Type-Based Modelling and Collaborative Programming for Control-Oriented Systems (Short Paper)

Weidong Ma[1,2](✉) and Zhaohui Luo[2]

[1] Institute of Electronic Engineering,
China Academy of Engineering Physics, Mianyang, China
mawd.sjtu@gmail.com
[2] Department of Computer Science, Royal Holloway,
University of London,
Egham, UK
Zhaohui.Luo@hotmail.co.uk

Abstract. Domain-specific languages (DSL) are more expressive and thus tackle complexity better, making software development easier and more efficient. DSL can automate the production of quality code that based on the proper abstraction of the system. This paper proposes a type-base approach to requirement modelling, called CosRDL, to Implementing a trusted real-time embedded system. A set of rules and formal methods are defined to build CosRDL models for embedded systems, from which the model may be verified apart the specification. CosRDL is described as abstract of event-driven behaviors that support communication between active objects (processes) to support concurrency and collaborative computing. The control processing and properties can be described by CosRDL syntax as an model extension and to make system implementation model. Meanwhile, a case study is presented to figure out how to apply the approach of CosRDL modelling for control systems.

Keywords: Modelling · Domain-specific · Type theory · Embedded system · Collaborative computing

1 Introduction

Embedded systems are used in many fields including mobile phone, automation, aeronautics, and so on. Many embedded systems are timed sensitive, application-specific, tightly constrained and system-in-a-system, so that most of them are safety critical systems. These systems are stimulated by some asynchronous events, and the components in the embedded systems need collaborate with

W. Ma—The work has been supported mainly by a project with foundational funds of China Academy of Engineering Physics (Grant No. 060608-2017).

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved

each other, waiting for the occurrence of external or internal events. For examples, time event is triggered by timer, and a data process task is executed by an arrival event of a data packet from network. After an event was triggered, the components need perform the appropriate actions or operations to manipulate the hardware or generate a new event that triggers other components. It is a classical reactive system that continuously interacts with the physical world.

The design complexity of embedded systems includes high degree of parallelism, sufficient design freedom and constraints, and multiple optimization objectives. Because of its design complexity, the software engineers have to improve productivity by promoting the level of abstraction of embedded systems. Therefore, the model abstracted from the system has more reachable the actual problem domain from the requirements and hides the implementation details. In embedded software development, the domain-specific modelling (DSM) also raises the level of abstraction beyond the programming codes by providing solution scheme and domain concepts. We may use the DSM model to generate the final products. The DSM methodology usually focuses on specific domains so that enables providing better productivity and code quality.

This paper define a modelling languages and development framework for control-oriented embedded systems. The embedded system is designed from the models that is easy understandable, and the specification documentations is conveniently created. According to the defined model, we could implement the designs and correct codes, and finish the debugging, testing, validation and verification on the code level. A control-oriented system requirements description language called CosRDL, is proposed. It is an approach to modelling for control-oriented system in the event-driven framework. The CosRDL-based model has three levels: CosRDL requirements model, CosRDL syntax, and CosRDL system model. It is very proper to develop for control-oriented applications.

2 Related Work

The model-based approaches help abstract away unnecessary details and increase the potential for efficient validation and verification, and easy reuse and evolution. There are several modelling languages such as UML [9], MARTE [2], and SysML [4] etc. Architecture Analysis and Design Language (AADL) is developed for the specification, analysis, automated integration and code generation of critical computer systems. Some researchers have been efforts to establish an effective relationship between AADL and MARTE [7,8]. The paper [5] proposes to extend hybrid MARTE statecharts based on MARTE and the hybrid automata. The formal syntax and semantics of hybrid MARTE statecharts are submited from labeled transition systems (LTS) or live transition systems.

The Spacecraft Requirements Description Language, called SPARDL, which is proposed by Wang, is designed as a requirement modelling language for periodic control systems (PCS) [10–12]. It can specify the features to implement the design, such as periodic driven behaviors, procedure invocations, timed guard, mode transition, and other basic element modes that represents the observable states in periodic control system.

The process algebras CCS and CSP were proposed to model asynchronous processes, describing concurrent, running at indeterminate speed, and can be modelled for embedded system. Hybrid CSP (HCSP) is to extend CSP for describing hybrid systems, which uses differential equations for modelling continuous time-domain environment. The syntax of HCSP can be described as follows [1,3]:

$$\begin{aligned} P \ &::= \text{skip} \mid x := e \mid \text{wait } d \mid ch?x \mid ch!e \mid P;Q \mid B \rightarrow P \mid P \sqcap Q \mid P^* \\ &\quad \mid \talloblong_{i\in I} io_i \rightarrow P_i \mid \langle F(\dot{\mathbf{s}},\mathbf{s}) = 0 \& B\rangle \mid \langle F(\dot{\mathbf{s}},\mathbf{s}) = 0 \& B\rangle \unrhd \talloblong_{i\in I}(io_i \rightarrow Q_i) \\ S \ &::= P \mid S \| S \end{aligned} \tag{1}$$

- x is variables, and $\mathbf{s}$ is vectors, respectively
- B is boolean and e is arithmetic expressions
- d is a non-negative real constant
- ch is the channel name, io_i stands for a communication event, and either $ch_i?x$ or $ch_i!e$
- P, Q, Q_i are sequential process terms, and S stands for an HCSP process term.

3 Modelling of the Control-Oriented Systems

3.1 The Development Based on CosRDL

In generally, the system designers write requirements in natural language using Word, Tex etc. Requirements documents written in natural language are prone to ambiguity and no proper formal syntax structure. The CosRDL-based development of control-oriented systems that submits in the paper, is showed in Fig. 1.

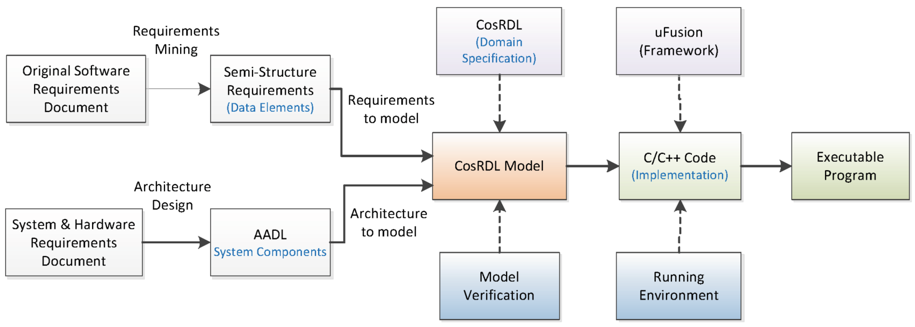

Fig. 1. CosRDL-based development.

The development process is begun from the original software requirements document and system requirements document. The original requirements document can translate into semi-structure requirements by requirement mining. The approaches are using key words of domain-specific fields in control system, so that the semi-structure requirements document keeps the exact semantics of the requirements document. For the system and hardware requirements, it is described by AADL (architecture analysis and design language). For the software requirements, we use CosRDL-based model to describe the control systems.

The CosRDL-based model has three levels: CosRDL requirements model, CosRDL syntax, and CosRDL system model. CosRDL requirements model is designed to define the key elements of the software system that include operations, events and the mapping between events and operations. CosRDL syntax defines a syntax of CCS/CSP style to a model of formal semantics. CosRDL system model is defined for software reuse based on designed components. Once the model has been designed by CosRDL syntax, it can be transformed from CosRDL to C Codes.

3.2 CosRDL Requirements Model

The CosRDL requirement model is defined as follow:

Definition 1. *The CosRDL requirement model is a domain-specific model with 7 objects: an operation set, an event environment set, a type set, an internal action set τ, a mapping function σ, an initial node s, and a set of termination nodes S:*

$$CosRDL_{req} ::= (Oprn, Envr, Type, \tau, \sigma, s, S) \tag{2}$$

- *$Oprn = \{a_1, a_2, ..., a_{N_a}\}$ is a set of operations for a control system.*
- *$Envr = \{e_1, e_2, ..., e_{N_e}\}$ contains communicating events which come from external channels.*
- *Type is a set of types that gives a computing semantic.*
- *τ is a set of internal variables that can not be directly observed in user space.*
- *σ is a mapping function defined as followed:*

$$b = \sigma(a, e, t), \;\; where\ a, b \in Oprn, e \in Envr, t \in R$$

- *$t \in R$ is a restricted variable of timer.*
- *$s \in Oprs$ is an initial node.*
- *$S \subseteq Oprn$ is a set of nodes which are represented termination of operations.*

In the CosRDL requirements model, every operations and events belong to a type. We use operator **typeof** to get the type of an operation or an event. We get the type of a_i and e_i as followed:

$$\text{(event type)}Ty_{e_i} : \text{typeof}(e_i)$$
$$\text{(opration type)}Ty_{a_i} : \text{tpyeof}(a_i)$$

The type is very important because it not only have formal grammars, but also clarified operational semantics. For examples, we can define a type $\mathbb{N}$ based on $\mathbb{Z}$ that combine with predicate subtypes:

$$\textbf{type} \quad \mathbb{N} = \{x : \mathbb{Z} | x \geq 0\} \tag{3}$$

That means if we have a object $p \in \mathbb{N}$, and once we meet $p < 0$ that must occur some error based the type definition of $\mathbb{N}$.

In the control-oriented systems, the signals of input and output that mostly are voltage or current, usually have special boundary, such as a range of voltage [0.00 v, 5.00 v]. Thus we can define a type of signals $\mathbb{VOL}_{\mathbb{A}}$ as

$$\textbf{type} \quad \mathbb{VOL}_{\mathbb{A}} = \{x : \mathbb{R} | x \geq 0 \quad \& \quad x \leq 5.00\} \tag{4}$$

If all operations or input/output signals are defined by basic types and predicates, we have the right type of the model. Every design and programming must based on the types and then any error maybe discovered in time.

The set of events is an external input/output events, and its type has a real semantics meaning, such as the data of position, speed, accelerator and others.

Every node $a \in Oprn$, if a has more than 2 input from other nodes, we must decide the relation of the input actions. Two operator $\otimes$ (or &) and $\oplus$ (or +) was defined as: $\otimes$ means if all input is triggered, then the node is activated; $\oplus$ means if one input is triggered, then the node is activated.

The $CosRDL_{req}$ can be described by a directed acyclic graph (DAG), see Fig. 2. Control systems model with directed acyclic graphs have a wide range of applications. It is closely linked to various control systems that operate in chronological order, such as cars, rail transportation, and aircraft. Usually such systems do not have a loop back to the past scene, the sequence of events and operations are one-way. The operating mechanism of such a system is completely constrained by the external environment and internal timers.

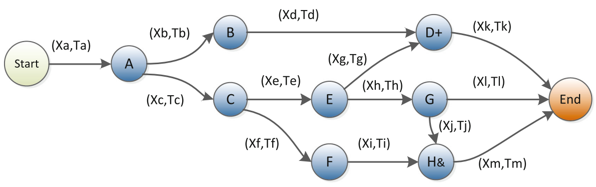

Fig. 2. DAG model of $CosRDL_{req}$.

3.3 CosRDL Syntax

Definition 2. *A formal language for describing hybrid systems, called CosRDL, is defined by the events and its execute operations to modelling the control system. The CosRDL syntax describes as follows:*

$$P ::= SKIP \mid STOP \mid \lambda.P \mid P;Q \mid P \parallel Q \mid P \Box Q \tag{5}$$

$$\lambda ::= (ch?(x : A), t) \mid (ch!(v : A), t) \mid \varepsilon \mid \tau \tag{6}$$

- ***SKIP*** *is an empty statement or represents successful termination.*
- ***STOP*** *is a termination statement, and represents the process that communicates nothing (or deadlock).*

- *λ is an event that executes by input/output actions or empty ($\lambda = \varepsilon$).*
- *P and Q are processes or tasks, and is an abstraction of a group of operations.*
- *$P; Q$ is represented a sequential execution.*
- *A ranges over a type.*
- *ch ranges over a channel name, and x, v ranges over the set of communicable data values.*
- *t is a restricted timer bind the event of x or v.*
- *$P \parallel Q$ behaves as if P and Q run independently.*
- *$P \Box Q$ is a external choice of P or Q.*
- *τ is a set of invisible actions in the internal of the system.*

If $\lambda = (ch?x, t)$, we define the operation $\otimes$ and $\oplus$ for all $x \in Envr$ as follows:

$$(x_1, t_1) \otimes (x_2, t_2) = [x_1 \& x_2), \mathrm{Max}(t_1, t_2)] \tag{7}$$

$$(x_1, t_1) \oplus (x_2, t_2) = [(x_1 | x_2), \mathrm{Min}(t_1, t_2)] \tag{8}$$

Where $x_1 \& x_2 =$ True means that both events x_1 and x_2 happen, and $x_1 | x_2$ that x_1 or x_2 happens.

We use *labelled transition system* to define the semantics. A transition of form

$$P \xrightarrow{(ch?x,t)} Q \tag{9}$$

is taken to express the ability of P to perform the event that inputs x at timer t from channel ch, and thereafter behave like the process Q (execute output v in time t on channel ch). In CSP, there is an *internal* choice syntax $P \sqcap Q$, and we do not use the statement in CosRDL syntax.

3.4 CosRDL: System Model

Definition 3. *The CosRDL system model is defined by a sequence of active objects, a global events set, a data dictionary, programming framework and global clocker:*

$$CosRDL_{system} ::= (ActiveObj, Evts, DataDict, uFrm, t) \tag{10}$$

- *ActiveObj is a set of active objects, which can be run concurrently.*
- *Evts is a set of global events that used by the ActiveObj.*
- *uFrm is the uFusion programming framework.*
- *DataDict is a set of variables that mainly have the global scope in system.*
- *t is a global clocker.*

An active object has its own private thread that executes all of its works, and it has private data and methods that can be invoked by other callers through event-trigger mechanics.

Definition 4. *The active object is a set including three tuples:*

$$ActiveObj ::= (Oprs, EvtQueue, pri) \tag{11}$$

– *Oprs is a set of some operations that process related events.*
– *EvtQueue is an event queue.*
– *pri is a priority used by system scheduler.*

The *ActiveObj* and *uFusion* are running by event-trigger model that introduced by [6]. We can design the event process function by using directly C/C++ codes, or using simulink/stateflow, pseudocodes etc.

4 Case Study

A case study is presented that figure out how to use CosRDL to build an abstract model. A simplified model of an aircraft was built that showed in Fig. 3.

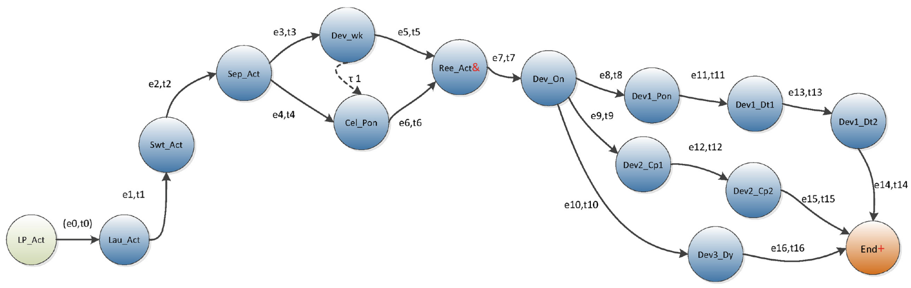

Fig. 3. A simplified CosRDL model of an aircraft electronic system

There are many nodes that represent the specific operations of phases in the system. Each node has to process some actions or operations based on event triggers coming from outer-environment or inner-timer. The Fig. 3 can be modelled in CosRDL as follows:

$$\begin{aligned}
Oprs &= \{LP_{Act}, Lau_{Act}, Swt_{Act}, Sep_{Act}, Dev2_{wk}, Cell_{POn}, Ree_{Act}, Dev_{On},\\
&\quad Dev1_{Pon}, Dev1_{Dt1}, Dev1_{Dt2}, Dev2_{Cp1}, Dev2_{Cp2}, Dev3_{Dy}, End\}\\
Envs &= \{e_0, e_1, e_2, \ldots, e_{16}\}\\
Type &= \{\text{typeof}(e_0), \ldots, \text{typeof}(e_{16}), \text{typeof}(LP_{Act}), \ldots, \text{typeof}(End)\}\\
Ts &= \{t_0, t_1, t_2, \ldots, t_{16}\}\\
s &= LP_{Act}, \quad S = \{End\}\\
&s;\\
&(e_0, t_0).Lau_{Act};\\
&(e_1, t_1).Swt_{Act};\\
&(e_2, t_2).Sep_{Act};\\
&(e_3, t_3).Dev_{wk}[ch!(v : A)] \parallel e_4(t_4).Cel_{Pon}[ch?(x : A)]];
\end{aligned}$$

$$
\begin{aligned}
&[e_5, t_5)\&(e_6, t_6)].Ree_{Act};\\
&(e_7, t_7).Dev_{On};\\
&\{(e_8, t_8).Dev1_{Pon}; (e_{11}, t_{11}).Dev1_{Dt1}; (e_{13}, t_{13}).Dev1_{Dt2};\} \parallel\\
&\quad \{(e_9, t_9).Dev2_{Cp1}; (e_{12}, t_{12}).Dev2_{Cp2};\} \parallel \{(e_{10}, t_{10}).Dev3_{Dy};\}\\
&[(e_{14}, t_{14}) + (e_{15}, t_{15}) + (e_{16}, t_{16})].End.
\end{aligned}
$$

Where Dev_{wk} and Cel_{Pon} communicate in channel. Dev_{wk} has an output action along the channel ch with $ch?(x : A)$and Cel_{Pon} has an action to accept a value on channel ch with $ch!(v : A)$. In the same time, We may refined in some restriction for e_k and t_k and define the types of all values, then execute model checking for the software system. For examples, we define a verified function **checktype** to verify the valid of the event e_k and the operation a_k in every step of the operation, such as:

$$(e_1, t_1).Swt_{Act}[\text{checktype}(e_1, t_1); \text{checktype}(Swt_{Act})];$$

5 Conclusions

The CosRDL model is designed for model-driven development environment. It supports the more reusability and efficient analysis of control systems that based on model-driven development and program framework. Traditionally, embedded programs have been developed in ad hoc time-sensing ways. If the requirements were changed, the CosRDL requirement model would be modified and automatically be translated into the system model, and the system model bind with a program framework. The CosRDL requirement model is built by engineers from the system specification to describe the system behavior through operations, events and functional mapping. The CosRDL system model is translated from CosRDL requirement model to expressed by actived objects, events, data dictionary and program framework, which have more reusability and extensibility. The CosRDL syntax defined a formal language for described the hybrid systems, and the method is more expressive and tackle complexity better, especially for the domain-specific field such as railway control and flight control systems etc.

Acknowledgements. The first author thanks for discussions of related work of the DSL model of embedded systems with prof. Geguang Pu and Weika Miu, and for discussions of formal modelling and hybrid CSP with prof. Naijun Zhan and Shuling Wang.

References

1. Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 511–530. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0020972

2. Espinoza, H., Cancila, D., Selic, B., Gérard, S.: Challenges in combining SysML and MARTE for model-based design of embedded systems. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 98–113. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02674-4_8
3. He, J.: From CSP to hybrid systems. In: A Classical Mind. Embedded system, Programming, Software. Prentice Hall International Ltd., Hertfordshire (1994)
4. Khan, A.M., Mallet, F., Rashid, M.: Combining SysML and Marte/CCSL to model complex electronic systems. In 2016 International Conference on Information Systems Engineering, pp. 12–17. IEEE, Los Angeles, April 2016
5. Liu, J., Liu, Z., He, J., Mallet, F., Ding, Z.: Hybrid marte statecharts. Front. Comput. Sci. **7**(1), 95–108 (2013)
6. Ma, W., Deng, Y., Xu, L., Lin, W., Liu, Z.: COSRDL: an event-driven control-oriented system requirement modeling method. In: Bi, Y., Chen, G., Deng, Q., Wang, Y. (eds.) ESTC 2017. CCIS, vol. 857, pp. 103–117. Springer, Singapore (2018). https://doi.org/10.1007/978-981-13-1026-3_8
7. Mallet, F., André, C., DeAntoni, J.: Executing AADL models with UML/Marte. In: 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, pp. 371–376. IEEE, Potsdam, June 2009
8. SAE: Architecture analysis and design language (AADL)
9. Samek, M.: Practical UML Statecharts in C/C++, Programming for Embedded System. Embedded system, Programming, Software, 2nd edn. Elsevier Inc., Oxford (2008)
10. Wang, Z., et al.: SPARDL: a requirement modeling language for periodic control system. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010. LNCS, vol. 6415, pp. 594–608. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16558-0_48
11. Wang, Z., et al.: A novel requirement analysis approach for periodic control systems. Front. Comput. Sci. **7**(2), 214–235 (2013)
12. Yang, M., Wang, Z., Pu, G., Qin, S.: A novel requirement analysis approach for periodic control systems. Sci. China Inform. Sci. **55**(12), 2675–2693 (2012)