2024 4th International Conference on Computer Communication and Information Systems (CCCIS) | 979-8-3503-8954-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCCIS63483.2024.00013

A Model Transformation Tool for Distributed Embedded Control System Development

Taiga Nishimura*
Tokyo City University
Tokyo, Japan
toramegane1104@gmail.com
*Presently with INFOCOM CORPORATION

Takanori Yokoyama**
Tokyo City University
Tokyo, Japan
tyoko@tcu.ac.jp
**Corresponding author

Myungryun Yoo
Tokyo City University
Tokyo, Japan
myoo@tcu.ac.jp

***Abstract*—The paper presents a model transformation tool designed for the development of distributed embedded control systems. While MATLAB/Simulink is commonly used for designing controller models, its models alone are insufficient for the comprehensive development of distributed embedded control systems. To overcome this limitation, we introduce UML deployment diagrams to represent task allocation to nodes and subsequently develop a model transformation tool. This tool takes a Simulink model and a deployment diagram as input, generating a UML model of a distributed control system. The UML model comprises a class diagram, object diagrams, sequence diagrams, and activity diagrams. The UML model is based on the time-triggered distributed object model, which is well-suited for distributed embedded control systems. The application of this model transformation tool facilitates the efficient development of distributed embedded control systems.**

***Index Terms*—Embedded Software, Model-Based Design, Software Tools, Distributed Control Systems**

I. Introduction

Model-based design is widely adopted in embedded control system development and model-based design tools such MATLAB/Simulink [1] are used for control logic design. However, model-based design tools have limitations, as highlighted by Sangiovanni-Vincentelli and Di Natale [2]. These include a lack of separation between functional and architecture models, insufficient support for defining task and resource models, absence of modeling for analyzing scheduling-related delays, and inadequate semantics preservation.

To address these shortcomings, it is recommended to use model-based design tools solely for control logic design and turn to software modeling languages like UML for software design. UML is suitable for architectural design and task structure design, encompassing nonfunctional design considerations such as real-time properties and data preservation. UML offers a range of diagrams that prove beneficial for both functional and nonfunctional design aspects.

Several tools, such as those by Ramos-Hernandez et al. [3][4], Müller-Glaser et al. [5][6], and Sjöstedt et al. [7], take Simulink models as input and generate UML models. However, there might be problems with the reuse of classes created by these tools, as each class usually corresponds to an element in the original Simulink models. To improve reusability, it is crucial to structure UML models using object-oriented concepts.

Our developed model transformation tool [8][9][10][11] addresses these concerns, generating reusable UML models, including class diagrams, object diagrams, and sequence diagrams. We established rules for transformation based on the time-triggered object-oriented software design method [12][13]. Additionally, a code composition tool integrates code generated from UML and Simulink models [14]. To improve data consistency verification in a preemptive multi-task environment, we developed a data consistency verification tool [15]. Nevertheless, manual addition of task elements to generated UML models is necessary for tool utilization. The existing model transformation tool also does not generate UML models representing distributed control systems. Hence, there is a need for a model transformation tool for distributed control systems, which automatically generate UML models containing information on tasks and their allocation to nodes.

This paper introduces a model transformation tool that can generate such UML models. The target distributed control systems are based on a time-triggered distributed object model [16], which is an extension to the time-triggered object-oriented software. We introduce deployment diagrams of UML to represent the task allocation to nodes because Simulink models does not contain the information on nodes and tasks. Then, we present a model transformation tool that takes not only Simulink models but also the deployment diagrams as input and then generates UML models containing nodes and tasks, which are modeled based on the time-triggered distributed object model.

The subsequent sections of the paper are structured as follows. Section II discusses the positioning of the model transformation tool in the software design process and the distributed computing model. Section III elaborates on the models serving as inputs to the transformation tool, while Section IV delineates the models generated as outputs. Section V describes the processing flow within the model transformation tool and application experiments of the tool. Finally, Section VI concludes the paper.

II. Overview

A. Software Design Flow with Model Transformation Tool

Fig. 1 illustrates the distributed control software design flow incorporating the model transformation tool. Initially, a

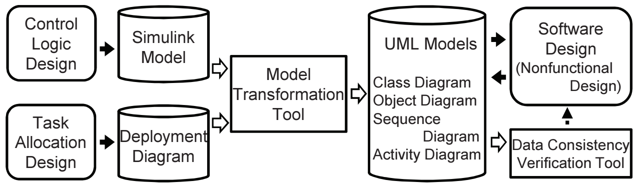


Fig. 1. Distributed Control Software Design Flow

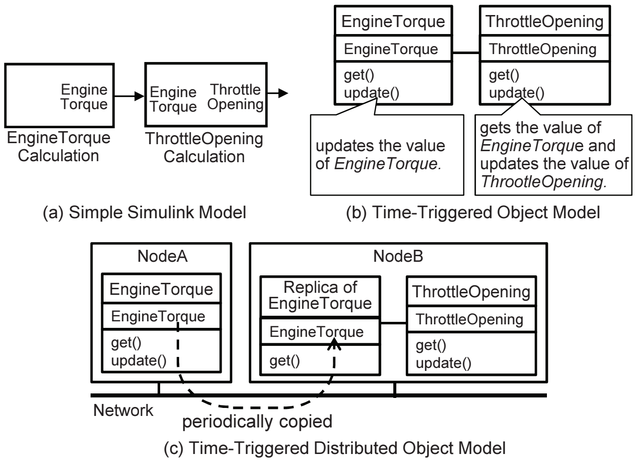


Fig. 2. Time-Triggered Distributed Object Model

Simulink model is developed during the control logic design. Subsequently, a UML deployment diagram is crafted in the task allocation design. Following this, employ the model transformation tool to convert both the control model and the deployment diagram into UML models, which encompass class diagrams, object diagrams, sequence diagrams and activity diagrams. This step corresponds to the functional design of the software. The next step involves verifying the generated UML models using the data consistency verification tool. Any necessary modifications to the UML models are made during the software design process. This step corresponds to the nonfunctional design of the software.

B. Distributed Computing Model

The model transformation tool generates UML models of distributed control systems based on the time-triggered distributed object model [16], which is an extension to the time-triggered object model [12][13].

Fig. 2 (a) shows a simple Simulink model, which consists of two subsystems: *EngineTorqueCalculation* and *ThrottleOpeningCalculation*. The value of *ThrottleOpening* is calculated using the value of *EngineTorque*.

In the time-triggered object model, the classes that calculate and store the values called value objects. The class of a value object has operation *update* and operation *get*. The former operation gets the value from other objects by calling their *get* operations if needed and store the calculated value into its attribute.

Fig. 2 (b) depicts the class diagram aligned with the time-triggered object model of the Simulink model illustrated in Fig. 2 (a). The class *EngineTorque* has the data *EngineTorque* and its operation *update* performs *EngineTorqueCalculation* of the Simulink model. The class *ThrottleOpening* has the data *ThrottleOpening* and its operation *update* performs *ThrottleOpeningCalculation* of the Simulink model.

The time-triggered distributed object model provides location transparent distributed computing using replica objects. If an object on one node refers to another object on different node, the replica of the latter object is allocated on the former node. The former object gets the value of the replica object, not the original one. The value of the replica object is maintained to be equal to the value of the original object by the distributed computing environment. The model is well-suited for distributed embedded control systems because it achieves location transparency efficiently through one-way inter-node communication, without relying on bidirectional communication typically employed in remote procedure calls.

Fig. 2 (c) shows the time-triggered distributed object model in the case that *EngineTorque* and *ThrottleOpening* are allocated to *NodeA* and *NodeB* respectively. The replica of *EngineTorque* is also allocated to *NodeB*. The value of the replica of *EngineTorque* is maintained by periodically copying the value of the original *EngineTorque* object to the replica object. *ThrottleOpening* object gets the value of *EngineTorque* by calling the *update* operation of the replica of *EngineTorque*.

Our existing model transformation tool generates only controller classes and value object classes. We extend the tool to generate classes that represent nodes, processors, tasks and replica objects for distributed computing based on the time-triggered distributed object model.

III. Input Models

A. Simulink Model

Simulink models are commonly structured in layers. MAAB (MathWorks Automotive Advisory Board, presently called MAB (MathWorks Advisory Board)) has presented control algorithm modeling guidelines using MATLAB, Simulink, and Stateflow. The guideline version 3 suggests that a Simulink model should be layered by the top layer, the trigger layer, the structure layer and data flow layer [17]. The guideline version 5 suggests that the model consists of the top layer and bottom layer [18]. The top layer can be divided into the function layer and the schedule layer and the bottom layer can be divided into the sub function layer, control flow layer, selection layer and data flow layer.

Our existing model transformation tool dealt with the structure layer of the guideline version 3 as input, which corresponds to the control flow layer and selection layer of the guideline version 5. However, the structure layer has no information on tasks. Thus, we extend the tool to deal with the trigger layer or the schedule layer that contains the information on tasks.

Fig. 3 shows an example Simulink model with the trigger layer (schedule layer). Triggered subsystems or function-

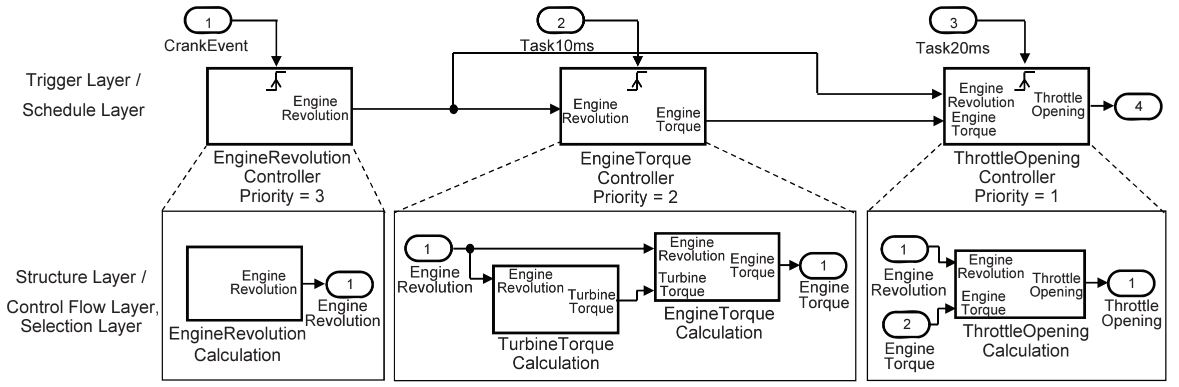
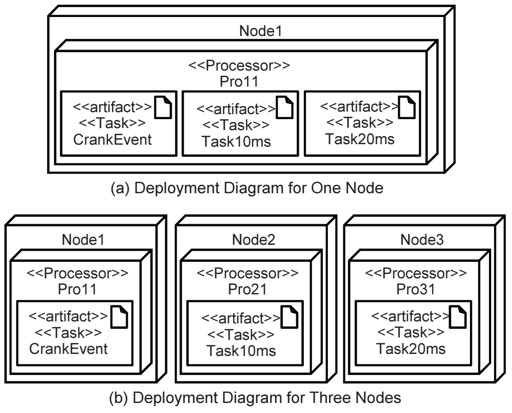


Fig. 3. Example Simulink Model



Node1
<<Processor>>
Pro11
<<artifact>> <<Task>> CrankEvent
<<artifact>> <<Task>> Task10ms
<<artifact>> <<Task>> Task20ms
(a) Deployment Diagram for One Node
Node1
<<Processor>>
Pro11
<<artifact>> <<Task>> CrankEvent
Node2
<<Processor>>
Pro21
<<artifact>> <<Task>> Task10ms
Node3
<<Processor>>
Pro31
<<artifact>> <<Task>> Task20ms
(b) Deployment Diagram for Three Nodes

Fig. 4. Example Deployment Diagram

call subsystems are used in the trigger layer. In Fig. 3, three triggered subsystems, *EngineRevolutionController*, *EngineTorqueController* and *ThrottleOpeningController*, with the trigger input ports, *CrankEvent* (triggered by the crank angle sensor event), *Task10ms* (periodically triggered each 10ms) and *Task20ms* (periodically triggered each 20ms). The execution priorities of the triggered subsystems are also defined. A trigger input port means the activation of a task and the priority of a triggered subsystem means the priority of the task.

B. Deployment Diagram

We use a deployment diagram of UML to describe the allocation of tasks to nodes of a distributed control system. Fig. 4 shows example deployment diagrams for the tasks for the Simulink model shown by Fig. 3. Fig. 4 (a) shows the diagram for the case that all tasks are allocated to one node and Fig. 4 (b) shows the diagram for the case that the task *CrankEvent*, the task *Task10ms* and the task *Task20ms* are allocated to *Node1*, *Node2* and *Node3* respectively. Tasks are represented as artifacts and allocated to processors on nodes. The current version of the model transformation tool deals with uniprocessor node.

IV. Output Models

A. Class Diagram

The design pattern of the time-triggered object model comprises the abstract class *Controller*, representing a single-function controller model, and the abstract class *ValueObject*, representing data within the controller model. Their subclasses denote specific single-function controllers and concrete data. The model transformation tool generates these classes based on the input Simulink model. If replica objects are needed to realize the task allocation represented by the input deployment diagram, their classes are generated. The tool also generated the classes *Node*, *Processor*, *Task* and *PeriodicTask* that represent nodes, processors, general tasks and periodic tasks respectively. The classes and objects of processors and tasks must be included in a generated UML model to verify the data consistency [15].

Fig. 5 depicts the class diagram generated from the Simulink model illustrated in Fig. 3. The classes *EngineRevolution*, *TurbineToque*, *EngineTorque* and *ThrottleOpening* that correspond to the data in the structure layer of the Simulink model are generated as subclasses of the class *ValueObject*. Those value object classes have the *update* operations to calculate their values and the *get* operations to read their values. The classes represented by dashed rectangles are the classes of replica objects and are generated only if needed. For example, replica classes are not generated for the deployment diagram shown by Fig. 4 (a). The replica class *EngineRevolution_in* (the replica of *EngineRevolution*) and the replica class *EngineTorque_in* (the replica of *EngineTorque*) are generated for the deployment diagram shown by Fig. 4 (b).

The classes *EngineRevolutionController*, *EngineTorqueController* and *ThrottleOpeningController* that correspond to the triggered subsystem in the trigger layer of the Simulink model are also generated as the subclasses of the class *Controller*. Those controller classes have the *exec* operations executed by tasks. The *exec* operation of a controller class calls the *update* operations of the value objects within the controller.

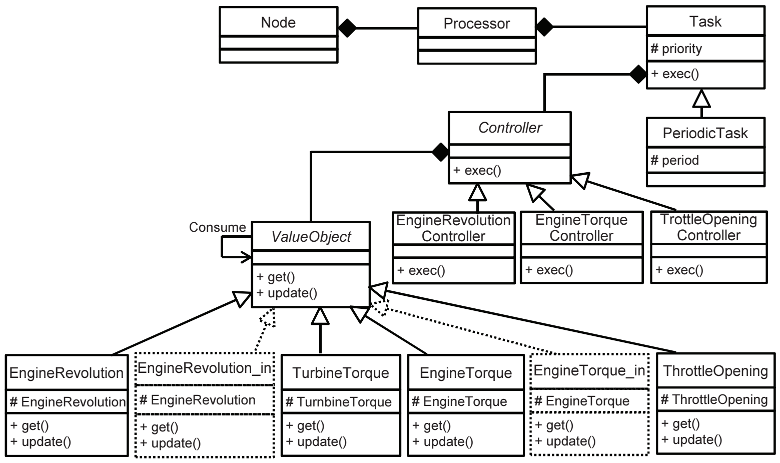


Fig. 5. Example Class Diagram

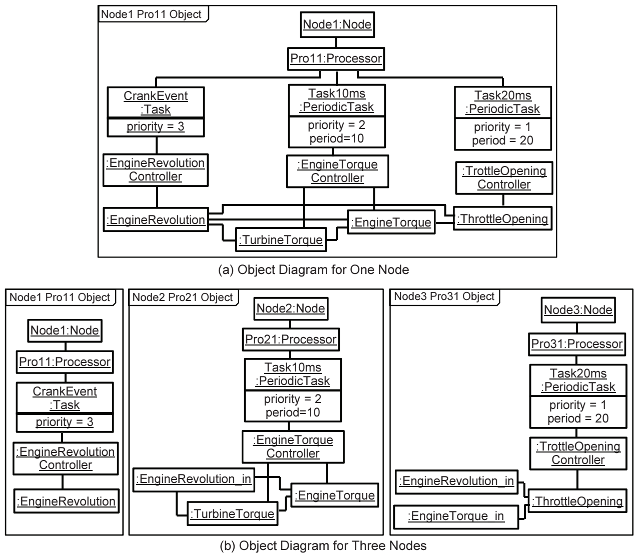


Fig. 6. Example Object Diagram

B. Object Diagram

Object diagrams are generated depending on the task allocation. Fig. 6 (a) shows the object diagram corresponding to the deployment diagram shown by Fig. 4 (a). All task objects, all controller objects and all value objects belong to the processor *Pro11* of the node *Node1*. There are no replica objects in this case.

An object diagram is generated for each node. Fig. 6 (b) shows the object diagrams corresponding to the deployment diagram shown by Fig. 4 (b). In this case, three object diagrams corresponding to nodes are generated. The task object *CrankEvent*, the controller object *EngineController* and the value objet *EngineRevolution* belong to the processor *Pro11* of the node *Node1*. The periodic task object *Taks10ms*, the controller object *EngineTorqueController*, the value objects *TurbineTorque* and *EngineTorque* and the replica object *EngineRevolution_in* belong to the processor *Pro21* of the node *Node2*. The periodic task object *Taks20ms*, the controller object *ThrottleOpeningController*, the value objet *ThrotleOpening* and the replica objects *EngineRevolution_in* and *EngineTorque_in* belong to the processor *Pro31* of the node *Node3*.

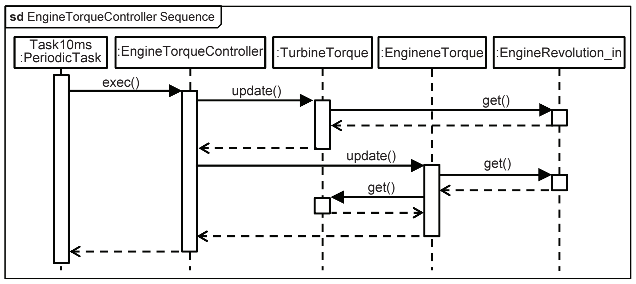


Fig. 7. Example Sequence Diagram

C. Sequence Diagram

A Sequence diagram is generated for each task. Fig. 7 shows the sequence diagram for the task *Task10ms* on the node *Node2*. The periodic task *Task10ms* calls the *exec* operation of the object of *EngineTorqueController*, which calls the *update* operation of *TurbineTorque* and the *update* operation of *EngineTorque*. The *update* operations call the *get* operation of the replica object *EngineRevolution_in*.

D. Activity Diagram

An activity diagram is generated for each *exec* operation of a task class or a controller class. Fig. 8 shows the activity diagrams of the *exec* operation of *Task10ms* and the *exec* operation of *EngineTorqueController*. The former represents that the *exec* operation of *Task10ms* calls the *exec* operation of *EngineTorqueController*. The latter represents that the *exec* operation of *EngineTorqueController* calls the *update* operation

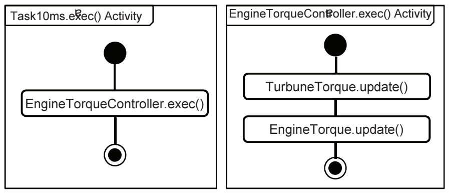

Task10ms.exec() Activity
EngineTorqueController.exec()
EngineTorqueController.exec() Activity
TurbuneTorque.update()
EngineTorque.update()

Fig. 8. Example Activity Diagrams

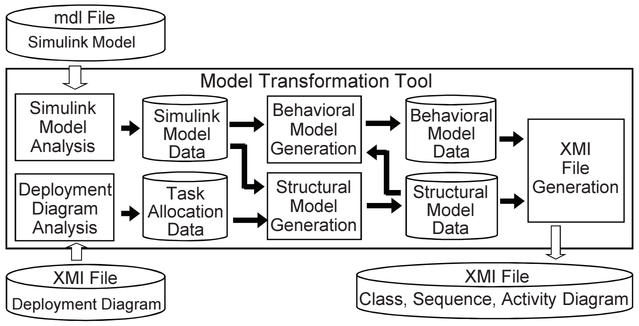


Fig. 9. Model Transformation Tool

of *TurbineTorque* and subsequently calls the *update* operation of *EngineTorque*.

V. Model Transformation Tool and Experiments

A. Model Transformation Tool

The model transformation tool takes as input the mdl file of a Simulink model and the XMI file of a deployment diagram, producing as output an XMI file that consists of a class diagram, object diagrams, sequence diagrams, and activity diagrams. XMI is a standardized file format used in UML [19].

Fig. 9 depicts how the model transformation tool processes data internally. The tool initially analyzes the mdl file, extracting essential Simulink model data. It also analyzes the XMI file of the deployment diagram, extracting task allocation data. Next, the tool generates structural model data by referencing the Simulink model data and the task allocation data. Additionally, it creates behavioral model data by again consulting the Simulink model data and the structural model data. Finally, integrating both the structural and behavioral model data, the tool produces an XMI file that includes a class diagram, object diagrams, sequence diagrams, and activity diagram, as explained in Section IV.

B. Experiments

We utilized the model transformation tool on sample Simulink models supplied by MathWorks, Inc. The sample Simulink models, however, does not contain trigger layer models. Therefore, we divide the sample Simulink models into several blocks and define triggered subsystems in the trigger layer. We also define several deployment diagrams for the layered Simulink models. Subsequently, we utilized the model transformation tool to convert the layered Simulink models and the deployment models into UML models.

Fig. 10 (a) shows the trigger layer of a layered Simulink model, which is built based on the sample Simulink model of a fault-tolerant fuel control system [20]. Fig. 10 (b) shows one of the structure layer Simulink models. Fig. 10 (c) shows one of the deployment diagrams.

Fig. 11 shows the generated UML diagrams for the fault-tolerant fuel control system: a class diagram, some of the object diagrams, one of the sequence diagrams and one of the activity diagrams.

We think the transformation tool is applicable to embedded control system development, as it effectively converts Simulink models and deployment diagrams into UML models. These UML models aptly represent distributed embedded control systems, aligning with the distributed time-triggered object model.

VI. Conclusion

We have developed a model transformation tool that takes a Simulink model with the trigger layer (schedule layer) and a deployment diagram representing task allocation as input, and generates class diagrams, object diagrams, sequence diagrams and activity diagrams of a distributed embedded control system. The application of this model transformation tool facilitates the efficient development of distributed embedded control systems.

We are going to expand the capabilities of the model transformation tool to accommodate multiprocessors, which are increasingly prevalent in embedded control systems. We also have a plan to develop a stub generator for our distributed computing environment [16], which requires stubs that connect the replica objects and the middleware. The stub generator generates the stub code from the UML models generated by the model transformation tool.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP18K11225 and JP21K11815.

References

[1] The MathWorks Inc., “Simulink,” https://www.mathworks.com/products/simulink.html.

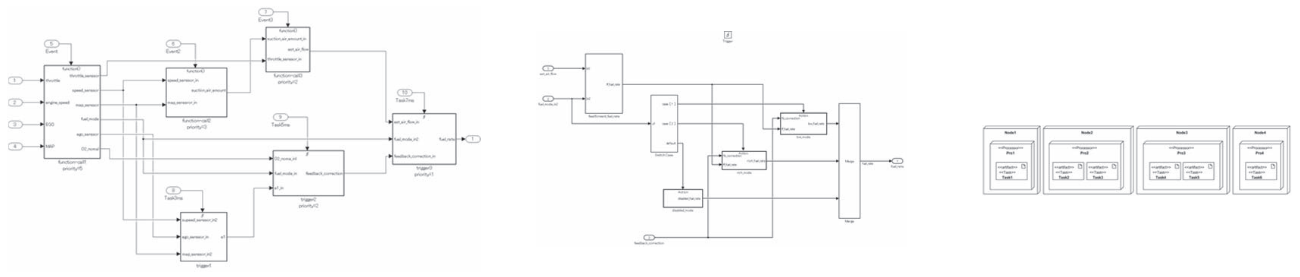
[2] A. Sangiovanni-Vincentelli and M. Di Natale, “Embedded System Design for Automotive Applications,” IEEE Computer vol.40, no.10, pp.42–51, 2007.

[3] D. N. Ramos-Hernandez, P. J. Fleming, S. Bennett, S. Hope, J. M. Bass and M. J. Baxter, “Process Control Systems Integration Using Object Oriented Technology,” Proceeding of Technology of Object-Oriented Languages and Systems (TOOLS 38), pp.148–158, 2001.

[4] D. N. Ramos-Hernandez, P. J. Fleming and J. M. Bass, “A Novel Object-Oriented Environment for Distributed Process Control Systems,” Control Engineering Practice vol.13, no.2, pp.213–230, 2005.

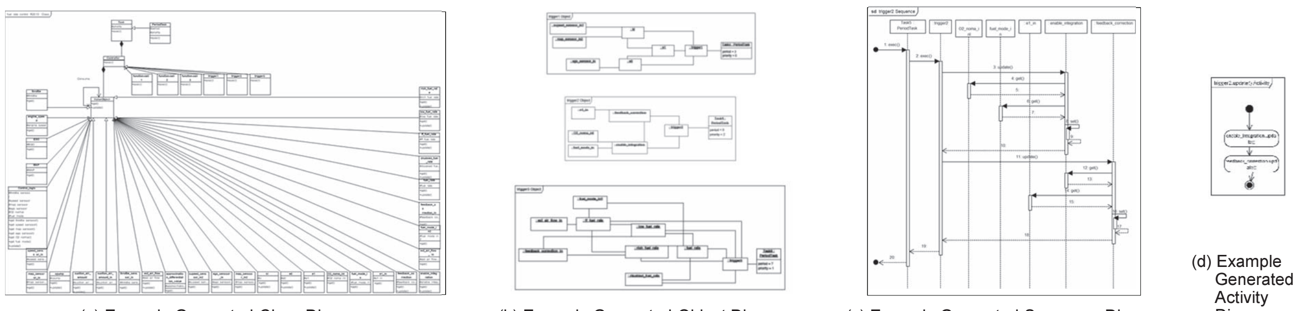
[5] M. Kühl, B. Spitzer and K. D. Müller-Glaser, “Universal Object-Oriented Modeling for Rapid Prototyping of Embedded Electronic Systems,” Proceedings of the 12th IEEE International Workshop on Rapid System Prototyping, pp.149-154, 2001.

[6] K. D. Müller-Glaser, G. Frick, E. Sax and M. Kühl, “Multiparadigm Modeling in Embedded Systems Design,” IEEE Transactions on Control Systems Technology, vol.12, no.2, pp.279–292, 2004.



(a) Example Trigger Layer Simulink Model (a) Example Structure Layer Simulink Model (c) Example Deployment Diagram

Fig. 10. Example Input Model for Experiments



(a) Example Generated Class Diagram (b) Example Generated Object Diagrams (c) Example Generated Sequence Diagram (d) Example Generated Activity Diagram

Fig. 11. Example Generated UML Model

[7] C.-J. Sjöstedt, J. Shi, M. Törngren, D. Servat, D. Chen, V. Ahlsten and H. Lönn, "Mapping Simulink to UML in the Design of Embedded Systems: Investigating Scenarios and Structural and Behavioral Mapping," OMER 4 Post Workshop Proceedings, 2008.

[8] T. Kamiyama, T. Soeda, M. Yoo and T. Yokoyama, "A Simulink to UML Transformation Tool for Embedded Control Software Design," International Journal of Modeling and Optimization, vol.2, no.3, pp.197-201, 2012.

[9] M. Tamura, T. Kamiyama, T. Soeda, M. Yoo and T. Yokoyama, "A Model Transformation Environment for Embedded Control Software Design with Simulink Models and UML Model," Lecture Notes in Engineering and Computer Science: Proceedings of the International MultiConference of Engineers and Computer Scientists 2012. pp.795–800, 2012.

[10] Y. Kuroki, M. Yoo and T. Yokoyama, "A Simulink to UML Model Transformation Tool for Embedded Control Software Development," Proceedings of 2016 IEEE International Conference on Industrial Technology (ICIT), pp.700–706, 2016

[11] Kosuke Tanaka, Shoumu Inaho, Masami Hatano, Yuta Kuroki, Myungryun Yoo, and Takanori Yokoyama, "An Extended Simulink to UML Model Transformation Tool for Embedded Control Software Development," Proceedings of 2017 International Conference on Industrial Design Engineering (ICIDE 2017). pp.76–81, 2017.

[12] T. Yokoyama, H. Naya, F. Narisawa, S. Kuragaki, W. Nagaura, T. Imai and S. Suzuki, "A Development Method of Time-Triggered Object-Oriented Software for Embedded Control Systems," Systems and Computers in Japan, vol.34, no.2, pp.338–349, 2003.

[13] K. Yoshimura, T. Miyazaki, T. Yokoyama, T. Irie and S. Fujimoto, "A Development Method for Object-Oriented Automotive Control Software Embedded with Automatically Generated Program from Controller Models," 2004 SAE World Congress, 2004-01-0709, 2004.

[14] T. Kamiyama, M. Tamura, T. Soeda, M. Yoo and T. Yokoyama, "An Embedded Control Software Development Environment with Simulink Models and UML Models," IAENG International Journal of Computer Science, vol.39, no.3, IJCS_39_3_5, 2012.

[15] T. Ito, M. Yoo and T. Yokoyama, "An Embedded Control Software Development Environment with Data Consistency Verification for Preemptive Multi-Task Systems," International Journal of Advances in Software Engineering & Research Methodology, vol.2, no.2, pp1–5, 2015.

[16] T. Ishigooka and T. Yokoyama, "A Time-Triggered Distributed Object Computing Environment for Embedded Control Systems," Proceedings of 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), pp.191–198, 2007.

[17] MathWorks Automotive Advisory Board (MAAB), "Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow," Version 3.0, 2012.

[18] MathWorks Advisory Board (MAB), "Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow," Version 5.0, 2020.

[19] Object Management Group, "XML Metadata Interchange Specification," Version 2.0.1, 2005.

[20] The MathWorks Inc, "Model a Fault-Tolerant Fuel Control System," https://www.mathworks.com/help/simulink/slref/modeling-a-fault-tolerant-fuel-control-system.html.