air flow through the xPPU. This information can be used to optimise the plant focusing on energy-saving aspects. It also allows to monitor if some parts consume more energy than usual, which might lead to the conclusion that these parts have to be replaced due to malfunction.
- Scenario Sc21: Within this scenario, the xPPU has four **valve blocks** that allow to turn off the air flow on some xPPU modules. This feature allows to simulate failures in the air flow, as well as turn off the air pressure in specific hardware parts for safety reasons.
- Scenario Sc22: To enable more flexible production management, the xPPU was extended with two **RFID-Reader/writer**. One was mounted at the crane parallel to the gripper position, and the other on the conveyor belt. Each WP now was

Table 4.3 Evolution scenarios of the xPPU

Sc	Cause of evolution	Stack			Crane			Stamp			Ramp			Conveyor			xConveyor			Realization
		C	P	S	C	P	S	C	P	S	C	P	S	C	P	S	C	P	S	
14	Distinction of workpieces by their mass	*M*	*M*	*M*	○	○	○	○	○	○	○	○	○	○	○	○	–	–	–	Weighing module to enable the measurement of mass
15	Implementation of more complex processes	○	○	○	○	○	*M*	○	○	○	*M*	○	○	*M*	*M*	*M*	*A*	*A*	*A*	Conveyor system to enable re-feeding workpieces into the manufacturing process
16	Workpiece processing by priority at conveyor system	○	○	○	○	○	○	○	○	○	○	○	○	○	○	*M*	○	○	*M*	Implementing handling module
17	Prevention of human hazard situations at stamp	○	*M*	○	○	*M*	○	*M*	*M*	*M*	○	*M*	○	○	*M*	○	○	*M*	○	Implement safety door and emergency stop switching module
18	Prevention of human hazard situations at the conveyor system	○	*M*	○	○	*M*	○	○	*M*	○	○	*M*	○	○	*M*	○	*M*	*M*	*M*	Implement light grid and emergency stop switching module
19	Additional operating mode (manual)	○	○	*M*	○	○	*M*	○	○	*M*	○	○	*M*	○	○	*M*	○	○	*M*	Push buttons are added to allow manual operations

(continued)

Table 4.3 (continued)

Sc	Cause of evolution	Stack			Crane			Stamp			Ramp			Conveyor			xConveyor			Realization
		C	P	S	C	P	S	C	P	S	C	P	S	C	P	S	C	P	S	
20	Monitoring energy consumption (electrical energy and compressed air)	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	Additional energy monitoring hardware
21	Decreasing air pressure for fault detection and isolation	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	Additional valve blocks
22	Enabling more flexible production management	*M*	*M*	*M*	*M*	*M*	*M*	o	o	o	o	o	o	*M*	*M*	*M*	o	o	o	RFIDs are added to the crane and conveyor
23	Networked PLC, distributed platforms	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	o	*M*	*M*	One PLC per machine part, that is crane, stamp, and stack
24	Web connectivity of xPPU	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	Enable OPC connection to xPPU

Sc Scenario, *A* Added, *M* Modified, *o* no changes, *C* Context, *P* Platform, *S* Software

labelled with an RFID tag, which contains specific information about the WP (e.g. weight and date of labelling).

- Scenario Sc23: Within this scenario, the xPPU is controlled in a decentralised way by a set of (PLCs). **Four different controllers** where used. One for the crane, stamp, stack, and conveyor belt and three for each conveyor of the conveyor system.
- Scenario Sc24: Within this scenario, an Open Platform Communications Unified Architecture (**OPC UA**) connection was enabled to read and write data information from the xPPU to an online server. The OPC UA standard is a collaboration partner of IEC 61131-3 officially[3] and enables flexible process planning as a feature of Industry 4.0. The read and write data information is used to monitor and control the xPPU. Using OPC UA allows us to monitor many variables and values of the xPPU and still have a quick response time. Third-party programs can access the online server and, therefore, the data and use them for monitoring purposes, data gaining, and big data mining purposes. With specified algorithms, statements can be made on the reliability of hardware parts. Also, the usage of each hardware part can be optimised, reducing maintenance work, energy consumption and, therefore cost.

4.4 Industry 4.0 Case Study

In this section, we present the Industry 4.0 community case by integrating both community case studies xPPU and CoCoME (cp. Fig. 4.16). This new case study implements common use cases in Industry 4.0 environments, such as ordering a customizable product, creating a production plan for a customizable product on multiple abstraction layers, and observing the progress of batch size one productions. We enabled event-based communications between information system and automated production components by providing a web-service-based communication model. The Industry 4.0 case study allows to define and emulate the automated production systems in a web-based frontend. Moreover, an automated production system (i.e. the xPPU) is interconnected to an information system (i.e. CoCoME) by a REST-based web services.

4.4.1 Industry 4.0 Interface of the xPPU

aPS are mostly controlled by PLCs, which are programmed in order to execute specific tasks. A program code is loaded onto the PLC through dedicated connection with the computer (i.e. EtherCAT). The PLC then performs, in a cyclic execution,

[3] https://opcfoundation.org/markets-collaboration/plcopen/.

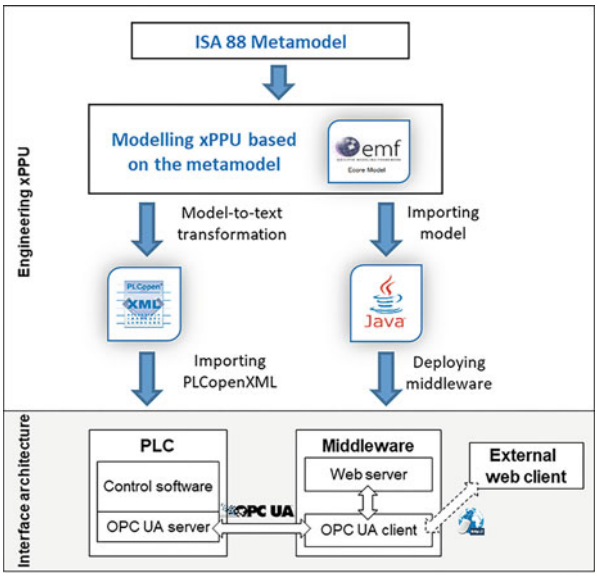

Fig. 4.21 Overview of the concept of dynamic reconfiguration and generation of the Industry 4.0 interface [Bou+17a]

the processing of internal operations, reading of inputs, execution of the program, and updating of outputs. In [Bou+17a], a design concept that enables flexible entry of orders in a production system during its operation, through dynamic reconfiguration of orders and process planning via a remote interface, was introduced. The developed concept allows dynamic services to the CoCoME via web services. The remote interface of the xPPU is considered as an evolution scenario that also affects the environment frame of the xPPU (cp. Fig. 4.20).

A model-based approach was used for the implementation of the PLC code, which controls the plant itself, and a middleware application that enables external access for interacting with the plant (Fig. 4.21). The model-based approach aims at configuring a model of a planned or an existing aPS and allowing a continuous extension and modification of the model. In case of additional functionalities or changes of the system, the model-based approach allows the engineer to efficiently perform modifications within the model.

The underlying meta-model is based on the ISA88 standard [Com+95]. Based on the ISA88 standard, an editor for modelling the plants was implemented. Using this editor the PLC control code is generated together with the industry 4.0 interface, allowing remote access and control through executing available services of the plant. It is designed to enable flexible entry of orders during the operation of the aPS through dynamic reconfiguration of orders and process planning. The aPS should independently check whether the services can be performed from a technical perspective. For the verification of the technical limits, pre-conditions and post-conditions are stored within the offered services.

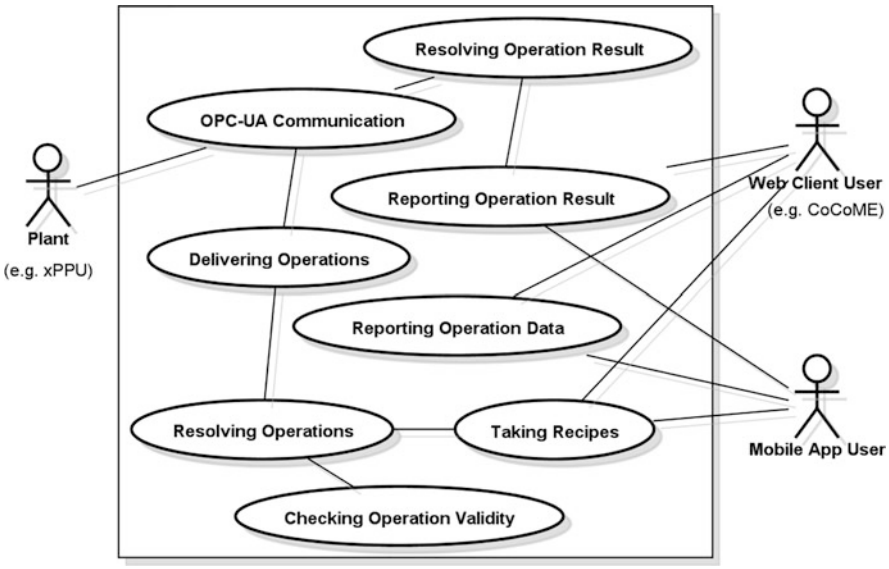

Fig. 4.22 Use cases of the Middleware

One of the main features of Industry 4.0 is connectability of the plant. Thus, besides the use case of the plant (cp. Fig. 4.17), it also includes the Middleware interconnect between the plant (i.e. xPPU) and the users (Fig. 4.22). Over this interconnection, it takes the execution orders from the user side and delivers them to the plant side. Users can be an external operator to execute the system from the remote site or another system to be connected with the plant. From the plant, information regarding the status of the plant or the execution of the operations is delivered to the user side. The communication between the PLC of the xPPU and the Middleware is established over an OPC-UA architecture, which is the official collaboration partner of IEC 61131-3. Middleware executes an OPC-UA client to connect to the OPC-UA server in the PLC and communicates with the PLC over this connection.

For the user side, RESTful (Representational State Transfer) web service is implemented to provide simpler and more lightweight access (Fig. 4.23). Thus, users can have access using an HTTP request (GET or POST) over a web-client application and execute their desired functionality, such as getting a history list, getting variable values, executing a single operation, or executing batch operations. This interface is also implemented in a mobile application form and provided to the users. The Middleware is available for the community on github.[4]

[4] https://github.com/x-PPU/I4.0_Interface.

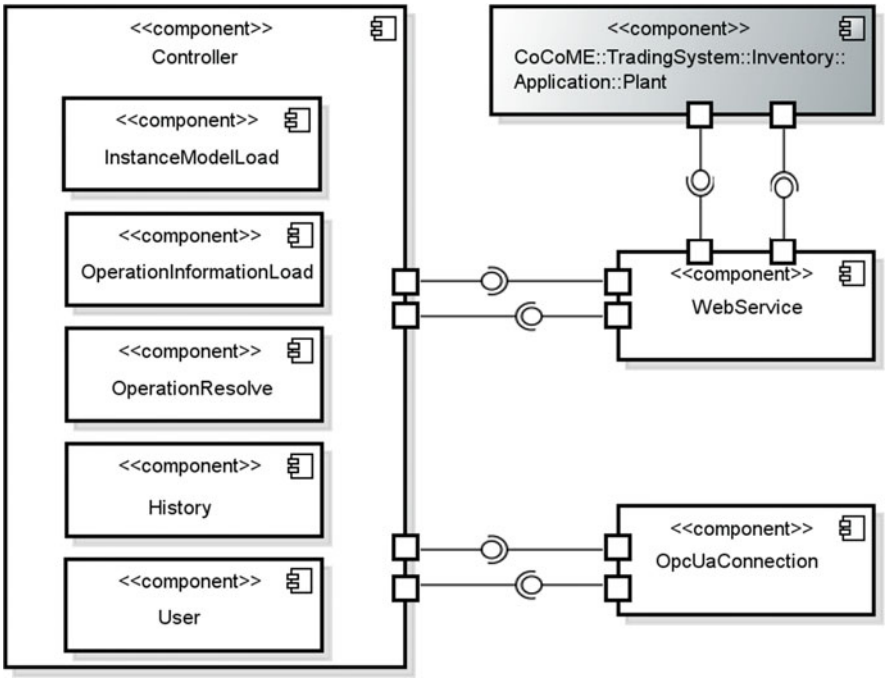

Fig. 4.23 Overview of the Middleware architecture (highlighted are the connection to CoCoME)

4.4.2 Integration of CoCoME and xPPU to Form an Industry 4.0 Case Study

The integration of both case studies, CoCoME and xPPU, can be considered as a further evolution scenario for CoCoME [Bic+18]. The integration of CoCoME and xPPU is based on the hybrid cloud-based variant of CoCoME. Figure 4.21 in the previous subsection shows that the xPPU plant provides a REST interface. The REST interface allows retrieving data and executing the production operations. CoCoME is extended to communicate with this interface (Fig. 4.24).

The main goal of integrating both case studies is to support the *OrderCustomProduct* use case (Fig. 4.25). In this use case, the customer can order individualised products in a store, which are then forwarded to the CoCoME enterprise. In the *DefineProductRecipe* use case, the enterprise manager creates a product recipe based on the order as an ordered list of the needed plant operations, and the CoCoME enterprise triggers the production. In the *SpecifyProductionPlant* use case, the plant operation templates are defined by the plant manager and then forwarded to the connected plants. The production units of the plants use this list to execute the appropriate operations [Bic+18].

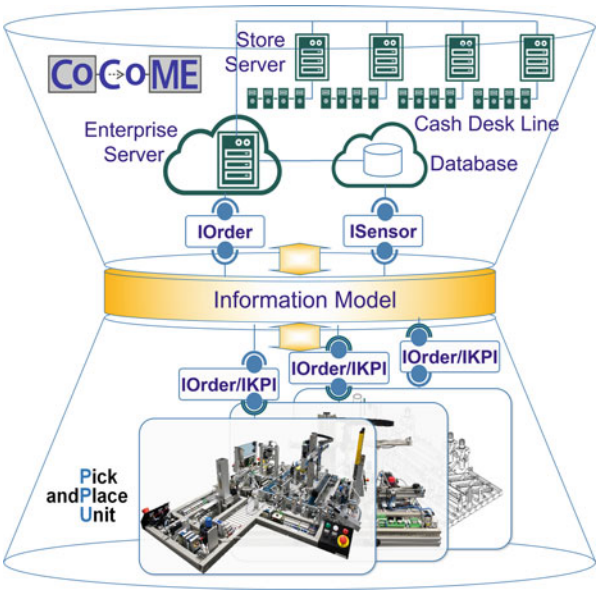

Fig. 4.24 Overview of the integration of CoCoME and xPPU enlarged from [Vog+09, VPF17]

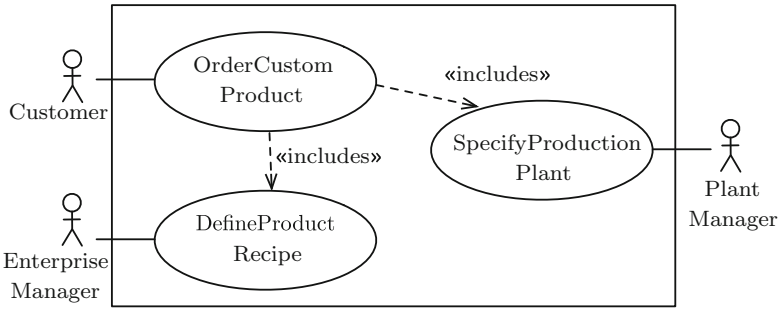

Fig. 4.25 Overview of the use cases for CoCoME after the integration of CoCoME and xPPU, adapted from [Bic+18]

Figure 4.26 illustrates our extensions to a hybrid cloud-based variant of CoCoME to enable the integration of CoCoME and xPPU. In the following, we describe the new components and the relevant changes to the existing components. The `TradingSystem::CashDeskLine::Configurator` component and the corresponding `WebService::CashDesk::Configurator-Service` component enable the customer to configure the custom products. The `TradingSystem::Inventory::Application::Production` component schedules the production order. The `TradingSystem::Inventory::Application::Plant` component provides the functionalities of plant servers,

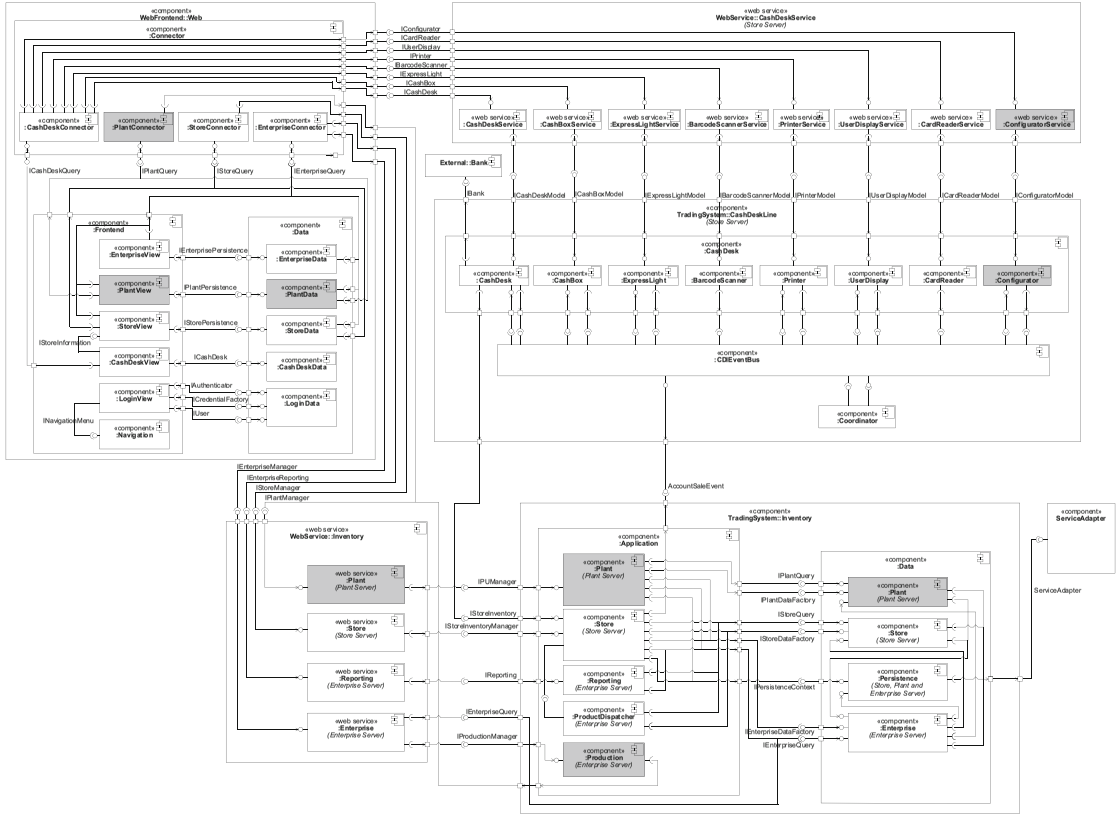

Fig. 4.26 Architecture overview of the Industry 4.0 variant of CoCoME [Bic+18]

such as creating production unit types. The `TradingSystem::Inventory::Data::Plant` involves various data structures, for example for production units. Additionally, we had to extend the `TradingSystem::Inventory::Data::Enterprise` to include further data structures, such as ordering plant or production operations. The `TradingSystem::Inventory::Data::Store` component was extended to manage customised products. The `WebService::Inventory::Plant` component represents the web service of the `TradingSystem::Inventory::Application::Plant` component. Further, we extended the `WebService::Inventory::Store` and `WebService::Inventory::Enterprise` components by the event-based messaging and the corresponding operations for the added data structures. The `WebFrontend::Web::PlantView` component provides the plant manager web-based views managing the production unit and plant operations. The `WebFrontend::Web::EnterpriseView` enables enterprise manager to manage the production order, plants, and custom products. Further, the `WebFrontend::Web:: EnterpriseView::Store` allows configuring and managing custom products. A detailed descriptions of the components of the industry 4.0 variant of CoCoME are given in the technical report [Bic+18].