Chapter 12
Case Studies for the Community

Safa Bougouffa, Kiana Busch, Robert Heinrich, André van Hoorn, Marco Konersmann, Stephan Seifermann, Emre Taşpolatoğlu, Felix Ocker, Cyntia Vargas, Mina Fahimipirehgalin, Ralf Reussner, and Birgit Vogel-Heuser

This chapter describes the benefits and deliverables of the case studies in SPP1593 for the outside community. Section 12.1 sums up the benefits of the Common Component Modeling Example (CoCoME) case study together with the deliverables for the community. Section 12.2 describes the benefits of the Pick-and-Place Unit (PPU) and its extension (xPPU) as well as the deliverables for the outside community. Section 12.3 describes the benefits and deliverables of the industry 4.0 case study that integrates CoCoME and xPPU.

S. Bougouffa (✉) · F. Ocker · C. Vargas · M. Fahimipirehgalin · B. Vogel-Heuser
Technische Universität München, Lehrstuhl für Automatisierung und Informationssysteme, Garching, Germany
e-mail: safa.bougouffa@tum.de; felix.ocker@tum.de; mina.fahimi@tum.de; vogel-heuser@tum.de

K. Busch · R. Heinrich (✉) · S. Seifermann · 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; stephan.seifermann@kit.edu; reussner@kit.edu

A. van Hoorn
Institute of Software Technology, University of Stuttgart, Stuttgart, Germany
e-mail: van.hoorn@informatik.uni-stuttgart.de

M. Konersmann
paluno – The Ruhr Institute for Software Technology, Spezifikation von Softwaresystemen, Universität Duisburg-Essen, Essen, Germany
e-mail: marco.konersmann@paluno.uni-due.de

E. Taşpolatoğlu
Software Engineering, FZI Forschungszentrum Informatik, Karlsruhe, Germany
e-mail: taspolat@fzi.de

R. Reussner et al. (eds.), *Managed Software Evolution*,
https://doi.org/10.1007/978-3-030-13499-0_12

12.1 Benefits and Deliverables of CoCoME for the Outside Community

CoCoME has been designed as a demonstrator for software systems to satisfy research requirements on architecture modelling and evolution of the SPP1593 community. CoCoME is open to the research community. As a community case study, CoCoME aims at providing several benefits to researchers of SPP1593 and the outside community:

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

CoCoME is limited in size and complexity; however, it shows all characteristics of an information system used in industrial practice. Therefore, CoCoME provides a trade-off between modelling complexity and evaluation effort. CoCoME represents a comprehensive knowledge base for the evaluation process that can be exploited and extended by researchers with different backgrounds and research interests. It provides assistance on diverse characteristics important for software evolution, like artefacts in different revisions, comprehensive evolution scenarios, and coverage of different life-cycle phases. The distinct evolution scenarios specified for CoCoME in the course of SPP1593 cover a wide range of adaptive and perfective changes to the system and result in various deliverables to the outside research community.

The several evolution scenarios of CoCoME address a variety of changes to the software architecture and infrastructure. For each scenario detailed description, requirements specification, and design documentation in the form of technical reports [HRR16, HKR18] are publicly available. The implemented source code in Java is available on github[1] for each evolution scenario of CoCoME in SPP1593. Furthermore, models to represent the structure (i.e. architecture) and behaviour of the different variants of CoCoME in the form of PCM are deliverables of the priority programme. These models can be applied for analysing and simulating CoCoME with respect to different quality properties like performance, maintainability, and security. In the following, we give a detailed description of models delivered to the community by SPP1593 to represent the architecture, deployment, and behaviour of CoCoME. Furthermore, we describe how the community applied CoCoME and the models delivered by SPP1593 beyond the scope of the priority programme.

[1] https://github.com/cocome-community-case-study.

12.1.1 Structural, Deployment, and Behavioural Models of CoCoME

The architecture of a software system represents the design decisions during its development and evolution [TMD+09]. Therefore, the architecture of a software system can be considered as one of the main influence factors on its quality properties such as performance, maintainability, or security [TMD+09]. The implementation of design decisions without knowing their effects can be a costly and risky task. Thus, modelling the architecture of a software system and simulating the model of the software architecture enable software architects to understand the effects of different design decisions on software quality properties before implementation. The software architect has to consider the following aspects while modelling the architecture of a software system: its structure, its deployment, and its control and data flow. The model of the software architecture can serve as the input of a simulator [Reu+16, Ros+15].

Palladio is an approach for modelling and simulating the architecture of the component-based software systems [Reu+16]. Palladio can predict the quality properties of the software architecture such as performance, maintainability, or security at design time. Palladio is based on Palladio Component Model (PCM). PCM is the architectural modelling language for component-based software systems. It was initially developed to model and predict the performance properties of a software system. In order to model the architecture of a software system, PCM provides the following view types: (1) repository, (2) system, (3) resource environment, (4) allocation, and (5) usage model [Reu+16].

To support model-driven approaches to predicting quality properties, we provide PCM of the CoCoME architecture. These models were created for the hybrid cloud-based variant of CoCoME describing its structure, deployment, and behaviour.

Modelling the Structure of CoCoME

CoCoME represents a component-based software system. A component-based software system can be modelled by its interfaces and components and their composition (also referred to as composite components) [Reu+16]. The hybrid cloud-based variant of CoCoME consists of the following composite components:

- `org.cocome.cloud.web`
- `org.cocome.tradingsystem.inventory`
- `org.cocome.cloud.webservice.inventory`
- `org.cocome.tradingsystem.cashdeskline`
- `org.cocome.cloud.logic.webservice.cashdeskline.cashdeskservice`

These components are composed of further composite components or individual components. The current architecture model of the hybrid cloud-based variant

of CoCoME contains more than 40 components. The architecture of CoCoME was modelled using the system-independent structural elements of the PCM, namely, components, interfaces, events, and data types. Additionally, the PCM allows assembling the components and composite components to further composite components. Thus, the current model of CoCoME can easily be extended to future implementation (e.g. adding or removing components, interfaces, or data types). Further, the PCM allows modelling the control flows in components at a high abstraction level. Thus, the effect of different control flows (i.e. implementations on a high abstraction level) on the quality properties of CoCoME can be analysed. The repository view type of the PCM allows modelling the previously described model elements.

After the software architect modelled the individual model elements, such as components and interfaces, the architecture of CoCoME can be modelled by assembling these model elements. Further, modelling individual elements allows assembling other variants of CoCoME by different composition of existing model elements, exchanging the model elements by other model elements, or adding new model elements. Thus, the resulting architecture variant of CoCoME and its effect on the quality properties can be determined at design time. The system view type of the PCM allows assembling the software system using the individual model elements defined in the repository view type [Reu+16]. Figure 12.1 shows different components with their provided and required interfaces. The required and provided interfaces are connected to each other using connectors.

The system model of CoCoME is composed of the previously described composite components at the highest level of abstraction. CoCoME provides various services as method invocation of its interfaces. The interfaces of the CoCoME system are defined as follows:

- `ICashDeskView`: This interface mainly provides support for the selling products and managing the express checkout.
- `IShowReportView`: This interface provides services for creating stock reports.
- `IReceiveOrderView`: This interface allows handling ordered products, which have been arrived, such as viewing received orders.
- `IStockOrderView`: This interface can be used to manage purchase orders in the stock.
- `IShowStockView`: This interface provides services for managing stock orders, such as the creation of a new stock order.

Modelling the Deployment of CoCoME

Using the PCM, the resource environment, such as resource containers or linking resources, can be modelled. Further, the resource containers can be annotated with values of different quality metrics, such as mean time to failure (MTTF) for the hard disk or resource demands for central processing units (CPUs) [Reu+16, BKR09].

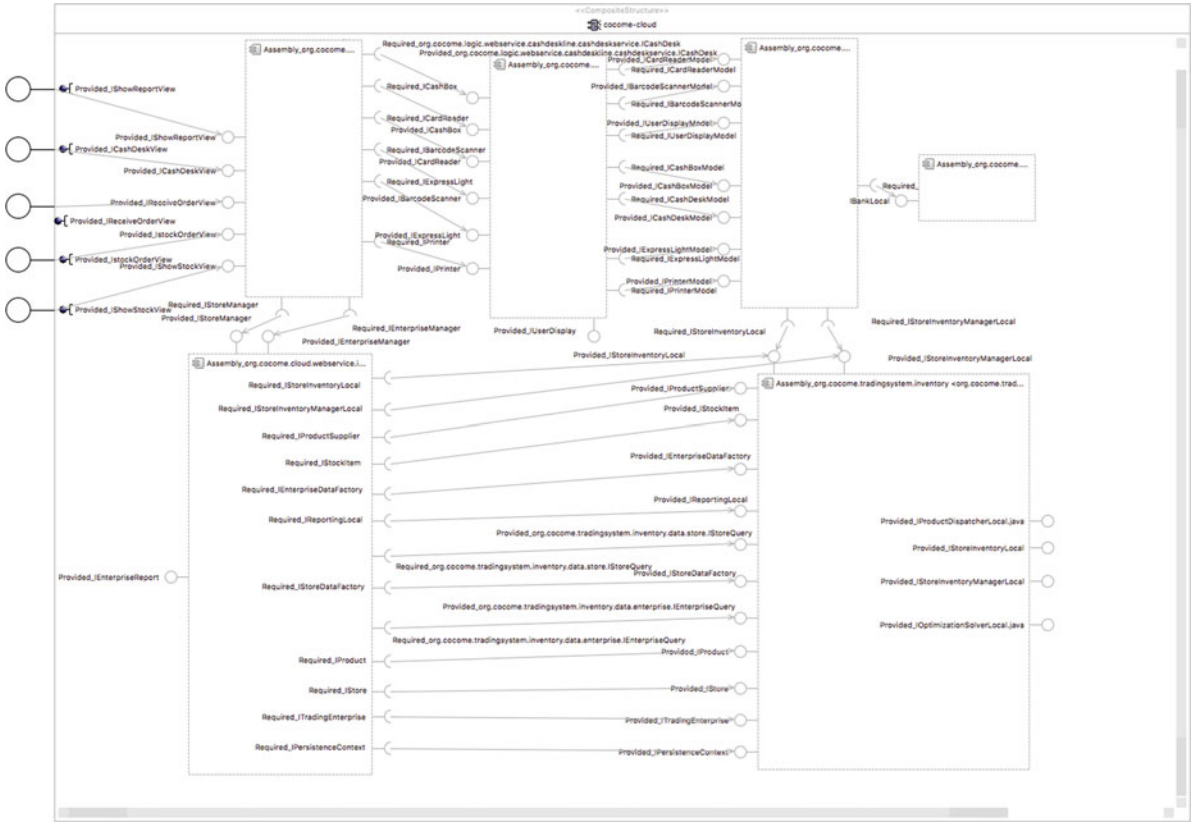

Fig. 12.1 The system model of CoCoME

The resource environment view type of the PCM allows modelling the resource environment. Examples of resource containers for CoCoME are enterprise server, store server, and web node.

After modelling the resource containers, the software architect has to determine which components of CoCoME should be deployed on which resource containers. The system-specific deployment is specified by the allocation view type. For example, the composite component `org.cocome.cloud.webservice.inventory` can be deployed on the resource container enterprise server. It is conceivable that different allocations between components of CoCoME and resource containers are possible. Thus, simulation allows analysing the effects of different allocations on the system's quality properties.

Modelling the Behaviour of CoCoME

The usage model of CoCoME can be specified by the PCM behaviour view type [Reu+16]. Modelling the behaviour allows specifying the interaction of users with CoCoME. For example, different user interactions with CoCoME can affect the performance of the software system. To enable the business process designers to model the usage models, we provided a business process meta-model within SPP1593. This meta-model extends the PCM usage model by actor steps and the resource usage. The business process models allow analysing the effects of changes in CoCoME on the interaction of its users [Ros+17]. This is especially important when we analyse the maintainability of CoCoME regarding different usage scenarios. The business processes can also to be used to analyse the performance of the software system and its business processes [Hei+17a].

To model the business process, the PCM usage model is extended by the specific elements of business process (hereafter referred to as business process usage model). Business process can be considered as a set of connected activities. At the lowest abstraction level, activities can be actor steps, system steps, or steps regarding the resource device usage. The main difference between the actor steps and the system steps is that the actor steps are completely performed by human actors, whereas system steps are executed automatically by the software system. Further, human actors can use the resource devices to perform their activities. Therefore, they can acquire the resource devices before using or release the device resources after using [Hei+17a].

The processes of CoCoME can be modelled using the business process usage model. The previously described services of CoCoME are part of processes (e.g. sale process), which CoCoME provides (cf. [Her+08a]). In the following we describe different processes of CoCoME:

- *ProcessSale* deals with selling products. It can be considered as the main process of CoCoME (cf. process 1 in [Her+08a]).
- *ManageExpressCheckoutProcess* describes the fast sale process for purchasing only few products (cf. process 2 in [Her+08a]).

- *OrderProducts* describes how new products can be ordered (cf. process 3 in [Her+08a]).
- *RecieveOrderedProducts* defines how to manage the arrived ordered products (cf. process 4 in [Her+08a]).
- *ShowStockReports* describes the process of the creating the stock reports (cf. process 5 in [Her+08a]).
- *ShowDeliveryReports* deals with creating the delivery reports (cf. process 6 in [Her+08a]).
- *ChangePrice* defines how the prices of products can be changed (cf. process 7 in [Her+08a]).

In the following section, we describe how the models of CoCoME can be applied to analyse different quality properties.

12.1.2 Analysing Maintainability for CoCoME

The maintainability of a system can be considered as the ease of implementing changes in that system [ISO10]. In other words, the maintainability of a system in the case of a change request correlates with the set of system elements that have to be changed [HBK18a]. As a community case study, several development artefacts of CoCoME are available. Examples of such artefacts are code, requirement descriptions, and aforementioned models. Thus, CoCoME is well suited for comparing different maintainability estimation approaches. The application of an approach to CoCoME allows comparing it with other approaches that have been applied to CoCoME. For example, if we have different maintainability estimation approaches, which estimate a set of changed elements for a change request, we can compare these sets with each other for given change requests. Thus, the application of maintainability estimation approaches to CoCoME allows improving them with regard to the change propagation analysis. Additionally, having a common community case with its development artefacts allows analysing how the change requests affect different artefacts [Ros+17].

The CoCoME models described in the previous section can serve as the input of model-driven approaches to maintainability analysis in software systems. In this context, the models describing the structure of CoCoME are especially important, as the structure of a software system affects the change propagation in that system [HBK18a]. Modelling CoCoME by components and interfaces in a fine-grained manner improves the change impact analysis in the software architecture.

Software systems can be used to support business processes of organisations. Therefore, there are mutual dependencies between the software systems and the corresponding business processes [Ros+17]. Thus, we have to consider both software systems and the corresponding business processes while analysing the change impact. In addition to the structural models of CoCoME, the maintainability estimation approaches regarding software systems and business processes can also

use its usage models describing the behaviour of CoCoME. As described in the previous section, the processes of CoCoME are modelled as a set of connected actor steps and system steps. This allows analysing the effects of changes based on the mutual dependencies between the software system and the corresponding business processes [Ros+17].

In the model of software systems, the structural model elements such as interfaces and components can be used to analyse the propagation of change [Ros+15]. In the model of business processes, the activities such as system steps and actor steps can be used to calculate the change propagation [Ros+17]. Further, the data flow plays an important role in change propagation [HBK18a]. The change can propagate based on the data flow between a software system and its corresponding business processes [HBK18a, Ros+17]. To model the data flow in CoCoME, we modelled different data types in its software model and different data objects in its business process model. As a data object can correspond to a data type and vice versa, the models of CoCoME allow analysing the change propagation based on the data flow [Ros+17].

To model the change request and to analyse the change propagation in terms of affected model elements, we also provided a further meta-model—the modification marks meta-model [Sta15, HBK18a]. Modification marks meta-model allows software architects and business process designers of CoCoME to mark the initially affected model elements (hereafter referred to as seed modification). Based on the seed modifications, the task list can be generated automatically. The generated task list contains a set of maintainability tasks, where each task refers to a model element that is potentially affected by the change. The maintainability tasks are grouped in different change propagation steps based on the cause of the change propagation [HBK18a]. Figure 12.2 illustrates a generated task list for the change request modifying the interface IBarcodeScanner. Thus, the interface IBarcodeScanner is the seed modification. Each task in the task list corresponds to a model element of CoCoME. Further, Fig. 12.2 shows several change propagation steps. For example,

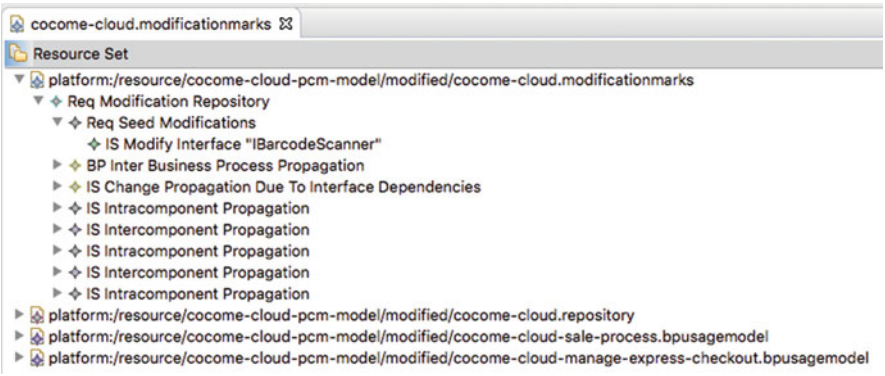

Fig. 12.2 Analysing the change propagation in CoCoME using the modification marks meta-model

business process (BP) inter-business process propagation indicates that the change propagates only in the business process. This change propagation step contains only business process model elements. As described previously the data flow may cause the change propagation [HBK18a]. Information system (IS) change propagation due to data dependencies in Fig. 12.2 refers to structural model elements of CoCoME software system that is affected by the change due to the data flow. The change can also propagate between the provided roles of a component and the required role of other components (i.e. IS intracomponent propagation in Fig. 12.2) and between the required role of a component and its provided role (i.e. IS intercomponent propagation in Fig. 12.2).

12.1.3 Modelling Security Patterns and Attacks for CoCoME

Security plays a crucial role in systems with important assets like critical tasks or such that includes personal information. Long-living systems going through software evolution face security problems similar to any other quality requirements which are open to degradation. It is very important to preserve the secure state of a system, as itself or its environment, usage, configuration, etc. change, which can affect directly or indirectly the correct functioning of security mechanisms. In such cases, made design decisions for mitigating threats and addressing possible security vulnerabilities can lose their validity. Furthermore, it would be worse if there are no signs for such invalidations, so that the problem based on any change can be first discovered after an attack occurs or a vulnerability is exploited. Hence, the software engineers confront the degrading security, and addressing it becomes a more challenging job, if any corresponding documentation of the security decisions made and their assumptions do not persist over time.

As previously described, Palladio architecture models are historically developed for performance simulations and analyses. However, being a model-based documentation of complex software systems and providing several abstract views (i.e. repository, system, allocation, etc.), it also qualifies for investigating security. To this end, we first use the security definition as a combination of confidentiality, integrity, and availability [CH13], which can be interpreted on architecture models.

The approach **PreReqSec** [TH16a] supports software engineers and architects to consider security as early as possible in the design time, which considers runtime, configuration, or usage information. Further support is provided during the software evolution, as the possible changes can be reflected upon the architecture, where less complex but still powerful security analyses are possible. To this end, security-related information is modelled as first-class entities on architectural level in so-called security catalogues, which at the time are being developed for the case study CoCoME. The hybrid cloud-based variant of CoCoME provides several evolution scenarios, in which different aspects and entities of the system change (see Sect. 4.2). The corresponding architecture models are well suited for demonstrating

the application of PreReqSec. In the following, we discuss two evolution scenarios from a security perspective.

- `Platform Migration:` This evolution scenario migrates several local resources (i.e. enterprise server and database) of the system to cloud for reducing operating costs and providing flexibility in adaptation and reconfiguration. However, it also introduces new challenges such as the privacy concerns or third-party trust that might affect the confidentiality and availability. Furthermore, providing such flexibility can make design time decisions obsolete. For example, before cloud integration, the system had to handle all private data in local servers, where no remote calls to data stored in cloud were necessary, which was no subject for, e.g. man-in-the-middle threats.
- `Adding a Pick-Up Shop:` As the customer landscape and competition between other providers grow, this evolution scenario provides new business models such as online shopping and new use cases like online payment. A completely new system interface is introduced. Hence, the attack surface expands and new attack vectors become possible. These changes affect both confidentiality and integrity as well as availability. As the very fundamental requirements and therefore the system itself change, it becomes necessary to validate the already-made security decisions as well as consider new ones.

Security catalogues within the PreReqSec approach can be used globally due to their reusable nature, or they can also be project- or application-specific. A small snippet from the security catalogue for CoCoME can be seen in Fig. 12.3. The catalogue is open for further development and also can be used in other web- and cloud-based software applications.

Mainly, the catalogues consist of three parts corresponding to two different pillars of security. These are the attacks and security patterns, which are combined by so-called prerequisites. Prerequisites in the PreReqSec approach are structural and logical information corresponding to assumptions or requirements that define in which cases a threat can succeed or a security measure can mitigate a given attack.

Threat models provide information about vulnerabilities and possible attacks, which are analogous to the usage profiles of the Palladio Component Model. They are in the PreReqSec approach, a simplified version of usage models, which are modelled as basic interface calls for maliciously getting into the software system. Based on black-box modelling, we do not provide any further information once the interfaces are passed, which is to be interpreted as the risk of any malicious penetration through system boundaries. However, if necessary an attack vector can be modelled with the help of profiling/stereotyping for representing more complex attacks like advanced persistent threats (APT attacks). Based on the evolution scenarios, a security expert provides the information regarding possible threats.

The security catalogue includes the following attacks: `HTTP-Flooding`, `SYN-Flooding`, `Persistent-Cross-Site-Scripting`, or `Cross-Site-Request-Forgery`.

On the other hand, security patterns (analogous to the well-known design patterns) provide the necessary information for a structural solution to recurring

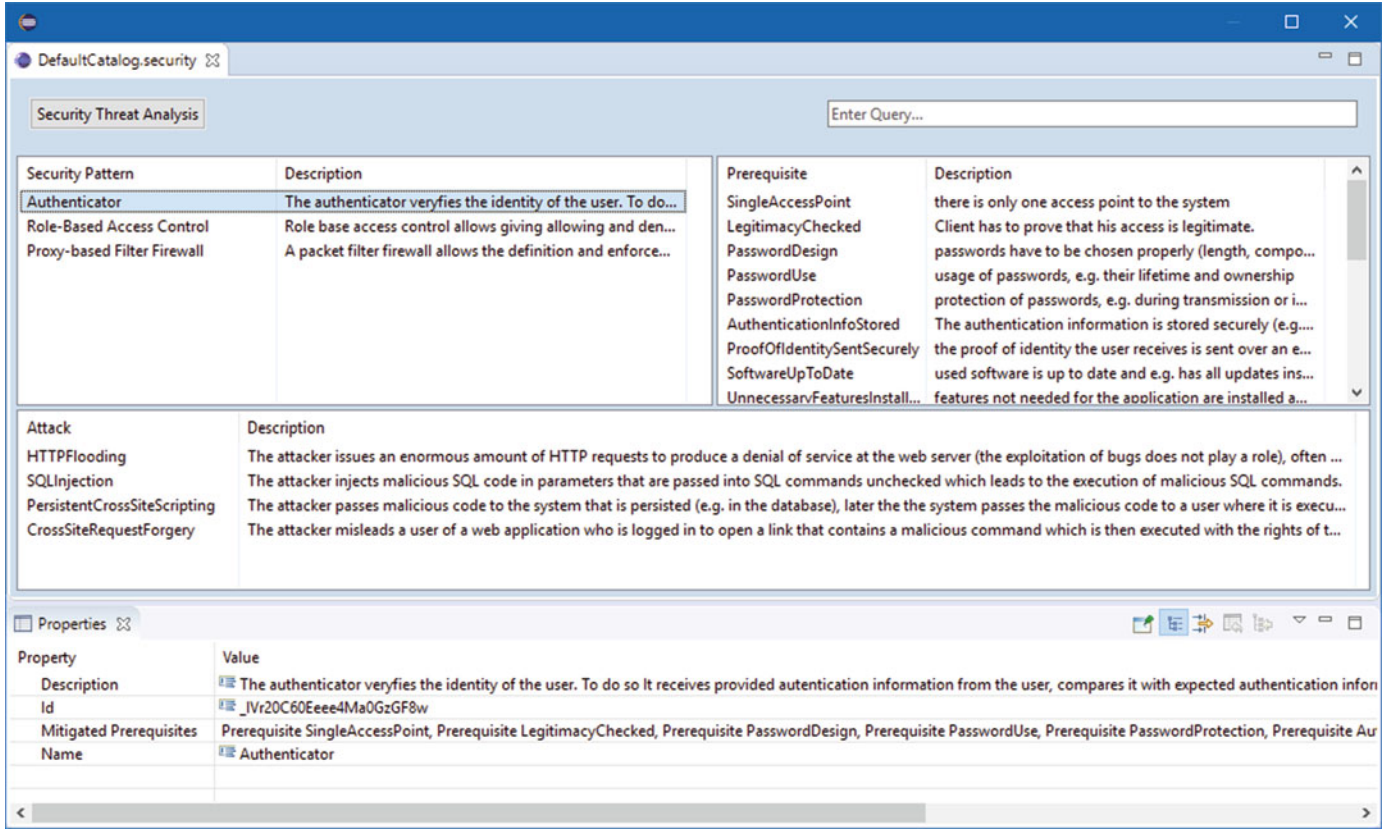

Fig. 12.3 A snippet from the security catalogue—shown in the column-based editor integrated with Eclipse properties view

security problems from the threat catalogue. Every security pattern consists of structural roles. A role corresponds always to one or more software (or in some cases hardware) components. It basically states that (in case) the related component fulfils a specific task within the entire security pattern. To provide reusability and to be able to use the security catalogues with any given architecture description language, security patterns are separated from the system models and used profiling/stereotyping to connect the elements of security pattern (i.e. roles) with the corresponding architectural elements (e.g. composite components, system interfaces). Due to these separation of security patterns as well as attacks, the PreReqSec approach provides a new security view.

After defining possible threats, security expert and software architect can first check which of the already-made security decisions may become obsolete, since they cannot mitigate the new threats. This is mainly a task within the security analysis, which is at the time an ongoing development within the PreReqSec approach. However, extending the documentation (i.e. security catalogues) is now possible with the security expert and software architect. Based on their expertise, state of the art, and security best practices, they make new decisions about possible security patterns, which are able to mitigate the newly introduced or changed attacks. Following security patterns are for the time being defined and modelled in the catalogues for CoCoME, for which several extensions are planned directly in CoCoME system models, such as:

- `Role-based Access Control` is a security pattern to provide secure access control for different users to different assets. The pattern is structured within several different elements from CoCoME like `org.cocome.cloud.webservice.LoginManager`, `org.cocome.tradingsystem.inventory.data.UserManager` or `org.cocome.tradingsystem.inventory.application.UserManager`.
- `Proxy-based Firewall` is a well-known firewall pattern for web applications.
  `org.cocome.cloud.proxybasedfirewall` is the main component for firewall security pattern. It is deployed on web node and traffics the communication of
  `org.cocome.cloud.web` composite component.

After instantiating the attacks and security patterns, it comes to how to combine them structurally. The combining elements between the attacks and security patterns are the logical prerequisites, as previously described, which are used as parameters for the analysis to validate the made security design decisions with respect to possible changes or to design time unknown information. However, besides the development of automated architecture-based security analysis, extending the security patterns and threats with modelled prerequisites is an ongoing part within this research. Due to highly considerable manual effort which involves many views and roles, it makes a very steady foundation for considering security within software

architectures and allows analysing this very important quality attribute without having to code, deploy, or penetration test the software system under consideration.

12.1.4 Modelling and Analysing Data Flow for CoCoME

Software architects can analyse many quality properties of software architectures by means of activities executed in the way a control flow defines. However, there are quality properties such as compliance with data privacy constraints that are naturally defined in terms of data and data flows. Ongoing research in the field of architectural data flow analyses [Sei16] tries to leverage definitions data and its processing as well as concise privacy constraint descriptions to determine compliance of architectures with privacy constraints. CoCoME serves as a case study for evaluating architectural data flow analyses.

The pick-up shop evolution scenario implies many privacy constraints because it introduces user-related data. In a first step, a subset of CoCoME about the creation of reports has been chosen. Depending on the realisation of the use case, the store manager might get access to data that (s)he does not actually need but are worth protecting. This includes personal information about users. An access control policy defines which roles have access to which data, which serves as an input for an analysis. The second step is making exchanged data explicit and specifying the data processing. As a third step, we create several realisations of the use case that imply privacy violations and analyse them for violations.

Even if extending CoCoME by data and data processing requires considerable effort, it is a good foundation for case studies in the field of data privacy. It processes sensitive and non-sensitive data in various ways defined by the use cases. Therefore, it defines a reasonable network of data processing operations and data exchanges. However, the case study is not artificial but realistic, which allows to draw conclusions about applicability.

12.1.5 Diagnosis of Privacy and Performance Problems for the CoCoME Mobile App Client

Users of mobile apps expect fast response times and high throughput. However, there are a lot of different mobile devices with different specifications, which makes it hard to show adequate performance on each of them. Another important quality property of mobile apps is privacy as lot of sensitive data is stored at mobile devices and transferred over a bunch of different networks. Thus, observing and analysing mobile apps for performance and privacy issues are crucial. The monitoring and analysis approach proposed in [MHH18] is capable of identifying both privacy and performance problems of mobile apps.

The approach has been evaluated based on the mobile app client evolution scenario of the CoCoME case study. In the mobile app client, the use cases AuthenicateAppUser and ProcessAppSale have been executed. Monitoring data for the use cases have been recorded and analysed for performance and privacy problems in order to evaluate the accuracy of the analysis and overhead of the mobile monitoring approach [MHH18].

12.1.6 Functional Decomposition for Identifying Microservices in CoCoME

A big challenge in designing microservice architectures is to find an appropriate partition of the system into microservices. Microservices are usually designed intuitively, based on the experience of the designers. A systematic approach to identify microservices in early design phase is described in [Tys+18]. The approach is based on the specification of the system's functional requirements and uses functional decomposition to identify microservices.

CoCoME has been used as a case study for evaluating this approach. Starting with the use case specification of CoCoME, system operations and state variables have been extracted and clustered for identifying microservices. The clusters identified by the proposed approach have been compared to microservices identified by human developers based on the CoCoME source code and design documentation (the component diagrams and sequence diagrams given in [HRR16]). For this purpose, CoCoME has been applied as a case study at the Centre for Research and Innovation in Software Engineering at Southwest University in Chongqing, China, and the Karlsruhe Institute of Technology, Germany. The outcome of the functional decomposition of CoCoME is analogous to the evolution scenario microservice architecture.

12.1.7 Distributed Quality Property Optimisation for CoCoME

Software products must satisfy a considerable number of non-functional quality properties, e.g. regarding performance and modifiability. It is known that quality properties may conflict with each other. The reason is that software changes aimed to improve one quality properties can and usually do have a negative impact on another property. Hence, trade-offs between quality properties need to be managed to achieve an overall accepted level of quality for the software product. Architecture-based optimisation aims for evaluating such trade-offs in early design stages [Ale+13].

CoCoME is being used as a case study for a novel distributed approach for optimising quality properties of software architectures, in an approach called

SQuAT [Rag+17a]. While the approach is property-agnostic, the evaluation focused on the quality properties performance and modifiability. SQuAT builds on the idea of scenario-based architecture evaluation [BCK12], which proposes to evaluate architectures based on their ability to support quality scenarios. In the SQuAT framework, so-called bots evaluate and try to optimise "their" respective quality scenario. In multiple iterations, a moderator aims to support the mediation and negotiation between the bots, by sharing solutions that can be further improved by the bots to find a joint solution. SQuAT employs model-based quality prediction, currently focusing on the Palladio Component Model (PCM). It currently, support two types of bots—one for performance and one for modifiability [Rag+17a].

CoCoME is used for a large-scale evaluation of the SQuAT approach, whose basic effectiveness has been evaluated in a smaller study before [Rag+17a]. The existing PCM are used with slight modifications. In total, eight scenarios are defined—four for modifiability and four for performance. Accordingly, eight bots execute to find an architectural CoCoME candidate by incrementally improving architectural candidates, using modifiability and performance tactics.

While the case study is still in progress, it has already greatly helped to reveal and resolve challenging situations related to the model size and topology in the SQuAT framework.

12.1.8 Extracting Architecture Models of CoCoME

Software architecture models in different modelling languages were automatically generated from the source code of the plain-Java variant of CoCoME in [Kon18]. These models and their generation serve as a showcase of architecture model extraction using *Codeling*.[2] In this context, two types of architecture models have been extracted—PCM and UML composite structure diagrams. The generated PCM and UML models are available for the community.[3] Codeling is a tool that can extract architecture models from the source code and maintain traces between the model and the code. That is, when the model is changed, the source code is changed accordingly. The source code that is not represented by the model is not lost during this operation. For example, when a component is renamed, the corresponding source code will still contain its operations and their implementation.

Code that follows the specification of a component framework—such as the Java Enterprise Edition (JEE) [Ora17]—is forced to be structured in specific ways for representing architectural elements such as components and their interconnection. For example, any type of "bean" in JEE can be considered a component. Beans in the JEE are Java class declarations with specific Java annotations. That means that a class declaration in the program code, which has that specific Java annotation, can

[2] https://www.codeling.de.

[3] https://github.com/cocome-community-case-study/models/tree/master/Codeling/.

be identified as a component. Codeling uses such predescribed code structures to generate a translation model that complies to an intermediate architecture language developed for Codeling. This translation model can be translated into multiple architecture description languages. Two architecture models of the CoCoME program code were extracted. The plain-Java variant of the CoCoME program code was used for this purpose. That variant actually does not follow any standardised component framework but established a custom notion of components and other architecture-related concepts. This project-specific component framework needed to be analysed first, to find out which structures were used to implement architectural elements. In the following sections, we present the models and briefly describe how they were extracted from the program code with Codeling.

PCM Extracted from the CoCoME Source Code

In the first case study, PCM were generated, which allow for performance simulations of the architecture. The PCM defines multiple view types for modelling an architecture. In this case study, we extracted the repository model, the system model, and the instance models of CoCoME's composite components from the code. Neither the resource environment and allocation nor usage models for performance simulations are encoded in the program code. These have to be added manually for the purpose performance simulations. The originally planned architecture model of the plain-Java CoCoME system is shown in Fig. 4.5 on page 43. Figures 12.4 and 12.5 show the PCM repository and system, which were extracted from the plain-Java CoCoME program code using Codeling. The repository contains all identified components and composite components, alongside with their required and provided interfaces and operations. For each composite component, the model declares an instance of each subcomponent, correctly interconnected as defined in the program code. These diagrams are not shown in the figures at hand. The system diagram declares one instance of each of the top-most components in the repository and interconnects them as declared by the program code. Provided interfaces are propagated to the system's context.

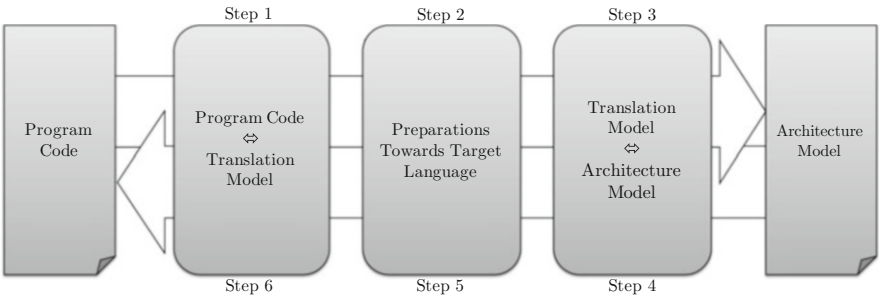

Fig. 12.4 The CoCoME architecture in an PCM repository diagram, as extracted with Codeling

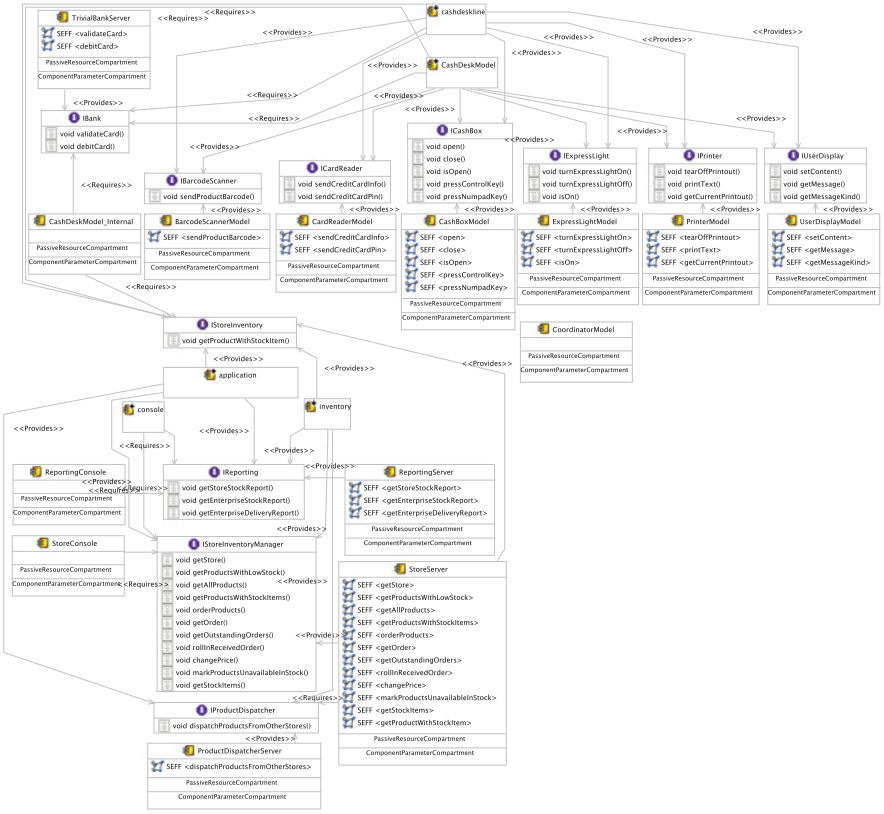

Fig. 12.5 The CoCoME architecture in an PCM system diagram, as extracted with Codeling

The translation between the CoCoME source code and the intermediate language is not complete in the sense that the event mechanism has not been translated. Therefore, the component *JMSEventBus* is missing in this model as well as event-based interfaces. Also, the data access components have not been translated in this case study. The missing pieces could also be extracted, if the corresponding code structures would be added to the Codeling. It should also be noted that Codeling only extracts the model information. The layout has been applied manually in the figures at hand.

For extracting the architecture model, Codeling uses the process shown in Fig. 12.6. Konersmann [Kon18, Chapter 8] describes the process and all steps in detail. Here we give an overview for understanding how the architecture models of CoCoME were extracted. The process can be started from either the architecture model or the program code. In the case of the CoCoME case study, an architecture model was extracted from code. Therefore, here the process was started from the program code.

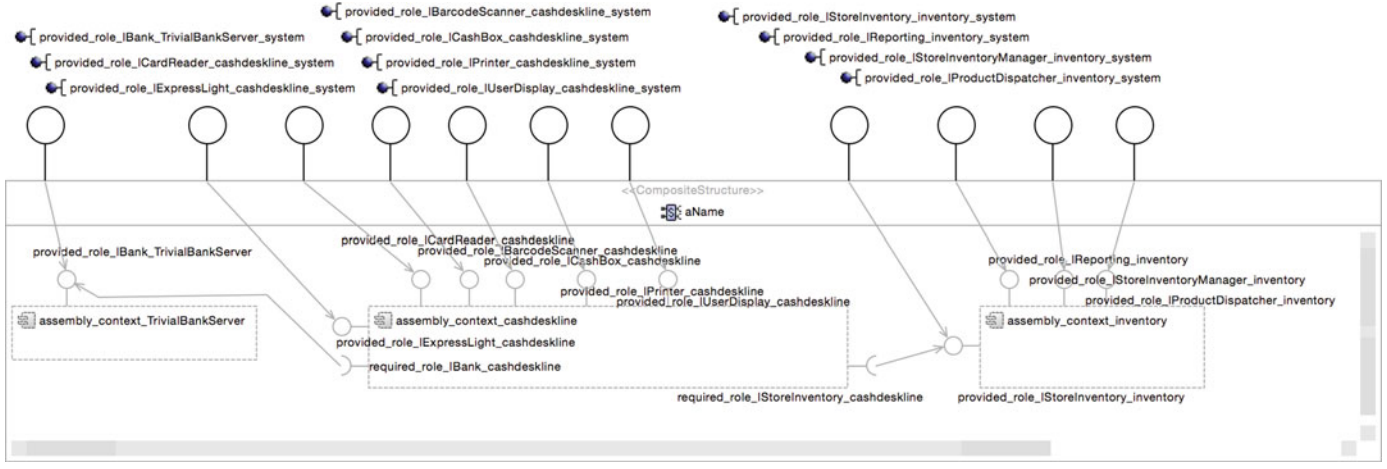

Fig. 12.6 Overview of the process to extract architecture models form program code with Codeling and the propagation of changes in the model to the code

The process defines three main steps for each direction. For extracting an architecture model, the following steps are executed:

Step 1 extraction of a translation model from the program code via an implementation model

Step 2 preparation of the translation model to the necessities of the target language

Step 3 translation of the translation model into the targeted architecture modelling language

In the step *Program Code to Translation Model*, a translation model is created based on the source code. The step comprises substeps: First, an implementation model of the source code is extracted. This model describes the implementation with the terms of the component framework in use. As described above, the CoCoME implementation uses a project-specific component framework. This framework needed to be analysed first, to find out which structures were used to implement architectural elements. Second, this model is then translated into a translation model, which complies to Codeling's intermediate architecture language.

In the step *Preparations Towards Target Language*, the translation model is prepared for the targeted architecture modelling language. This is used to compensate differences between component frameworks and architecture modelling languages. For example, imagine a component framework, which requires deployment information to be valid. In this step, minimal deployment information would be added in the intermediate language to enable the translation. In the step *Translation Model to Architecture Model*, the translation model is translated into the targeted architecture modelling language, in this case study into a PCM representation.

When the architecture model is available, it can be viewed, analysed, and changed with its original tools. For propagating the changes to the code, reverse steps (**steps 4 to 6**) are executed, while following the traces collected during the translation from the code to the model.

UML Models Extracted from the CoCoME Source Code

In the second case study, a UML composite structure diagram was generated from the same translation model, effectively reusing the transformations between the program code and the intermediate language. Parts of the transformation between the intermediate language and the UML were reused from another case study. The UML model extracted with Codeling is shown in Fig. 12.7. The arrows without a keyword are *ComponentRealization* relations in UML.

The extraction of the UML model is based on the same translation model as the extraction of the PCM. Therefore, the translation between the CoCoME source code and the implementation model and the transformation between the implementation model and the intermediate architecture language of Codeling can be reused from the PCM extraction case study presented above. As these translations do not include event-based communications and data management, these are also not included

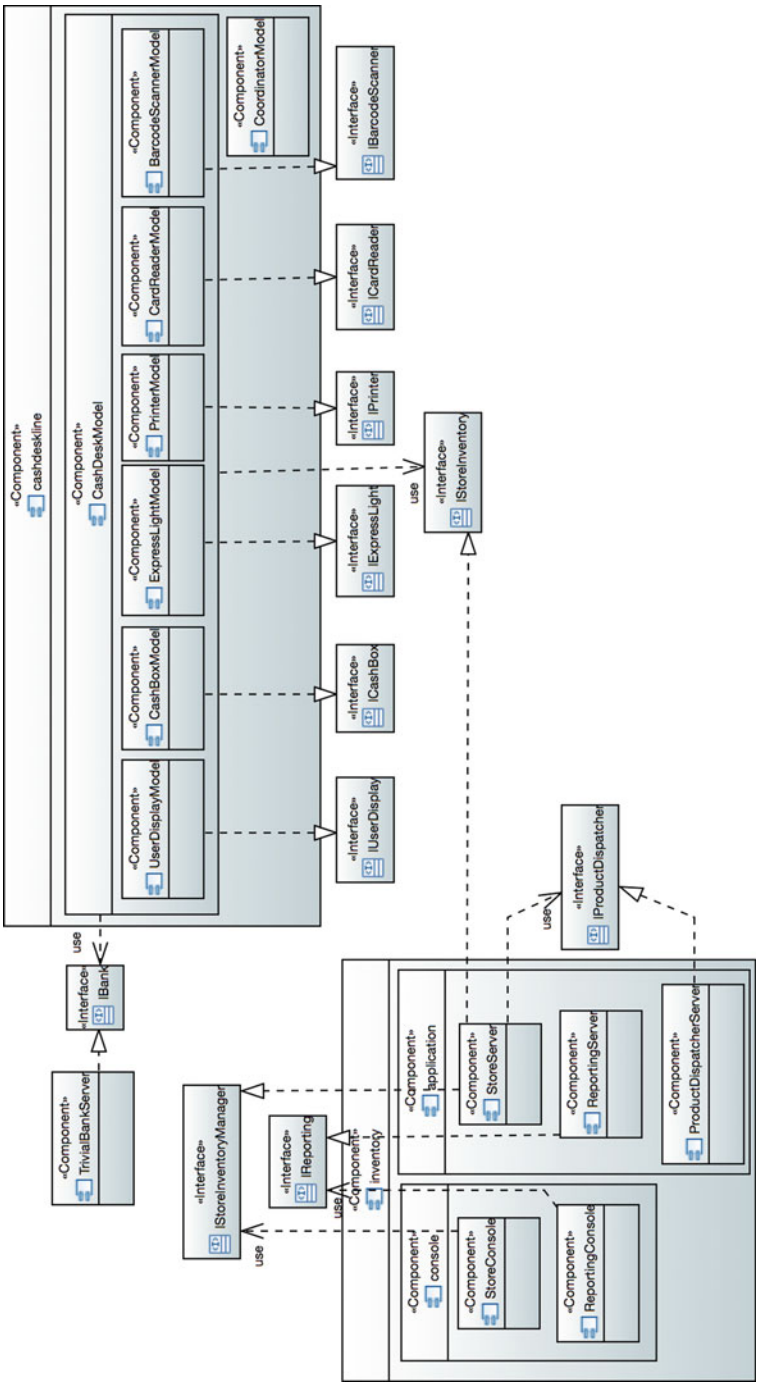

Fig. 12.7 The CoCoME architecture in an UML composite structure diagram, as extracted with Codeling

in the UML diagram. Translations between the intermediate language and UML already exist from another case study, which translates between JEE program code and a UML representation [Kon18, Section 10.1]. The UML is flexible to use, and the use of the UML can vary in different contexts. The existing translations between the intermediate language and the UML are project-specific in their handling of composite components. Therefore, we derived a new, more general translation between the intermediate architecture language and the UML.

The goal of the case studies presented above was to extract PCM and UML models, respectively. The propagation of model changes to the CoCoME source code was not intended, because the source code was considered legacy code, which is not maintained anymore. Therefore, the translation from the model representation to CoCoME source code structures was not implemented in these case studies.

12.1.9 Extracting Behaviour and Usage Models of CoCoME

Similar to the approach by Konersmann, the approach proposed by Langhammer et al. in [Lan+16] and [Lan17] extracts static architecture models of CoCoME from the source code. The automated approach by Langhammer et al. additionally extracts behaviour and usage models in the form of PCM from very limited information like source code and test cases. The PCM can then be used as an input to existing analysis techniques like the aforementioned for performance, reliability, and maintainability.

The plain-Java variant of CoCoME has been applied for validating the accuracy of the usage model extraction in [Lan+16]. Usage models have been extracted based on existing unit tests for the use cases of CoCoME. An example of an extracted usage model for the *ProcessSale* use case is given in Fig. 12.8. Further PCM for specifying the system's architecture and behaviour have been extracted in [Lan17].

12.2 Benefits and Deliverables of xPPU for the Outside Community

The PPU and xPPU were developed to meet the research requirements on the evolution in plant and machine manufacturing of the SPP1593 community. The xPPU is open to the researcher community providing some promising opportunities such as comparability of different approaches and facilitating the tackling of aspects that are less frequently dealt with. Furthermore, an eased exchange of research ideas and a simplified coordination are given.

The xPPU is limited in size and complexity but nevertheless provides a trade-off between problem complexity and evaluation effort. To support different research directions, the xPPU evolution scenarios were extended regarding more sophisticated requirement modelling, as well as fault handling functionality. Evolution in

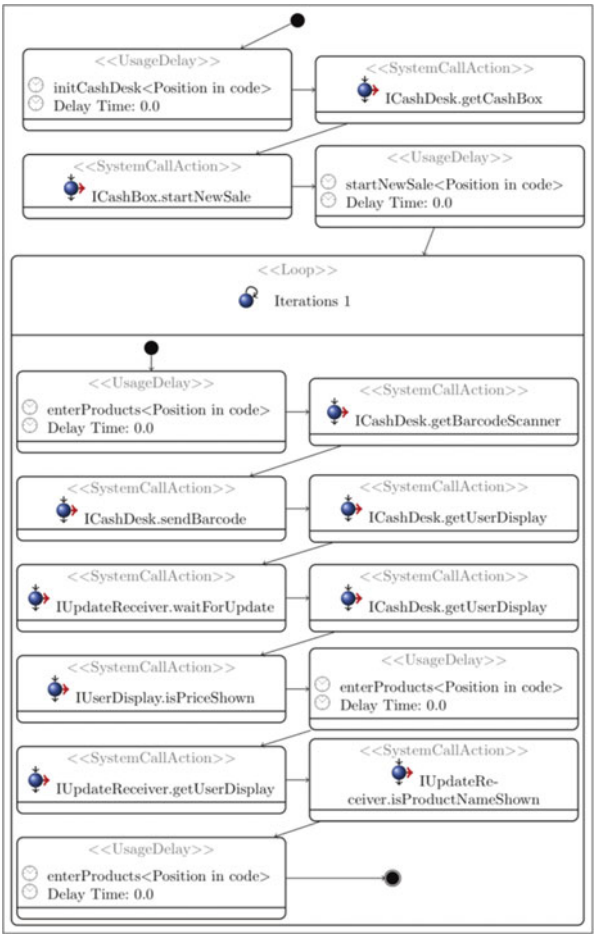

Fig. 12.8 Usage Model of the *ProcessSale* use case extracted from unit tests [Lan+16]

xPPU not solely includes sequential development but also parallel and asynchronous evolution; see Fig. 12.9. Due to the demand for higher throughput of workpieces (WPs), Scenario 12a with a faster sorting of WPs is developed. A drive with increased dynamics is installed to realise faster WP movement, which entails that faster pushers are required for extruding WPs. In parallel, a customer demands an adjusted variant of xPPU's Scenario 12, which is able to handle larger and heavier WPs. Furthermore, depending on the country, a machine or plant should be located in, different supply and control voltage must be supported by field devices. Whereas the existing xPPU is engineered to be located in Germany, a customer requests an xPPU which can be operated with different supply and control voltage (as used, e.g. in the USA). Accordingly, all field bus components, which are not capable to handle the desired control voltage, have to be replaced. This results in another

Triggering requirement	Scenario	Transportation belt			Remarks
		M	AH	S	
Higher throughput of workpieces	12a	×	0	×	Increased dynamics of pneumatics for pushing WPs into slide resulting in different time constraints for monitoring
Increase in workpiece size and weight	12b	×	0	×	Larger slide and increased pneumatic force and dynamics
Different control voltage	12c	×	×	0	Different I/O modules required
Different platform supplier, e.g. Siemens, Rockwell, Beckhoff or Microcontroller	12d	0	×	(×)	Difference in automation components, i.e. drives, fieldbus and I/O modules or HMI may be required
Other than IEC 61131-3 environment requested by customer	12e	0	×	×	Different PLC and software required, e.g. IEC 61499 or C++
Additional functionality selfhealing machine and diagnosis	12f	×	0	×	Additional sensors and software required, automatic mode enlarged
Remote service required	12g	0	×	×	Data logging and analysis, remote access necessary

Fig. 12.9 Parallel evolution influencing mechanics (M), automation hardware (AH), and software (S) [Vog+14a]

variant of the xPPU referred to as Scenario 12c. In addition, a typical variation point driven by automation hardware is the requirement of a specific vendor platform, e.g. Siemens or Rockwell. This aspect is reflected in Scenario 12d, which is a version controlled by a vendor other than CODESYS platform. Besides the platform, another typical variation driven by automation technology is the implementation environment, i.e. whether the PLC software is, e.g. implemented according to IEC 61131-3 or IEC 61499. In Scenario 12e of the xPPU, an implementation framework other than the currently applied IEC 61131-3 environment is used. In parallel to these variants, Scenario 12f is more reliable due to the realisation of self-healing functionality. To extend the business cases of plant and machine suppliers, a variant supporting remote services is provided with Scenario 12g. To realise remote service functionality, data logging and analysis techniques are required as well as the possibility to remotely access operational data.

The different evolution scenarios of the xPPU targeting a variety of sequential changes in system architecture and behaviour are publicly available. These scenarios are documented with structural and behavioural models, PLC implementations in classical IEC 61131-3 as well as implementations based on state charts in plcUML (i.e. an alternative way to implement PLC software), Matlab/Simulink simulation projects, mechanical CAD files, and describing products, processes, and resources of the xPPU is available in Automation Markup Language (AML). Furthermore, a technical report of the scenarios is available to allow the understanding of the different evolutions capturing models as well as detailed description of changes in each scenario [Vog+14b].

12.2.1 Structural and Behavioural Design Models of the xPPU

The language profile SysML was developed to support the physical systems' design [Esp+09]. SysML covers the modelling of functional requirements and corresponding software applications as well as the modelling of non-functional requirements [Fra+11]. SysML models consisting of class, state machine, and block definition diagrams of the xPPU's evolution scenarios are provided.

The SysML models of the xPPU are based on the ANSI/ISA S-88 standard of batch process control [Com+95], published by the International Society of Automation, and the OMAC State Machine, part of the Packaging Modelling Language (Pack ML) standard [Are+06]. The ISA88 standard processes a recipe of hierarchical management of batch control and frameworks to segment batch manufacturing processes. The separation of product and process by the hierarchical structure enables flexibility and reuse of equipment, as well as easier integration of new equipment or alteration of the production flow. ISA88 separates batch management into Procedural Control Model and Physical Model; see Fig. 12.10.

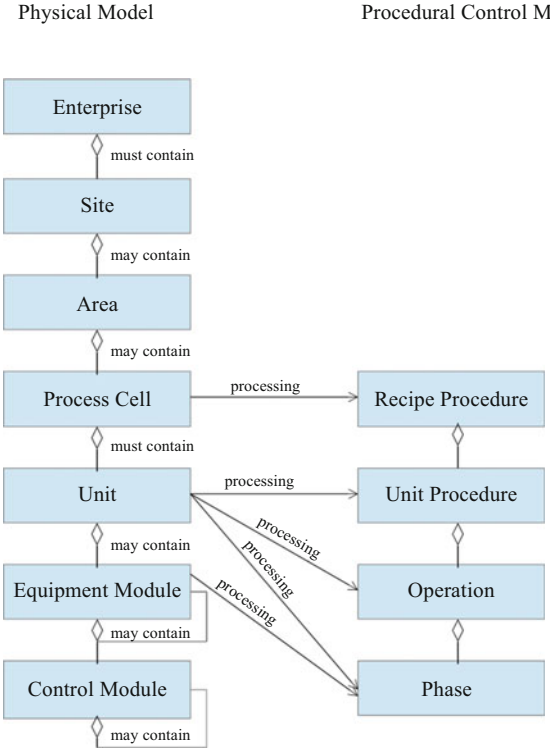

Fig. 12.10 Overview on the ISA88 structure [Com+95]

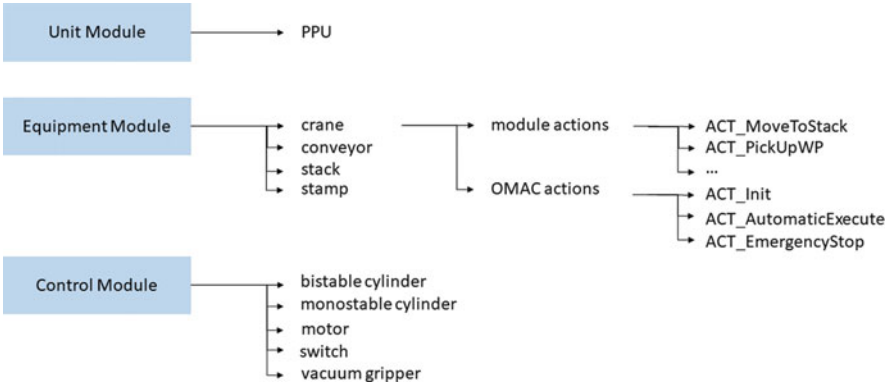

Fig. 12.11 Overview on the module structure of the PPU

The Procedural Control Model describes the production flow of the current batch as it is linked to the physical equipment. This model consists of recipe procedure, unit procedure, operations, and phases. The production flow of a single unit is controlled by the unit procedure. Each unit has an exclusive operating unit procedure, which is bound to this single unit. However, a single unit is allowed to contain more than one operation, but only one operation can be active at a time. An operation consists of a set of phases (tasks), which are allowed to run simultaneously. The xPPU as a unit is part of a process cell. A unit is an independent set of equipment, running a recipe and operating on a batch simultaneously. Units must contain equipment modules, which could contain equipment modules themselves. As a functional group of equipment, it fulfils a finite number of tasks, only one activity at a time. Equipment modules are, for instance, the crane, the conveyor, the stack, or the stamp. An equipment module contains control modules, and these are the most basic elements of an equipment, such as motors, cylinders, or valves (Fig. 12.11). The SysML models of the xPPU are provided as editable files in the modelling environment "Papyrus" [Lan+09]; see Fig. 12.12. The models are available for the community.[4]

12.2.2 Products, Processes, and Resources as Industry 4.0 Enabler for xPPU

The three viewpoints, products, processes, and resources (PPR), were initially supported by Automation Markup Language (AML) [SD09, SL15]. AML is extensible in a way that it can be enriched with other data formats to increase acceptance

[4] https://github.com/x-PPU/Models.

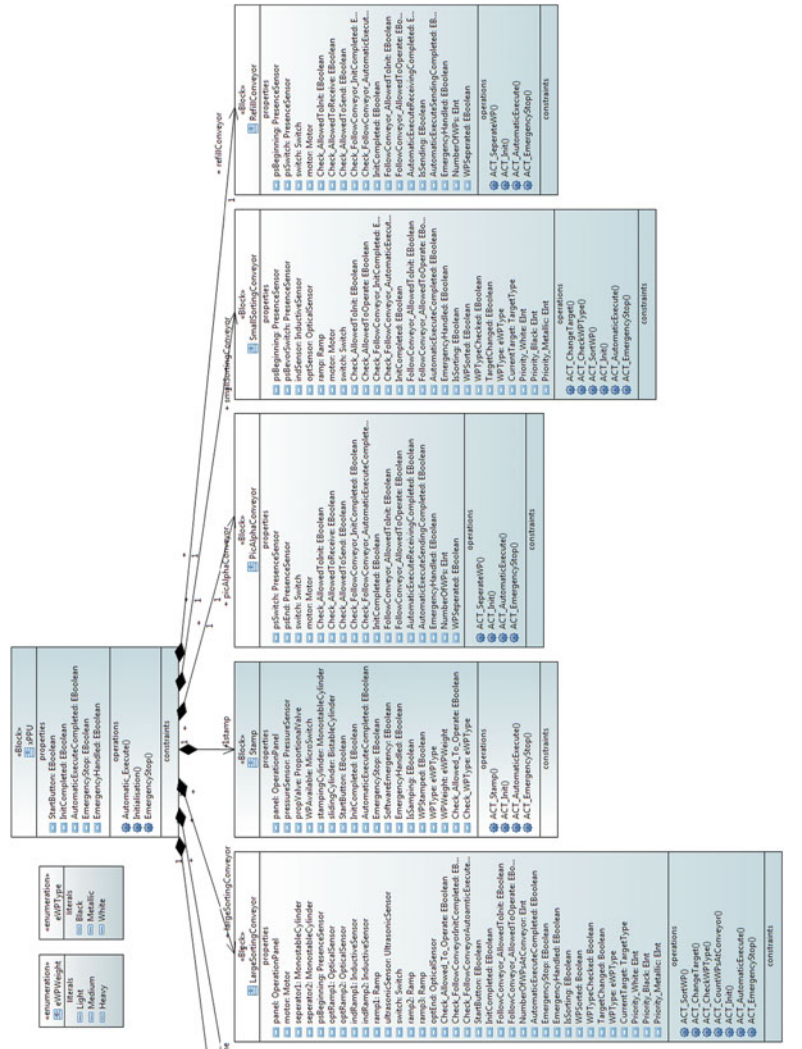

Fig. 12.12 Overview of the xPPU's class diagram of Scenario 15

[Dra+08]. The topology of the production system and the included relations are represented by the use of CAEX, while CAD information is represented in COLLADA and control related logic data is expressed in PLCopenXML [SL15]. AML and the corresponding engineering process intertwine the different domains more closely, increase flexibility, and reduce engineering costs [Lüd+10, SL15]. Schmidt et al. [Sch+14] also ascertained that an integrated approach like AML is more promising, concerning a reduction in engineering costs, than further optimisations of individual tools.

Schmidt et al. [Sch+14] also identified the main requirements towards a cross-disciplinary exchange format in the same expert survey. Firstly, industry needs a consistent, lossless exchange of data among heterogeneous engineering tools, which is enabled by AML itself. Secondly, version management is necessary. This is partially realised within AML but should be further supported by additional tools [MB15, Ber+16]. And thirdly, the involved disciplines need means to ensure consistency. This is not part of AML itself, but external reasoning tools are being developed [Bif+14, Sab+16]. The AML architecture is structured into four parts [SL15]. These are the role class (RC) library, the system unit class (SUC) library, the interface class (IC) library, and the instance hierarchy (IH). The RC library defines the general object semantics, the SUC library includes reusable components, and the IC library defines relevant interfaces. Interfaces can thereby be either internal, i.e. among different instances, or external, in which they reference external data such as COLLADA files. Within the IH, finally, the actual project data is specified.

These three viewpoints PPR are closely interconnected and strongly influence each other. At the example of the xPPU, the resource stamp executes the process stamping on a product (i.e. workpiece). Within AML, engineers can express these three viewpoints as separate hierarchies derived from the roles, product, process, and resource, respectively. To connect these three hierarchies, engineers can use the interface class PPRConnector [SD09]. If necessary, more specific subclasses can be specified. Trentesaux et al. [Tre+13] propose to describe products via jobs, i.e. processes, executed by machines, i.e. resources. This way of thinking conforms to the PPR concept and results in appropriate role classes and four SUC libraries in AML. The role classes for machines and products provide templates including typical attributes such as the article number or weight for products. The process role class library already provides different sorts of generic machine operations. These range from logistics, exemplarily (un)loading, to manufacturing, exemplarily stamping; see Fig. 12.13.

The three relevant SUC libraries are the product model, the process model, and the resource model (Fig. 12.14). Within the product model, engineers specify the products throughout all relevant stages in the production process. In case of the xPPU, this includes, e.g. a “black plastic workpiece” but also the resulting “product 3”.

The process model defines jobs that can be executed by machines to realise products. Jobs are defined as “sub-assemblies” by [Tre+13] and include production steps. Jobs are included in the SUC library, because they may apply to different products and can then simply be instantiated multiple times. Additionally, the

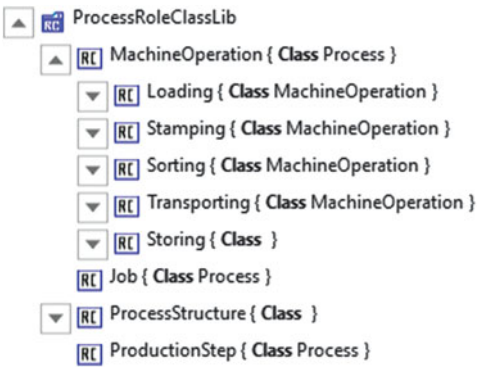

Fig. 12.13 An excerpt of process role class library

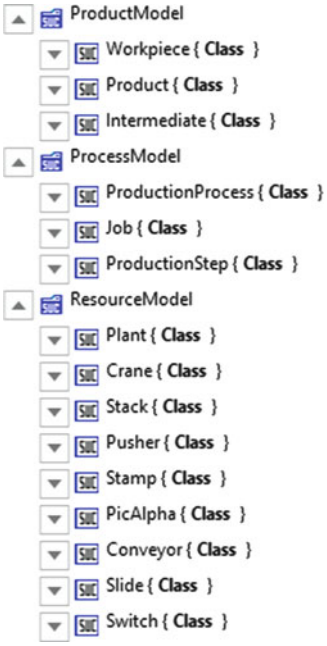

Fig. 12.14 PPR of SUC library

process model includes an SUC production process that is used as a superordinate element in the instance hierarchy to group the jobs. The resource model finally includes all available sorts of machines, namely, the crane, the stack, the pusher, the stamp, the PicAlpha module, and the conveyor. Additionally, the machine model includes the class plant for grouping the machines within the instance hierarchy.

To combine the three different points of view of the PPR concept, three interfaces are derived from PPRConnector within the InterfaceClassLib. Connector DeviceOperation associates machines with production steps; Connector WPOperation represents the connection between processes and their workpieces, i.e. their inputs;

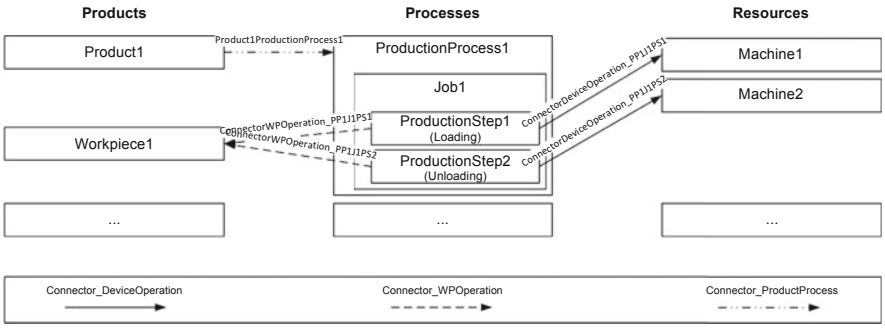

Fig. 12.15 Schema for PPR connections

Fig. 12.16 Representation of xPPU's topology on instance level

and Connector ProductProcess is used to connect products to their respective production processes (Fig. 12.15).

The three different SUC libraries have corresponding instance hierarchies, in which SUC classes are instantiated to represent reality. The resource SUCs are used within the instance hierarchy to represent the actual plant in the form of a topology. This topology, which includes the crane, the stack, the pushers, the stamp, the PicAlpha module, and the conveyors, is depicted in Fig. 12.16. A bird view of the layout of the xPPU with its resources is presented in Fig. 12.17. The bird view also depicts the paths of the various products.

The process instance hierarchy includes the production processes for the different products. Each production process consists of at least one job, which, in turn,

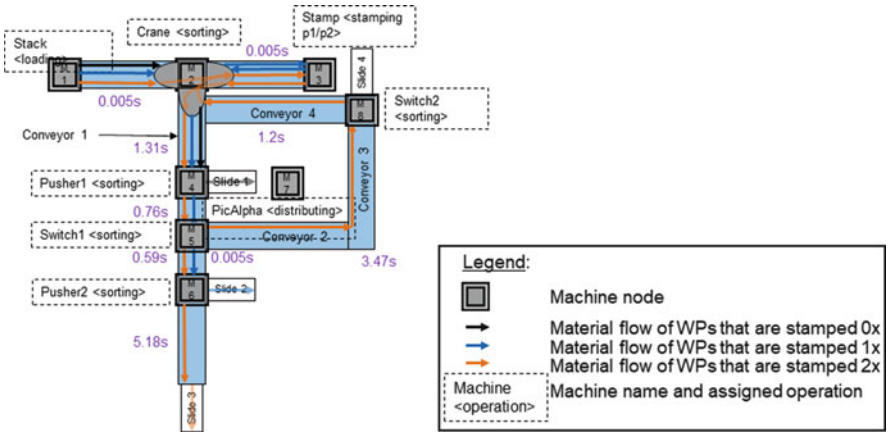

Fig. 12.17 Bird view of the xPPU

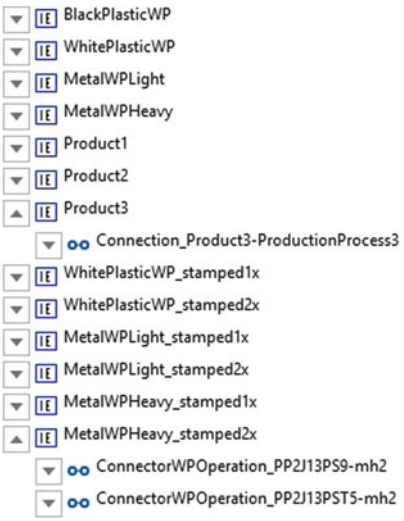

Fig. 12.18 An excerpt of internal links of PPR of the xPPU

consists of at least one production step. Since attributes of the jobs, such as the processing time, may differ depending on the product, the jobs are associated with production processes and thus with products. The IH also include the links among the PPRConnectors, which tie the different views together. Figure 12.18 gives an overview of the links realised to represent the manufacturing of products 1, 2, and 3. These internal links also include all TransportConnections, which are used in two ways. First, they are used to link resources, which can transfer products. That is, all possible ways a product can take through a plant are represented. Second, every sorting or transporting production step is connected to the resource that provides

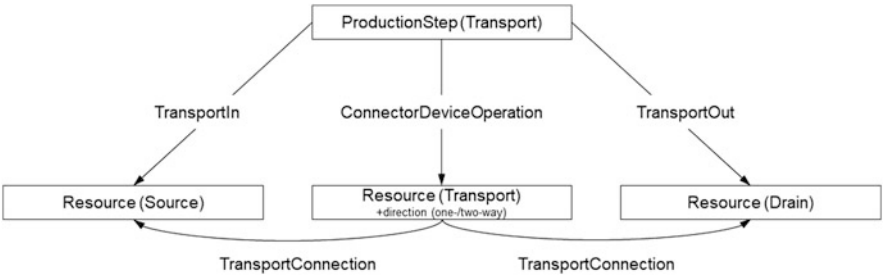

Fig. 12.19 Connections among resources and between processes and resources

the product (source) and to the resource the product is transferred to (drain); see Fig. 12.19.

Changes in any of the three views may lead to changes in the other views, too. For example, if Conveyor2 is not working anymore, it can be substituted by the PicAlpha module. This is because the PicAlpha module can not only change the order of products, but it can also fulfil a transport task from Switch1 to Conveyor3. Thus, a redundancy concerning this specific task is created, and the alternative production process may be chosen. Changes to a product may also propagate to the other domains. Mainly, it will influence the associated production process, which possibly influences the linked machines. Through the interconnectedness of the different views, the propagation of changes can be easily traced.

12.2.3 Simulation Models for Testing

For each evolutionary stage, a simulation model was created in Matlab/Simulink including an own 3D visualisation, which represents the behaviour of the real demonstration system. The Matlab/Simulink models are available to the community via an OPC interface (object linking and embedding for process control) with known time constraints. An OPC interface is also available for the demonstration system itself, in order to keep the cost of the conversion during the evaluation on the real laboratory installation low. Alternatively, it is possible to create the control code directly in IEC 61131-3 or the state chart diagram (CODESYS) and to evaluate this simulation or the real system coupled to the OPC. Due to the properties of the PPU, the OPC interface is sufficiently for most project approaches. In addition, a PLC-based controller is available in the TUM laboratory for direct connection to automation technology.

12.2.4 Generating PLC Code Using UML Models

In the work of [Wit13], the authors provided a methodology to generate PLC software based on UML diagrams. Therefore, the PLC software of the xPPU's evolution scenarios from Sc0 to Sc13 was generated based on UML diagrams. The UML diagrams consist of UML class diagram for the implementation of a PLC-project structure as well as state chart diagrams and activity diagrams for the implementation of the behaviour of the elements of the UML-PLC (e.g. classes, methods). This approach was implemented using the development environment CODESYS. Moreover, the PLC code of the xPPU scenarios was developed according to the international standard IEC 61131-3 [JT10], which is widely used for programming PLCs in the domain of aPS, comprising five programming languages: two are textual (the assembler-like Instruction List (IL) and the Pascal-like Structured Text (ST)) and three are graphical (Ladder Diagram (LD) in the style of circuit diagrams, Function Block Diagram (FBD), and Sequential Function Chart (SFC)). The PLC code of the evolution scenarios of the xPPU is available for the community.[5]

12.2.5 Detecting Anomaly Behaviour by Analysing Signal Data of the xPPU

The available data set includes the values of the different variables in xPPU in each time stamp, while it is working based on different scenarios (time stamp, variable, value). The recorded variables are most of the time logical variables, which take only zero or one values. However, there are a few variables like analogue sensors, which have analogue values (e.g. analogue pressure). Since the most variables in the data set are logical variables, it can be considered that the data set includes the activation and deactivation information of the variables in each time stamp while xPPU is running. This data set can be used for different purposes. One example of the usage of this data is to recognise the cycle of activation (deactivation) of different variables. Then it would be possible to check that during the activation (deactivation) period of one variable, which other variables are activated (deactivated). It can be helpful to recognise the dependencies between variables and detect what is the consequent effect of activation (deactivation) of some variables on the other variables. In addition, the hierarchical dependencies among variables can be useful to get some information concerning the hierarchical structural of the plant using the data-driven methods. For example, it can be found which part of a plant (including special variables) is a subset of a larger part of the plant (with more variables and longer activation period). By using both the dependencies between variables and the

[5] https://github.com/x-PPU/PLC_TwinCAT_Projects.

Fig. 12.20 Example of sequence of activation and deactivation of different variables

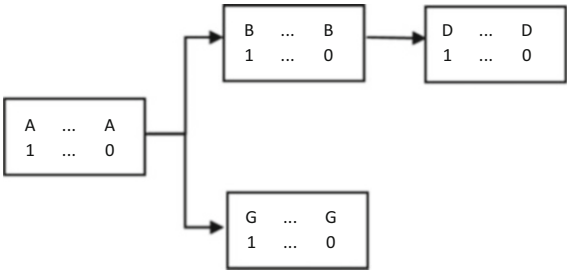

Fig. 12.21 Hierarchical structure using the activation period of different variables

hierarchical structure of the plant, it will be possible to detect the source of some anomaly behaviours of the plant using the data-driven methods. In this case, we can analyse that after the activation (deactivation) of some variables at the top of the hierarchy, we expect to see the activation (deactivation) of which variables at the bottom of the hierarchy. Then, this information can be used to predict the behaviour of the plant, and if the behaviour of the plant is not based on our expectation, we can see which variable (or variables) does not have a proper value, which leads to abnormal behaviour.

The first approach, which is used to achieve this goal, is to derive a tree structure from the data based on the activation (deactivation) period of the variables. In this case, the variables are sorted from the longest activation (deactivation) period to the shortest period. Then, we try to detect which variables with shorter period can be seen during the period of the variables with longer activation (deactivation) period. This hierarchy can lead to the shortest activation duration of one variable (activated and deactivated immediately) at the lowest level of hierarchy. The Deep First Search (DFS) algorithm is used to construct the tree structure based on the activation (deactivation) period of variables. In this tree, the children of each node can be considered as the set of variables, which are dependent to the variables in this node, and they are activated (deactivated) after the variable in the parent node. Here, there is an example to illustrate the method (Fig. 12.20). Based on the activation period of the variables and DFS algorithm, the tree structure in Fig. 12.21 can be constructed.

The first result of this analysis on xPPU data is provided and discussed with the expert. Based on expert knowledge, most of the hierarchical relationships between variables in the tree structure are meaningful. The next step of this analysis will be to detect the anomaly behaviour of the plant based on the expected values and dependencies between variables. It is worth mentioning that the proposed approach can also be useful in analysing PLC software. In this case, the structure of the

code, e.g. function blocks, can be considered as an input into the DFS algorithm. Therefore, DFS tree can be constructed based on the function calls in each function block. Parent nodes include the functions in high level of hierarchy which call some other functions, and the called functions can be considered as the children nodes. The benefit of this analysis is that it is possible to follow the code based on the structure of the constructed DFS tree. In this case, we expect that a function call, which is a child node, has to be done completely to make sure that the higher-level function in the parent node can also be done consequently. This approach can be useful to detect which function blocks in children nodes do not perform properly which leads to the propagation of the abnormality to a higher level of hierarchy of function blocks.

12.2.6 Maintaining Security of the xPPU

The middleware application that allows to remotely access and interact with the xPPU demonstrator increases the opportunities for malicious users to compromise the information and correct behaviour of the xPPU and its interacting parties (e.g. CoCoME). Hence, it is necessary to improve the security of the middleware application, as well as its environment in order to reduce the vulnerabilities that might be exploited for malicious purposes. Within the context of the xPPU demonstrator and CoCoME, an exemplary goal of this kind of malicious purposes may be to provide erroneous information to both the xPPU and CoCoME which may result in unnecessary production being carried out (e.g. a wrong number of product orders being placed) that may increase costs, incorrect warehouse information provided to the supply chain that may affect sales and generate disgruntled and unsatisfied customers, and manipulation of the cyber-physical system that may compromise the safety of personnel, among others.

The main focus of security from the point of view of an information system developer or service provider is to protect against targeted attacks towards the system. However, from the point of view of system integrators and asset owners, the main focus is to correctly implement the security mechanisms provided by the product vendors and integrate any changes that may be required based on system- and domain-specific characteristics and requirements. Within the context of the ongoing work of the xPPU demonstrator, these two points of view are considered on the middleware application and the automation environment, respectively.

Many methodologies to elicit system requirements exist [Poh10, NE00]. The two methodologies most widely used to elicit security requirements are to carry out risk assessments and derive them from well-defined documentation such as guidelines, regulations, laws, or standards.

Risk assessments are commonly performed at later stages of the development life cycle (e.g. after design) or after a system has already been deployed, as previous knowledge of the target system and its assets is required. It also requires participation from multiple stakeholders and knowledge regarding the threat landscape that

such assets may be susceptible to and their likelihood and impact. The goal of a risk assessment is to identify appropriate countermeasures that allow to decrease both security risks and unnecessary security costs.

On the other hand, security elicitation from well-defined documentation is more straightforward. In this type of elicitation, the requirements are abstracted by analysing such documentation and verifying its applicability to the target system based on its characteristics, target environment, or specific security assets. The goal of this type of elicitation is to prove compliance with well-known regulations. This compliance may be important for accountability and/or product or system certification.

As both the middleware application and the xPPU exist within the context of a demonstrator and are located at a facility within a wider educational infrastructure, it is assumed that the likelihood for any type of risk is unknown, as this may depend on the infrastructure itself. Therefore, in order to identify their security requirements, these are abstracted from well-defined documentation rather than from a risk assessment.

The documentation considered to elicit these security requirements is the ISA/IEC 62443 standards of Security for Industrial Automation and Control Systems. This set of standards was selected as they are the de facto standards in the industrial field. They are divided into multiple parts, two of which address the two points of view previously mentioned.

The IEC 62443-3 parts target system integrators; hence, they are relevant to abstract security requirements that can be applied system-wide (i.e. environment). On the other hand, IEC 62443-4 parts target component developers or manufacturer; hence, they are relevant to abstract component-specific requirements such as the middleware application. These requirements are found in parts IEC 62443-3-3 and 62443-4-2, respectively.

IEC 62443-3-3 provides a set of security requirements that must be fulfilled for specific systems based on their desired security level. On the other hand, IEC 62443-4-2 provides a set of security requirements that must be fulfilled by components of the system.

From these standards, a set of ten security requirements have been abstracted. These requirements can be summarised in three classifications: security requirements to ensure secure communication among the interacting parties (i.e. PLC—middleware—client), security requirements that ensure user authentication, managed access control and accountability, and security requirements to ensure secure and reliable session management.

The validation of these security requirements has been carried out implementing the Secure Tropos methodology [MGM05]. This methodology allows to analyse the interaction among actors, tasks, goals, resources, and their dependencies in order to better understand the system and possibly identify missing security requirements or constraints that were not considered previously during requirement elicitation; see Fig. 12.22.

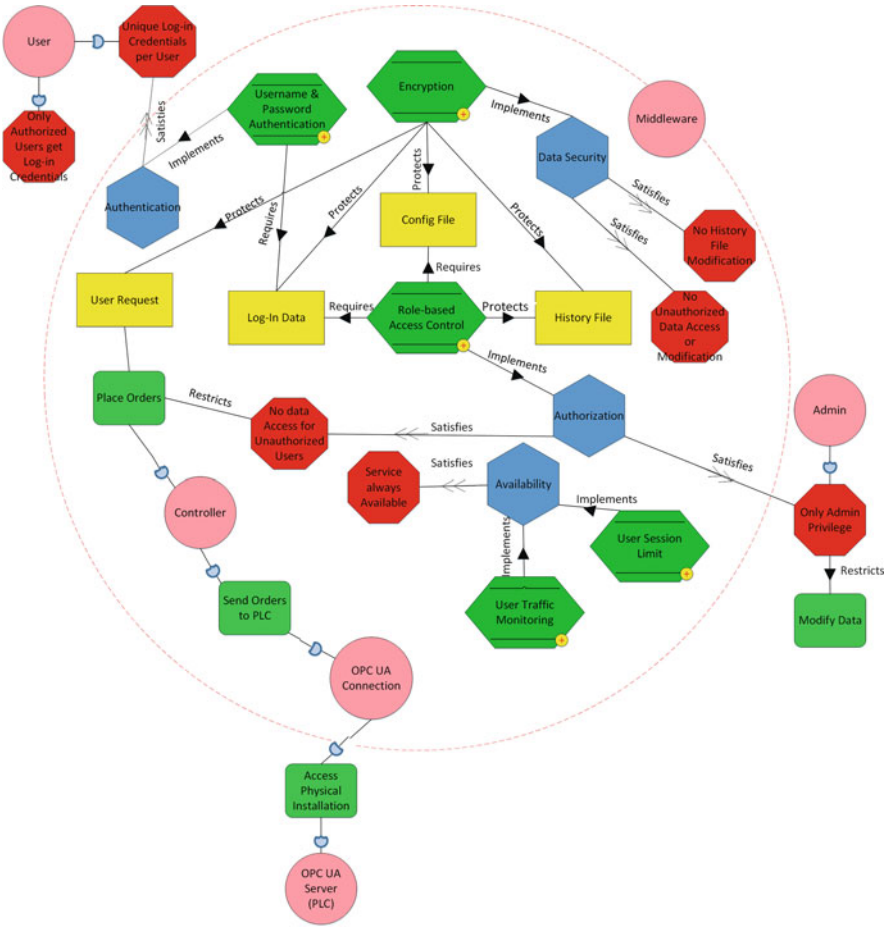

Fig. 12.22 Secure Tropos diagram [Tsc18]

The aforementioned security requirements involve modifications to the middleware application. These modifications ensure the protection of the application against targeted attacks.

However, as it is well known from security incidents and other reports provided by multiple research and analysis authorities [CS17, Cor17], the most common approach to compromise a system is to exploit vulnerabilities. These vulnerabilities may or may not be found on the targeted components. This occurs as security attacks commonly exploit these vulnerabilities in unsuspecting or vulnerable parties (e.g. system components or personnel) that provide means to reach the attack target.

In order to decrease system vulnerabilities, the requirements presented in IEC 62443-3-3 are relevant. From these requirements, a set of appropriate open-source security solutions have been identified. These security solutions are being integrated

into the xPPU demonstrator in order to provide protection to key system components and locations. One of these solutions is a vulnerability scanner (i.e. OpenVAS) that allows, among other things, to scan system components in order to identify possible vulnerabilities in them.

The usage of pre-existing security solutions provides seamless integration with automation systems. Depending on the case, it may be possible to install them and manage them without affecting the automation system itself during runtime. This is especially important, as one of the many fears of security in industrial systems is that they may negatively influence the performance or behaviour of the automation system.

As the technological and threat landscape keeps changing, so must these security considerations. Security, just as information systems, must evolve. The lifetime of the hardware components of an automation system may last years, and the lifetime of its software components may last months or years; however, the lifetime of security is extremely unpredictable. Hence, it is necessary to continuously monitor any new requirements that may arise in order to maintain the same security level throughout the whole system's lifetime.

12.3 Benefits and Deliverables of Industry 4.0 Demonstrator for the Outside Community

Recent trends in industrial digitalisation, i.e. Industry 4.0, require the systems to be connected with each other. In particular, boundaries between the information systems and the production systems are blurred in this era. The Industry 4.0 demonstrator (Chap. 4) implements the common use cases of the information system, which is represented by CoCoME, and the automated production system, which is represented by xPPU. With the integrated case study, it can be demonstrated that production system information is more visible and controllable from the information system side and end user needs are more visible from the production side. Therefore, Industry 4.0 demonstrator enables researches on the integrated environments such as ordering a customisable product, creating a production plan for a customisable product on multiple abstraction layers, and observing the progress of batch size on productions. The CoCoME was extended to allow integration with an aPS plant. An example of a plant is the interface offered by xPPU. Further, CoCoME enables the domain experts to use a mock-up of a plant without actually connecting a physical plant. This allows simulating the integration using an arbitrary plant [Bic+18]. Furthermore, the xPPU was extended with the interface to be connected to the remote users (including external systems). This interface allows the users to connect to the xPPU over the web service either to run it or to gather some data about the plant from the remote site without any effort to travel to the plant (see Fig. 12.23). Also, uses can take the advantage of the model-based dynamic reconfiguration

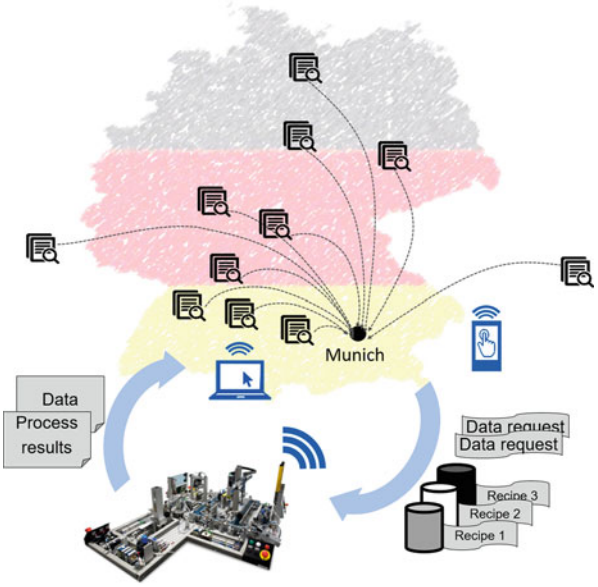

Fig. 12.23 Remote connection of the xPPU Industry 4.0 interface

implementation from this case study. In addition, security issues which might happen on these interfaces are also envisioned and accessible (see Sect. 12.2.6).

The Industry 4.0 demonstrator shows the prototypical implementation of cyber-physical production systems (CPPS) by containing the following characteristics [VBF12]:

Architecture models: by using ISA88 architecture model for the plant that provides hierarchical structure and process-based model for batches and coupling modules and processes.

Data analysis: by allowing synchronising values over plants or systems as well as gathering and transferring operation-related values (online and offline).

Flexible production unit: combining the generated services from the systems in different ways support various recipes which allows flexible production.

Digital networks and interfaces for communication: by enabling both OPC-UA connection to the PLC side and RESTful web service to the user side, the middleware enables to connect the remote user to the plant (Fig. 12.24). The middleware places between the xPPU side and the remote user side. Using this connection, users can either execute the desiring operations (e.g. transporting or sorting materials) or get information about xPPU (e.g. execution history or archived variable values). Android mobile application[6] as well as a web page is also available to ease the user connection (see Fig. 12.25).

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

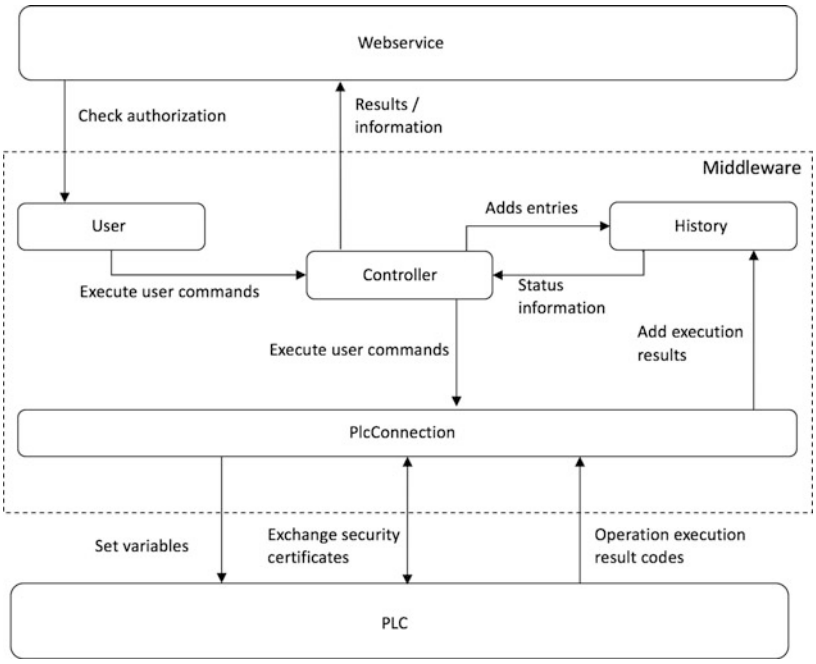

Fig. 12.24 Middleware architecture of the xPPU Industry 4.0 interface

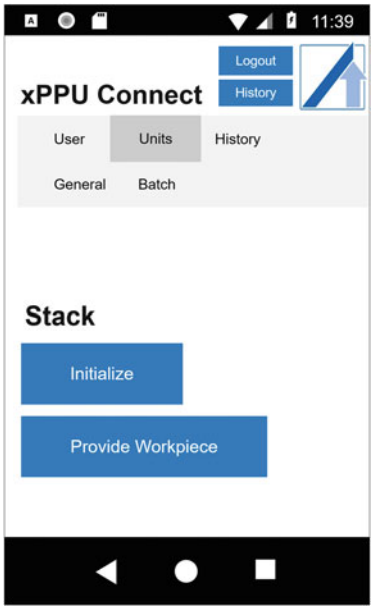

Fig. 12.25 Mobile application of the xPPU Industry 4.0 interface

Worldwide distribution of data, high availability, and access protection: on both sides of the systems' connection, security is implemented by setting standard authentications.

Chapter 4 describes the architecture of the Industry 4.0 case study (i.e. the integration of CoCoME and xPPU). This architecture can be used for applying approaches to domain-spanning change impact analysis. An example of a change can be the modification of the interface provided by the xPPU. This change can propagate to CoCoME, which uses this interface. Further, the case study allows analysing other quality attributes such as security or performance by the approaches in Industry 4.0 context [Hei+18a].