Bridging the Gap between Structural and Behavioral Models in a Software-centric Environment

Noël Hagemann[0000−0001−9441−9889] and
Bernhard Bauer[0000−0002−7931−1105]

Software Methodologies for Distributed Systems, University of Augsburg,
Universitätsstraße 6a, 86159 Augsburg, Germany

Abstract. Induced by the last evolutionary step of systems, the virtualization and decentralization of systems is thriving leading to more complex systems. This causes the system behavior described by models to be split into additional models, creating gaps.
As a result, we present a novel approach that combines model artifacts describing the architecture of a system to recover the complete view of a system's behavior. Our design relies on model transformation to create a consistent model basis to enable cross-model connections. The combining process is carried out in two phases. First, an expert defines cross-model connections mapping behavioral models onto structural models. Second, these connections are used to derive direct connections between behavioral models to bridge the gap that emerged.

Keywords: Cyber-Physical Systems · Model-Driven Software Development · Model Transformation · Systems Modeling

1 Introduction

Advances in communications technology and global interconnectivity have led to the next evolutionary step of systems. As a result, technologies such as cloud computing (CC), the Internet of Things (IoT) and cyber-physical systems (CPS) have emerged. As defined by the NIST, "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[15]. While CC widens the gap between software and hardware by focusing on virtualization and decentralization, the IoT and CPS close the gap between the cyber and the real world. The IoT is defined in [1] as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies". CPS addresses several concepts also present in IoT but more focused in an industrial environment [17]. As a result, IoT devices are mostly used to monitor the real world [14] while CPS are designed to actively shape the world [8].

2 Problem Statement

The ongoing trend towards virtualization, decentralization and real-world integration is adding complexity to systems. In case of virtualization and decentralization, components of a system often no longer interact directly with each other but must first overcome hidden behavior to reach other system components. In a software-centric environment, we assume that each such component is controlled by software which results in its behavior being set by programming language. In this context, hidden behavior describes software without available source code. Cloud computing is a prime example of highly virtualized and decentralized systems as it allows companies to outsource the hosting of their IT infrastructure making it a matter of cost [13]. In comparison to traditional distributed systems, a fundamental difference in architecture is that software in distributed systems is tied to a specific physical machine while software in cloud computing is tied to a specific virtual machine [16]. This results in the software being fully abstracted from the hardware leading to new challenges. [12]. One of these challenges is reliability [11]. However, there are several approaches that address this challenge on the architectural level [9]. Anyhow, cloud computing significantly adds complexity to systems which creates new points of failure. In recent years, there was effort to apply the concept of cloud computing to the concept of the IoT to cope with its problems [7]. The IoT allows systems to react and interact with the real world introducing physical processes to systems. This applies to CPS as well since the IoT and CPS follow a similar approach as described in section 1. Physical processes add uncertainty to systems [18]. Uncertainty describes the lack of knowledge about a physical process i.e. state of the parent physical system, timing and nature of inputs. As a result, virtualization, decentralization and uncertainty are heavily adding complexity to systems.

A system, in general, is defined as a set of at least two components where each component has to affect at least one other component and has to be affected by at least one other component of the system [4]. Consequently, a component consists of properties which describe its behavior and connections to other system components. As stated in the specification of UML [10], UML model elements are categorized either as structural or behavioral. Structural Model Elements (SMEs) represent the static properties of a system that describe what the system is composed of. Behavioral model elements (BMEs) describe the dynamic features of a system that characterize how the components behave. Therefore, SMEs are used to model the properties of a system at a particular point in time while BMEs are used to model how they change over time. However, this categorization applies not only to the model elements but also to the models themselves. A structural model may contain BMEs but these are only used to request a particular behavior. In turn, a behavioral model may contain SMEs showing properties that do not change over time. As a result, we assume that models can be roughly categorized in either being structural or behavioral. Structural models often show the connections between system components such as UML component diagrams which show the interfaces of components and their structural connection. Behavioral models, in contrast, are often tied to a system

component such as UML state machine diagrams that represent the internal behavior of a particular system component. As a result, the behavior of a system is usually described by several models, each representing a particular aspect of the system. As system complexity increases, these models become more complex as well. Subsequently, the number of models to describe a system potentially increases.

As the number of models to describe a system increases, we see a strong need to regain a complete view of the behavior of a system. Since structural models usually focus on representing the connections between components and behavioral models usually describe the internal behavior of a component, there is no model that sufficiently contains the behavior of the entire system. To derive the behavior of the system as a whole, the behavioral models describing a system need to be connected or merged.

3 Bridging the Gap

In this section, we propose a novel approach for using structural models to connect the behavioral models of a system. First, the structural and behavioral models that represent the architecture of a system are reduced to mixed graphs. This is detailed in subsection 3.1. Thereafter, the models are merged and transformed to a directed graph to enable cross-model connections which is described in detail in subsection 3.2. Finally, cross-model connections are defined that link structural models to behavioral models, from which additional cross-model connections are derived to directly link behavioral models. This is detailed in subsection 3.3. In this section we focus on the syntax of the approach while the semantics are addressed in more detail in section 4.

3.1 Reduction of Structural and Behavioral Models

In the context of models, dependency is often expressed by directed edges. A dependency induces some kind of order when it is included in a structural model as for instance inheritance in class diagrams. A directed edge as part of a behavioral model, in turn, enables the formation of paths that characterize the behavior of a component or the interactions between components. An example of behavior models that describe the behavior of a component in detail are UML state machine diagrams or activity diagrams. As behavioral models describe the behavior of the system, their edges are directed. However, behavioral models may contain edges that are undirected as described in section 2. In addition, structural models may contain directed edges but they are usually undirected. As both types of model contain directed and undirected edges, we aim at reducing both types of model to mixed graphs. A mixed graph $M = (V, A, E)$ consists of a non-empty finite set $V(M)$ of elements called vertices, a finite set $A(M) = V(M) \times V(M)$ of ordered pairs of distinct vertices and a finite set $E(M) = V(M) \times V(M)$ of unordered pairs of distinct vertices as defined in [6]. We call $V(M)$ the node set of M, $A(M)$ the set of directed edges of M and

$E(M)$ the set of undirected edges of M. As the edges in $A(M)$ are directed, the edge $(a, b) \in A(M)$ represents a source-to-target connection where the node a is the source and the node b represents the target.

Let the set $X(S)$ contain all models that describe the system S structurally. In addition, let the set $Y(S)$ contain all models that describe the behavior of the system S. We reduce the behavioral and structural models of the system by transforming them into mixed graphs. Each structural and behavioral model is thereby transformed into exactly one mixed graph. This results in the sets $X_M(S)$ and $Y_M(S)$. The set $X_M(S)$ contains all mixed graphs derived from the structural models. In addition, the set $Y_M(S)$ consists of all mixed graphs derived from the behavioral models. The nodes of the behavioral model $x \in X(S)$ are thereby added to the node set $V(x_M)$ of the mixed graph $x_M \in X_M(S)$ that represents the model x. The nodes of the structural model $y \in Y(S)$ are added to the node set $V(y_M)$ of the mixed graph $y_M \in Y_M(S)$ representing the model y. The edges of the behavioral model $x \in X(S)$ are categorized as either directed or undirected and added to their respective set of the corresponding mixed model x_M. The edges of the structural model $y \in Y(S)$ are categorized and added in the same way.

3.2 Merging of System Architecture

To enable cross-model connections, the mixed graphs representing the system architecture are merged to form a consistent model basis. Therefore, the undirected edges of the mixed graphs are transformed into directed edges and the mixed graphs are merged into a single directed graph holding the whole system architecture. A directed graph $D = (V, A)$ consists of a non-empty finite set $V(D)$ of elements called vertices and a finite set $A(D) = V(D) \times V(D)$ of ordered pairs of distinct vertices as defined in [6]. We call $V(D)$ the node set of D and $A(D)$ the set of directed edges of D. As the edges are directed, the edge $(a, b) \in A(D)$ represents a source-to-target connection where the node a is the source and the node b represents the target.

As undirected edges can be considered bidirectional [6], undirected edges can be transformed to directed edges. The function $T : E(M) \to A(M)$

$$T(E(M)) = \bigcup_{(k,l) \in E(M)} (k, l) \cup (l, k) \tag{1}$$

transforms every bidirectional edge $(k, l) \in E(M)$ into the directed edges (k, l) and (l, k).

Let the consistent model basis be the directed graph $S = (V, A)$. Consequently, we define the set

$$V(S) = \bigcup_{s \in X_M(S)} \bigcup_{b \in Y_M(S)} V(s) \cup V(b) \tag{2}$$

as the union of all nodes of the mixed graphs that describe the system S. In addition, the set

$$A(S) = \bigcup_{s \in X_M(S)} \bigcup_{b \in Y_M(S)} T(E(s)) \cup A(s) \cup T(E(b)) \cup A(b) \tag{3}$$

of ordered pairs holding the edges included in the mixed graphs that describe the system S. The model thus consists of several independent model artifacts.

3.3 Directly connecting behavioral models

In context of the consistent model basis, we define a cross-model connection as an edge between two nodes contained in the consistent model basis that originated from two different model artifacts that described the same system. Therefore, let the nodes a and b be contained in the set $V(S)$. Let the nodes a and b originate from two different models x and y. Let the models x and y be contained in $X(S)$ or $Y(S)$ and describe the system S. The node a can be paired with the node b resulting in the edge (a, b) being added to the edge set $A(S)$ of the consistent model basis S. As the edges are directed, the node b can be paired with the node a resulting in the edge (b, a) being added to the set of directed edges $A(S)$. The semantics are addressed in the following section 4.

As the concept of transitive closures applies to directed graphs, we can use this property to directly connect behavioral models by deriving cross-model connections from walks. As defined in [6], a walk in D is an alternating sequence $W = x_1 a_1 x_2 a_2 \dots x_{k-1} a_{k-1} x_k$ of vertices x_i and arcs $a_j = (x_i, x_{i+1})$ from D such that $\forall x_i \in W(D)\ \exists v \in V(D) : x_i = v$ and $\forall a_j \in W(D)\ \exists a \in A(D) : a_j = a$ with $i = j = 1, \dots, k-1$. Consequently, if a path exists from $x \in V(S)$ to $z \in V(S)$, then the edge (x, z) can be formed without violating the model. This allows the behavioral models to be directly connected by transitivity, bridging the gap.

4 Case Study: Smart Home

In this section, a case study is performed, design decisions are discussed and semantics for this approach are defined. The case study is performed on a simple system measuring the outside temperature and sending the information to a device called the *Smart Mirror*. The device capturing the temperature is called *Thermometer*. It reads the outside temperature every minute and sends the result to the smart mirror over a TCP connection. Since a TCP connection acknowledges the receiving of packets, both components influence each other and therefore fulfill the constraints of the definition of system. After receiving a packet, the smart mirror processes the message and displays the temperature.

The system architecture is composed of structural and behavioral models. Figure 1 shows the system structurally as an UML component diagram. The component *Thermometer* provides the interface *Temperature* which is required

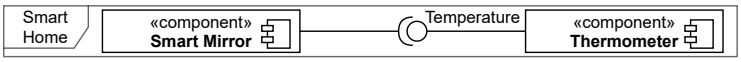

Fig. 1. Component diagram illustrating the system structurally

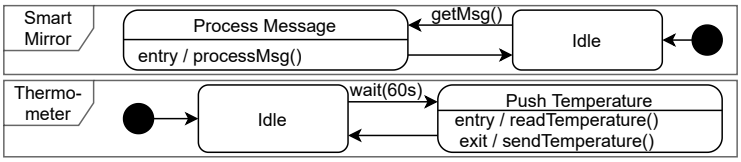

Fig. 2. State machine diagrams visualizing the behavior of each system component

by the component *Smart Mirror*. We chose UML for our representation because, despite a declining trend in its use, UML is still widely used [5].

In figure 2, the behavior of the components is described by UML state machine diagrams. They are framed by their respective name and are defined by two states. The smart mirror consists of an idle state and a state in which the temperature is collected from an internal sensor and transmitted to the network connecting both components. The device starts in the idle state and switches to the transmit state every sixty seconds. After the temperature is pushed to the network, the device changes its state to idle.

The thermometer includes an idle state and a state in which messages are processed and the new temperature value is displayed. This device also starts in the idle state and switches to the process state after a message is received. After the message is processed and the temperature is displayed, the device returns to idle mode.

As described in section 3, cross-model connections need to be enabled to bridge the gap. For this purpose, the behavioral and structural models are merged in this approach. To derive the behavior of the system as a whole, undirected edges are considered bidirectional and cross-model connections need to be formed. In section 3.3, the formation of cross-model connections is defined syntactically. In the following, the semantics of such connections are briefly discussed.

A cross-model connection describes an edge between two different models resulting in the possibility to connect structural with structural, behavioral with behavioral, behavioral with structural or structural with behavioral models. Since the approach is designed to extract the behavior of a whole system, connections from structural models to structural models are ignored as we assume that there is always a structural model that covers the structure of a system in its entirety. Connections from behavioral models to behavioral models are relatively difficult to derive directly since they usually describe the behavior of a specific system component or a directed information flow between components. If a behavior model describes the behavior of a component like state machine diagrams, then it can be linked to a system component but usually does not

include information identifying that component. In this case, an expert for this system is needed to link a behavioral model to a node of a structural model. If a behavioral model, in turn, characterizes the communication between components like sequence diagrams, there are nodes that can be directly associated with components of the system and thus with other nodes of structural models. In this case, the connections can be formed automatically. However, the formation of cross-model connections need to be further refined as our goal is to establish cross model connections between nodes of behavioral models. For this purpose, nodes of behavioral models are identified as valid candidates for cross-model connections between such models. They can be marked as either input or output nodes to implicitly define the direction of connections. An input node requires stimuli from other system components enabling the formation of incoming cross-model connections. An output node provides stimuli to other system components enabling the formation of outgoing cross-model connections. Based on this information, cross-model connections between structural and behavioral models are formed.

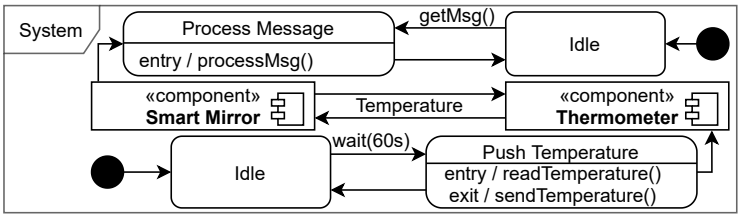

Fig. 3. Consistent model basis with cross-model connections defined

Figure 3 shows the behavioral and structural models representing the architecture of the case study in their merged form. The expert associated the behavioral models shown in figure 2 with their component in the structural model shown in figure 1 and marked the state *Push Temperature* as an output node and the state *Process Message* as an input node. Consequently, the edges *Push Temperature* to *Thermometer* and *Smart Mirror* to *Process Message* were formed.

The cross-model connections involving nodes of structural models are only used to derive the behavior of the system as a whole. In more detail, they are used to help the formation of direct connections between behavioral models to prevent unnecessary steps as SMEs do not infer behavior. To enable the formation of such connections, the modeler needs to add semantic information for every cross-model connection that link nodes of behavioral models to nodes of structural models. Based on these information, direct connections between behavioral models are derived. A cross-model connection between behavioral models is valid if a mapping exists that maps the properties of a node of a behavioral model to properties of a node of another behavioral model. In case

of the case study, the function `sendTemperature()` is mapped on the function `processMsg()`.

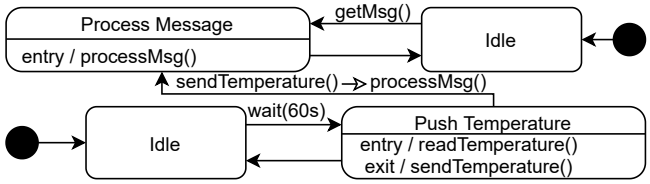

Fig. 4. Cross-model connection between behavioral model elements

Figure 4 shows the connected behavioral models of the case study. Note that all nodes of the structural models, their edges and cross-model edges between the structural and the behavioral models are not shown in the figure. They remain in the consistent model basis. Since there is a path from the *Push Temperature* state to the *Process Message* state in the consistent model base, the edge from the former to the latter is formed, directly connecting the two behavioral models and thus bridging the gap.

This two-phase decision approach has several advantages over a single-phase decision approach in which the input and output nodes of behavioral models are directly linked. By splitting the decision phase, the number of possible direct links between behavioral models is reduced as the inclusion of structural models significantly limits the possibilities. Furthermore, if the expert intends to directly connect nodes of different behavioral models, he needs to exactly know how the system behaves. In turn, if the modeler intends to associate a node of a behavioral model with a node of a structural model, then all he needs to know is whether the behavioral model details the node of the structural model as the node represents a system component. Consequently, mapping a node of a behavioral model to a node of a structural model is easier to accomplish than mapping a node of a behavioral model to a node of another behavioral model because less knowledge is required.

5 Related Work

The topic of systems modeling has been discussed for quite some time. There are several modeling languages and methods to capture the behavior of a system. One approach is SysML. SysML is a modeling language that reuses a subset of UML 2.5 [3]. It is particularly designed to specify requirements, structure, behavior, allocations and constraints on system properties. Cross-model connections are established by a matrix whose format is not prescribed. It can be used to loosely connect model elements of any SysML model. Our approach follows a similar technique but we aim to establish connections between nodes of differ-

ent behavioral models and additionally connect their properties to validate the connection.

ArchiMate Enterprise Architecture is another modeling language [2]. It is designed to visualize and describe different architecture domains and their dependencies and relations. As it is not intended to model the behavior of a system component in detail, a complete view of the system's behavior is not derivable. However, an ArchiMate model is organized in layers. Connections between layers can be compared to cross-model connections. Cross-layer connections follow a specific rule set and therefore are syntactically restricted. We take a more general approach where cross-model connections between all nodes of different models are allowed but they are only valid if the properties of the nodes are mapped.

6 Conclusion and Future Work

The system architecture is often captured by models. In general, models can be categorized as either structural or behavioral. A behavioral model describes either the behavior of a system component or the communication between components. As decentralization proceeds, the behavior of a system is distributed away from one component to many components working together in a system. As a result, the overall behavior of the system is divided among an increasing number of models. As these models are not directly connected, they open a gap making it harder to derive the behavior of the system as a whole.

In this paper, we presented a novel approach to bridge the gap that has resulted from the last evolutionary step of systems. To simplify the idea behind the approach, the structural and behavioral models are first transformed to mixed graphs. These graphs are then merged and transformed into a directed graph to form a consistent model basis that enables the formation of cross-model connections. The behavioral models as part of the consistent model basis are connected by a semi-automatic two-phase decision approach. First, the behavioral models are connected with the structural models of the consistent model basis by an expert. Second, if the derivation of a direct connection results in more than one edge, the expert decides which edge is valid. In contrast to direct linking of behavioral models, this approach requires less system knowledge and significantly reduces cross-model linking possibilities.

In future, we want to further refine the approach and conduct various studies to prove our assumption that this approach is in fact a valuable way of combining models to derive the behavior of a system as a whole. Therefore, we want to apply this approach to a more complex case study and proof that the two-phase decision approach is in fact superior to the one-phase decision approach. We also see the possibility of using this approach in deriving models that describe communication between system components.

Acknowledgment Electronic Component and Systems for European Leadership (ECSEL) supported the development of this approach within the project CPS4EU (Grant Agreement Number 826276).

References

1. Y.2060 : Overview of the Internet of things, https://www.itu.int/rec/T-REC-Y.2060-201206-I/en
2. Archimate 3.1 specification. Standard, The Open Group (Nov 2019), https://pubs.opengroup.org/architecture/archimate3-doc/
3. Omg systems modeling language (omg sysml) version 1.6. Standard, Object Management Group (OMG) (Nov 2019), https://sysml.org/.res/docs/specs/OMGSysML-v1.6-19-11-01.pdf
4. Backlund, A.: The definition of system. Kybernetes (2000)
5. Badreddin, O., Khandoker, R., Forward, A., Masmali, O., Lethbridge, T.C.: A decade of software design and modeling: A survey to uncover trends of the practice. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. pp. 245–255 (2018)
6. Bang-Jensen, J., Gutin, G.Z.: Digraphs: theory, algorithms and applications. Springer Science & Business Media (2008)
7. redha BOUAKOUK, M., ABDELLI, A., MOKDAD, L.: Survey on the cloud-iot paradigms: Taxonomy and architectures. In: 2020 IEEE Symposium on Computers and Communications (ISCC). pp. 1–6. IEEE (2020)
8. Chaâri, R., Ellouze, F., Koubâa, A., Qureshi, B., Pereira, N., Youssef, H., Tovar, E.: Cyber-physical systems clouds: A survey. Computer Networks **108**, 260–278 (2016)
9. Cheraghlou, M.N., Khadem-Zadeh, A., Haghparast, M.: A survey of fault tolerance architecture in cloud computing. Journal of Network and Computer Applications **61**, 81–92 (2016)
10. Cook, S., Bock, C., Rivett, P., Rutt, T., Seidewitz, E., Selic, B., Tolbert, D.: Unified modeling language (UML) version 2.5.1. Standard, Object Management Group (OMG) (Dec 2017), https://www.omg.org/spec/UML/2.5.1
11. Dai, Y.S., Yang, B., Dongarra, J., Zhang, G.: Cloud service reliability: Modeling and analysis. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing. pp. 1–17. Citeseer (2009)
12. Dillon, T., Wu, C., Chang, E.: Cloud computing: issues and challenges. In: 2010 24th IEEE international conference on advanced information networking and applications. pp. 27–33. Ieee (2010)
13. Lin, G., Fu, D., Zhu, J., Dasmalchi, G.: Cloud computing: It as a service. IT Professional Magazine **11**(2), 10 (2009)
14. Mahmoud, R., Yousuf, T., Aloul, F., Zualkernan, I.: Internet of things (iot) security: Current status, challenges and prospective measures. In: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST). pp. 336–341. IEEE (2015)
15. Mell, P., Grance, T., et al.: The nist definition of cloud computing (2011)
16. Mishra, S.K., Sahoo, B., Parida, P.P.: Load balancing in cloud computing: A big picture. Journal of King Saud University-Computer and Information Sciences **32**(2), 149–158 (2020)
17. Pivoto, D.G., de Almeida, L.F., Righi, R.d.R., Rodrigues, J.J., Lugli, A.B., Alberti, A.M.: Cyber-physical systems architectures for industrial internet of things applications in industry 4.0: A literature review. JOURNAL OF MANUFACTURING SYSTEMS **58**, 176–192 (2021)
18. Zhang, M., Selic, B., Ali, S., Yue, T., Okariz, O., Norgren, R.: Understanding uncertainty in cyber-physical systems: a conceptual model. In: European conference on modelling foundations and applications. pp. 247–264. Springer (2016)