Towards Modeling Cyber-Physical Systems From Multiple Approaches

Márton Búr[1,2], András Vörös[1,2], Gábor Bergmann[1,2], Dániel Varró[1,2,3]
[1]Budapest University of Technology and Economics, Department of Measurement and Information Systems, Hungary
[2]MTA-BME Lendület Research Group on Cyber-Physical Systems, Hungary
[3]McGill University, Department of Electrical and Computer Engineering, Canada
Email: {bur, vori, bergmann, varro}@mit.bme.hu

***Abstract*—Cyber-physical systems are gaining more and more importance even in critical domains, where model-based development and runtime monitoring is becoming an important research area. However, traditional approaches do not always provide the suitable toolset to model their dynamic characteristics. In this paper, we aim to overview and highlight the strengths and limitations of existing runtime and design time modeling techniques that can help runtime monitoring and verification from the viewpoint of dynamic cyber-physical systems. We evaluated instance modeling, metamodeling, and metamodeling with templates, and provided example use-case scenarios for these approaches. We also overview the applicability of SysML in these contexts.**

I. Introduction

Critical cyber-physical systems (CPS) are appearing at our everyday life: healthcare applications, autonomous cars and smart robot and transportation systems are becoming more and more widespread. However, they often have some critical functionality: errors during the operation can lead to serious financial loss or damage in human life. Ensuring trustworthiness of critical CPS is an important task in their development and operation. CPSs have complex interactions with their environment, however, environmental conditions are rarely known at design time. In addition, the behavior of CPSs is inherently data dependent and they have smart/autonomous functionalities. These properties make design time verification infeasible. In order to ensure the safe operation of CPSs, one can rely on runtime verification. Various techniques are known from the literature for monitoring the different components constituting a CPS [1], however they do not provide system level assurance. Moreover, traditional monitoring techniques do not cover data dependent behavior and structural properties of the system.

Runtime verification is a technique to check if a system or a model fulfills the specification during operation by observing the inputs and outputs. It extracts information of a running system and checks whether it violates certain properties. We plan to use models at runtime as a representation of our knowledge of the systems. The model is built and modified according to the various information gathered for runtime verification.

This paper is partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

Models created at design time can be subject to traditional verification techniques. They can guarantee the correctness of the system design. In contrast, runtime techniques analyze the runs of the system during its operation and they can reveal errors in the real life implementation.

Our approach is similar to the one of model-driven engineering (MDE) that facilitates problem description by utilizing domain specific languages such as domain specific models. In terms of this paper models are considered as typed graphs.

In this paper we introduce our envisioned approach for modeling dynamically changing, extensible cyber-physical systems. We aim to overview possible model-based approaches for IoT/cyber-physical system development, and will investigate how these techniques can provide support for runtime modeling of such systems.

II. Vision

According to our approach, there are two main modeling aspects: *design time* and *runtime*. For each aspect there are three main pillars of modeling:

- *Requirements* specify the properties the system must hold.
- *Environment information* represents the physical environment.
- *Platform information* describes the available sensors, computation nodes and actuators in the system, as well as encapsulates functional architecture with deployment information.

In the followings we summarize the goals of modeling at various phases of the development.

a) Design time models: Requirements, environment information, and execution platform information are present in a model of a CPS. The design time models describe (i) initial configuration, such as structure and topology, or (ii) behavior. The information contained in such models is static, the information captured by design time models does not change during runtime. The implementations of such designs are typically deployed software components or configuration files. During the design process the system also has to be prepared for different operational contexts, because most of its details are only known at runtime.

b) Runtime models: Runtime models, also called *live models*, capture information about the system dynamically. At runtime the actual system configuration is known, as well as sensors can provide details about the operational context of the system. From design time models deployment artefacts can be obtained, which link the design time information to the runtime models. This information is then used in the live model, eventually amended with additional knowledge.

Since the role of the design time and runtime models differ in the system life cycle, parts of runtime information may be omitted from design time models, such as values of data streams and events. Similarly, design time information may only be present in the runtime model in an abstract way.

For example, a camera and a computer is represented in the design time model, but the stream data is not available at design time, so that it is not present in the design model. Another example is when a controller and its parameters are represented in a design time model, but the live model for the controlled process only has a boolean value expressing whether the output complies to the requirements, but the used controller parameters are not included.

The purpose of the live model is both (i) to capture domain-specific information about the system and its functions, (ii) to describe the heterogeneous, dynamically changing platform, (iii) to describe its operational context and environment, and also (iv) to check runtime verification objectives.

Our vision of using live model-based computations for different purposes such as operation and monitoring of cyber-physical systems is depicted in Figure 1. In order to efficiently handle data obtained by sensors, we envision *sensor data integration* step to normalize and transform data so that it is adapted to the *live model*. *Computation* refers to the process of determining the required actuation to the physical environment and live model updates using the current state of the runtime model. This concept shown in Figure 1 can be used both for *live model-based control* and *runtime monitoring* of the system. The former actuates system processes, while the latter only observes the system and the environment using the live model.

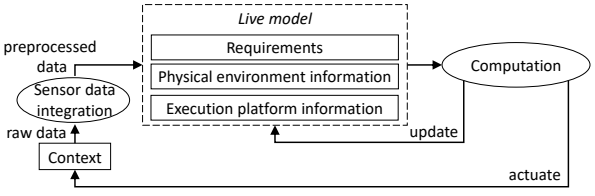


Fig. 1. Vision of using a live model

In terms of classical control theory concepts, this approach represents a *closed-loop control*-like mechanism, where the *physical processes within the system context* together are regarded as the *plant*, and *controller* tasks are realized by the *live model-based computation*. The processes in the system context may already be controlled processes.

III. Modeling Aspects

In this section we introduce and discuss both design time and runtime modeling approaches for dynamic cyber-physical systems. We detail the support provided by the *SysML* standard [2] for the approaches, as well as point out their missing features. We also illustrate the main challenges of defining and creating both design time and live models using an example of a fictitious smart warehouse. In example autonomous *forklifts* are operating, which are equipped with onboard cameras to detect changes in their environment. Due to space limitations we include examples about execution platform models, while modeling the requirements and environment information are not discussed.

A. Metamodeling

One of the basic modeling approaches is metamodeling. Using metamodels, one can define (i) node types, (ii) node attributes and (iii) relationship types. This allows the modeler to describe constraints regarding the overall structure of the system model, on the type-level.

For cyber-physical systems we consider each attribute as read-only by default, as they represent information sources, e.g. sensors. A special type `signal` denotes that the value of the property is time-varying, based on the data received. Signals can be *discrete time signals* or *continuous time signals*. In case of discrete time signals, their value may be changed at given time instants, but stays constant in between. For continuous time signals change in its value may occur anytime. Similarly to `signal`, the type `event` represents time-dependent information, but it is provided only at certain discrete time instants. There are signals of different types in the metamodel fragment depicted in Figure 2 as well, such as *feed* for a camera or *currentRPM* for ECUs.

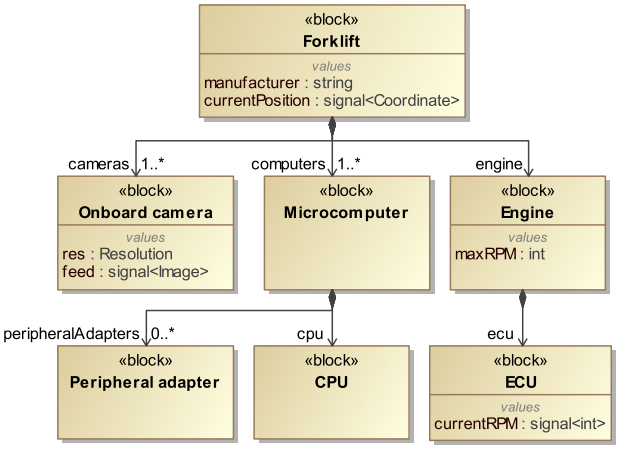


Fig. 2. Example metamodel with containment edges and attributes

Metamodels at design time can be used to create a functional model that satisfies the requirements, define platform model structure, and to describe the possible entities in the environment.

The part of the design time metamodel for the system representing the relevant platform information in our example system is illustrated in Figure 2. Containment edges show that the root container element is the *Forklift*, which contains an *Engine*, and at least one *Microcomputer*, and at least one *Onboard camera*. Microcomputers are further decomposed into *CPU*s and *Peripheral adapters*, while an engine encapsulates its corresponding *ECU*. Cross references between types in the model are not shown in the diagram.

According to our vision, models also hold information about runtime properties based on the requirements. In order to represent requirements as well, we defined *Goal*s for the type *SafetyCriticalDevice*, which is added as a supertype of forklift, as show in Figure 3. Goals are functions in the system that check whether the system holds the properties specified by the requirements. If a requirement stated that a device shall not collide with other devices, the corresponding goal would be a function that checks whether the device keeps enough distance from other trucks.

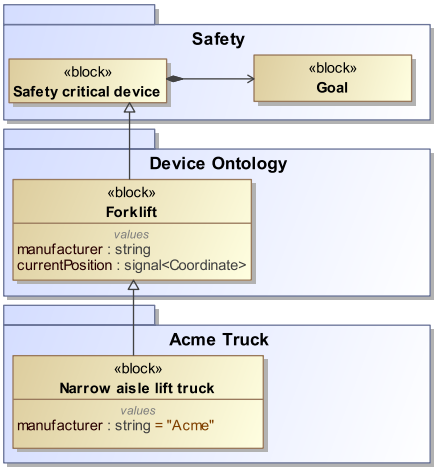


Fig. 3. Example for inheritance and packages for the forklift metamodel

Reusability is an important aspect in engineering. If a metamodel is already given for a domain, it is the best to have the corresponding model elements grouped as a toolset to make it available for reuse. For this reason, one can define *packages* that are similar to *libraries* in programming, containing model elements for the same domain.

When creating concrete applications, elements in general packages shall be specialized by *inheritance*, that can be used to specify fixed values for properties. Multiple packages can be used and specialized for the same application.

Figure 3 shows a possible packaging of types in our example. The package *Safety* contains essential concepts to include safety-related verification information in a model, the *Device Ontology* package is intended to hold different device types, such as forklift, and the *Acme Truck* package is the application-specific container. In our example *Narrow aisle lift truck* is a special type of forklifts, and each of its instances are manufactured by `Acme`.

The presented example figures show only views of the metamodel, so that additional relationships (such as containment, inheritance) as well as model elements, which are present in the model, are omitted from Figure 2 and Figure 3.

Design time metamodeling is facilitated at runtime to create instances based on the defined metamodel, where the model elements, relationships and properties are representing the knowledge-base of the system.

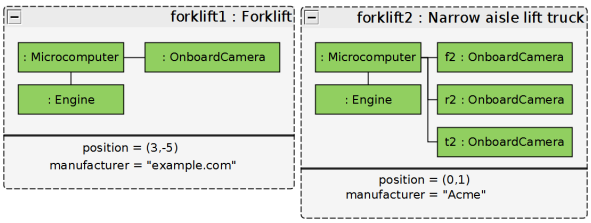
Support in SysML: SysML supports metamodeling by *block definition diagrams*. Focusing on cyber-physical systems, however, there is a need for elements that are not necessarily required for traditional software development. First, *flow properties* can be used to represent signals in SysML. Second, binding parameters can be expressed using the combination of default values and marking the parameter as read-only.

B. Instance Modeling

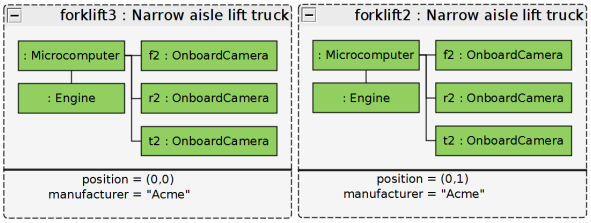
Instance models can describe a concrete configuration of the system, for which they can show multiple views. A typical usage for modeling requirements at design time is to create behavior models, such as statecharts.

Additionally, the environment and the platform can also be modeled on instance level at design time. However, it only has a limited usage to describe concrete arrangements – for example specifying test cases.

At runtime, however, views of the instance level model of the system platform and physical environment are essential. Considering our smart warehouse example, we can represent the system platform at time point t_0 with two *forklift*s, from which one is a special type of forklift named *narrow aisle lift truck*. The model is depicted in Figure 4a. Forklift main properties and their internal elements are also present. At a later point of time t_1, if the forklift leaves the warehouse and a new narrow aisle lift truck appears, the live model changes to as depicted in Figure 4b.



(a) Live model instance at t_0



(b) Live model instance at $t_1 > t_0$

Fig. 4. Live models at different points of time

Support in SysML: SysML has no dedicated support for instance-level modeling. However, the standard and best practices recommend using *block definition diagrams* (BDDs) to model snapshots of the system at design time.

C. Metamodels with Templates

One can define the structure of the instance models with metamodels. However, there are cases when it is desired to describe configurations including predefined attribute values and reference edges between certain types. For this purpose *templates* provide a solution to describe patterns in instance models. In our example metamodel, the forklift can have onboard cameras. Additionally in the template shown in Figure 5, we declare that *forklift* instances shall have a microcomputer unit, which has access to the engine and an onboard camera.

Fig. 5. Structure template of forklifts

Another template is defined for a microcomputer, where a CPU communicates with an ECU via a peripheral adapter. The ECU is not contained within the microcomputer, yet the peripheral adapter controls it, so that it is marked with dashed lines in Figure 6.

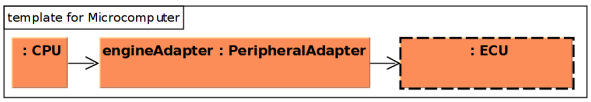


Fig. 6. Structure template of microcomputers

Furthermore, for the subtype *narrow aisle lift truck* this structure is changed, and the central microcomputer communicates with three different cameras, as depicted in Figure 7. This can be interpreted as there are exactly three onboard cameras in this type of truck, and the relation binding is formulated as $cameras = \{top, rear, front\}$. Additionally, for this specific type the *maxRPM* of the engine is 12000, and the front, rear and top camera resolutions are also bound to 800x600, 800x600, and 1504x1504, for each instance respectively.

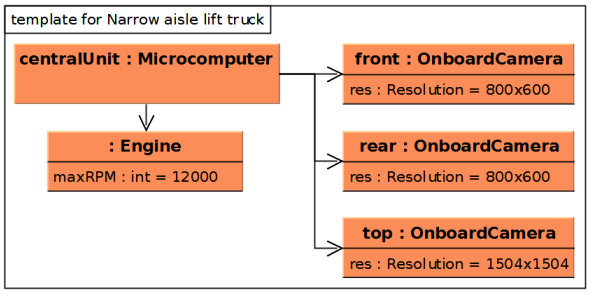


Fig. 7. Structure template of narrow-aisle lift truck

One of the main benefits of this approach is the description of certain runtime changes in the live model are more simple than in a purely metamodel-based case. For example, when a new forklift is added to the system, the change does not need to include the elements contained within the forklift or truck, for they are known from the template of the type, which can be a huge advantage when the model is changing frequently.

Support in SysML: SysML has *internal block diagrams* (IBDs) for template-like purposes. This description is also connected to a type, but is not found in traditional metamodels.

D. Strengths and Limitations of the Approaches

To conclude the introduction of the approaches, we summarize their strengths and limitations.

Strengths: The introduced modeling techniques support both design time and runtime modeling and analysis as long as the metamodel of the system is known at design time and remains unchanged during runtime.

Limitations: The cornerstone of each of the introduced approaches above is a metamodel introducing domain-specific types. It provides a basis for prescriptive modeling, which means model instances can only have elements of the types and relations defined within the metamodel. However, in case of dynamic cyber physical systems and IoT applications it is possible to extend the system at runtime with new components having new types. A solution for this issue can be to use ontologies, where types are assigned to the entities in a descriptive way. In such cases new objects can be classified using the types included in the ontology based on their capabilities.

IV. Conclusion

We overviewed design time and runtime modeling solutions to describe cyber-physical systems. We discussed metamodeling, instance modeling, and metamodeling with templates approaches, and provided use-case scenarios for them, as well as added examples how SysML supports each technique.

The provided overview has only covered a few main modeling aspects. In [3] the authors introduce their concept of evolutionary design time models that try to minimize the discrepancy between the design time and runtime concepts. They aim to minimize static information in design time models.

Additionally, there are many ways to extend the system description, one of them is modeling uncertainty. Uncertainty in cyber-physical system modeling is discussed in [4]. It is also a possible direction for model-based description of such systems to include probability and uncertainty models in graph-like live model-based representations.

References

[1] S. Mitsch and A. Platzer, "Modelplex: verified runtime validation of verified cyber-physical system models," *Formal Methods in System Design*, vol. 49, no. 1-2, pp. 33–74, 2016.

[2] Object Management Group, "OMG Systems Modeling Language," p. 320, 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/PDF/

[3] A. Mazak and M. Wimmer, "Towards liquid models: An evolutionary modeling approach," in *18th IEEE Conference on Business Informatics (CBI)*, 2016, pp. 104–112.

[4] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: A conceptual model," in *Proc. of Modelling Foundations and Applications (ECMFA)*, 2016, pp. 247–264.