X LANGUAGE: AN INTEGRATED INTELLIGENT MODELING AND SIMULATION LANGUAGE FOR COMPLEX PRODUCTS

Lin Zhang
Fei Ye
Yuanjun Laili
Kunyu Xie
Pengfei Gu
Xiaohan Wang

Beihang University
Xueyuan Road No.37, Haidian District
Beijing, CHINA
{zhanglin,yefei,lailiyuanjun,
zy1903114,by2003151,by1903042}@buaa.edu.cn

Chun Zhao

Beijing Information Science and Technology University
North Ring No.35, Chaoyang District
Beijing, CHINA
zhao_chun@189.cn

Xuesong Zhang

Jilin University
Qianjin Street No.2699, Chaoyang District
Changchun, Jilin Province, CHINA
xs_zhang@126.com

Minjie Chen

Beijing Huaru Technology Co., Ltd
Dongbeiwangxi Road No.10
Haidian District, Beijing, CHINA
jimi_chen@163.com

ABSTRACT

Modeling and simulation have become an important means of supporting analyses and development of complex products. At present, for the development of full-process and full-system modeling and simulation, system modeling languages (such as SysML) are often required to cooperate with multi-physics modeling languages and simulation platforms (such as Modelica, Simulink), which is difficult to ensure the true unity of the whole system, the consistency between the various layers and the traceability of the modeling and simulation process. In response to this problem, this paper proposes a new integrated intelligent modeling and simulation language—X language, which supports the description of system-layer structure and physical behavior, as well as modeling of complex agent models. Interpreter and engine are developed to enable X language to support the simulation of continuous, discrete event and agent models. Finally, the tank model is taken as a case to verify the modeling and simulation capabilities of the X language.

Keywords: complex products, X language, modeling and simulation, integration, intelligent.

ANNSIM '21, July 19-22, 2021, Fairfax, VA, USA;

1 INTRODUCTION

Complex products refer to a class of products with complex customer requirements, system components, product technology, manufacturing processes, test and maintenance, project management, and complex working environment (Li et al. 2011). It has the characteristics of high design difficulty, high cost of test operation and maintenance, strict quality requirements, and high demand for intelligence. At the same time, it faces major challenges such as one-time success, on-time delivery, cycle, and cost compression. To solve the above problems, it's essential to rely on modeling and simulation.

In recent years, Model-based Systems Engineering (MBSE) has become an important means to support system modeling and development (Ramos, Ferreira, and Barcelo 2012). Taking complex products as an example, MBSE transforms the traditional R&D method based on documents and physical models into a model-driven R&D method. This formal description method makes MBSE reusable, unambiguous, easy to understand, and easy to spread. MBSE uses System Modeling Language (SysML) to model the whole process of the system to realize the model-based unified management and optimization of the whole process of product development (Friedenthal, Moore and Steiner 2008). Since SysML cannot be directly simulated, it is necessary to use other multi-domain modeling and simulation methods to verify the correctness and completeness of the model.

A mainstream approach is to uniformly describe system components in different domains based on a unified modeling language to achieve seamless integration and data exchange of multi-domain models (Zhao et al. 2006). For complex products with integrated mechanical, electrical, hydraulic, and control types, firstly based on system modeling language (such as SysML, IDEF, etc.) for demand modeling and architecture design, and then based on physical modeling language (such as Modelica, etc.) and coordinate integration standard specifications (FMI, HLA, etc.), to achieve the development and integration of physical models, and finally through the mapping and conversion among the system model and the physical model for full system modeling and simulation, to achieve unified management of different stages of product development. However, due to the disconnection between the system modeling language and the physical domain modeling language, the connection needs to be realized through conversion. Therefore, it is difficult to ensure the consistency and traceability of the whole process. Moreover, this method lacks the ability for intelligent product modeling and simulation.

To solve the above problems, this paper proposes a new integrated intelligent modeling and simulation language, which supports the description of system-layer structure and physical behavior and simulation, modeling of complex agent models, and supports continuous, discrete event and hybrid simulation. At the system modeling layer, the framework is divided into five parts of definition, connection, state machine, equation, and action, designed to represent structure and behavior. At the layer of physical modeling and simulation, the continuous model, discrete event model, and agent model are incorporated into the couple models of DEVS (Discrete Event System Specification) (Zeigler and Sarjoughian 2017). The interpreter and engine are developed to realize the simulation of the whole system of complex products.

2 RELATED WORKS

In terms of complex products modeling, typical modeling languages and methods include modeling methods based on DEVS, system modeling methods based on SysML, multidisciplinary unified modeling methods based on Modelica, Bond diagram-based system dynamic structure modeling method, European simulation language (ESL)-based software and hardware coordination modeling method, Dymola language-based system dynamics modeling method, and high-level architecture (HLA)-based distributed simulation system modeling method.

SysML is the standard modeling language for systems engineering. It is particularly effective for requirements analysis, structural design, functional description, and system verification in system engineering applications (INCOSE 2012). However, the native SysML model is static and cannot be directly used to verify the correctness and completeness of the model. In this case, the SysML model should

be converted to a model of a specific domain, such as Modelica models. Generally, for systems with different characteristics, the types of SysML diagrams used are also different. Peak et al. (2007) proposed a method of using SysML parameter diagrams to describe the behavior of continuous systems. Batarseh and McGinnis (2012) elaborated on the description of discrete event systems based on SysML activity diagrams, sequence diagrams, and state machine diagrams. Although the SysML model can be extracted and used by the conversion method, the system engineer must add a large amount of simulation code, especially the code related to the system behavior, to obtain an executable simulation model. This method is cumbersome and not very versatile (Kapos et al. 2014). As a system-level description language, SysML can describe the event relationships between agents very well, and can easily establish conceptual models of agents (Sha, Le and Panchal 2011; Maheshwari, Kenley and Delaurentis 2015). However, the native SysML model is static, and the simulation of the agent model needs to rely on other simulation tools. Moreover, SysML is not originally designed for the agent model.

Modelica was proposed in 1997 based on summarizing and unifying the previous multiple modeling languages. The language has many advantages such as high model reusability, simple and convenient modeling, no symbol processing, etc. At the same time, the system standard library of Modelica also provides basic components and typical system models in many fields, including electrics, fluids, thermodynamics, machinery, etc. (Fritzson 2011), which provides great convenience for model development and simulation of physical systems. Although Modelica can model and simulate continuous, discrete, and hybrid models, it lacks sufficient support for discrete models due to its equation-based characteristics (Nutaro et al. 2012) and has the problems of inconvenience in description and low simulation efficiency. Therefore, Modelica lacks support for modeling large-scale discrete systems (Elmqvist et al. 2012; Beltrame and Cellier 2006). As one of the most commonly used languages in modeling and simulation, Modelica is also employed in agent models. However, due to the insufficient support of Modelica for discrete models, the research of agent modeling based on Modelica is mainly focused on the support of third-party libraries. In addition, some studies focus on using Modelica to describe continuous behaviors or actions in agents, rather than modeling and simulation for agents or the whole agent system (Aertgeerts et al. 2015; Schaub, Hellerer and Bodenmüller 2012).

DEVS is a modular, hierarchical, and formal specification for system modeling and simulation, supporting object-oriented mechanisms. The specification was initially only used to build discrete systems, and subsequent research has enriched and perfected the specification for modeling and simulation of hybrid, i.e. both continuous and discrete, systems. As a kind of discrete model, the agent model can be described in DEVS very well. Zhang (2013) constructed a complex agent perception architecture based on multiple types of atomic models, and the BDI model is used as a component of the entire perception architecture. However, the entire model is too large, and many parts are not common to most agent models, so it appears to be relatively redundant. Akplogan et al. (2010) used the DEVS couple model to build a BDI agent model to solve the problem of agent decision-making in agricultural applications and proved the feasibility of the overall architecture. Müller (2008) used DEVS to build a set of system models from the perspective of multi-agents and modified the original DEVS atomic model to adapt to the characteristics of multi-agents. However, the method in the article is not applicable when facing a single complex agent. Although there are many agent models based on DEVS, the specifications provided by DEVS are relatively simple compared to the agent model, and the process of constructing the agent model also needs to reorganize the DEVS model.

To sum up, the existing modeling languages are mostly aimed at a certain part of modeling and simulation and lack the ability of full-process collaborative design. Although the integrated method of system design and simulation can realize the unified management of different stages of product development, its essence is still achieved through the mapping and conversion between languages. It may be easy to deal with a single domain model, however, it is difficult to support the modeling and simulation of complex systems that contain continuous, discrete event and intelligent properties.

3 HIERARCHY AND BASIC ELEMENTS OF X LANGUAGE

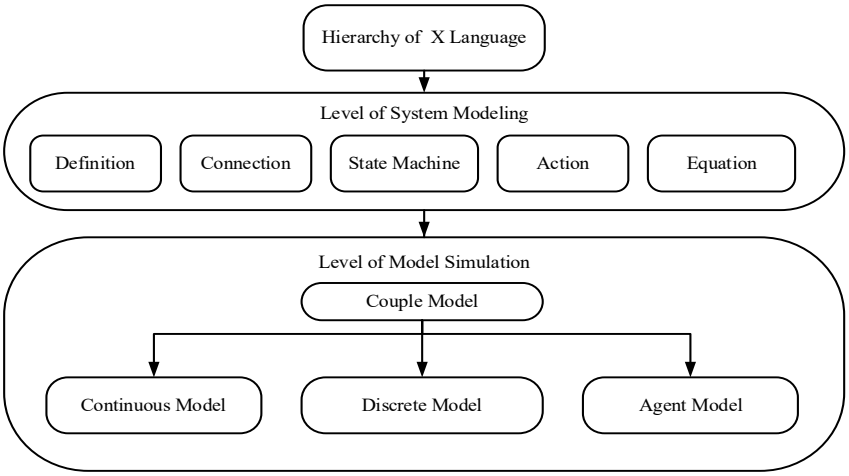

Figure 1: Hierarchy of X language.

The original intention of designing X language is to support the description of system-layer structure and physical behavior and simulation verification, modeling and simulation of various complex agent models, continuous/discrete and hybrid simulation. As shown in Figure 1, at the level of system modeling, five parts are designed to express the structure and behavior of the system. At the model simulation level, the continuous model, discrete model, and agent model are regarded as part of the couple model, which supports the verification of physical behavior.

As shown in Figure 2, X language modeling framework consists of 5 parts. The definition and connection parts define the system model from a global perspective, explaining which components the system contains and the connection relationship among them, mainly used to describe the structure. The equation, action, and state machine are used to describe system behavior and the description form varies according to the characteristics of the model. For example, continuous models can be modeled by equations, and discrete models modeled by state machines.

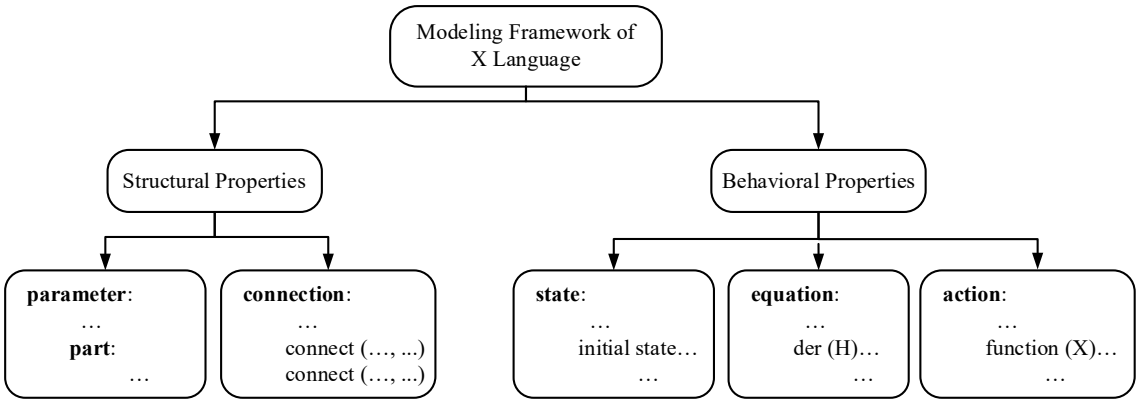

Figure 2: Modeling framework of X language.

X language supports model-based system engineering and can provide the ability to verify the entire process of system design. The model building can be directly interpreted as simulatable DEVS codes via the

interpreter. The X language engine is a multi-domain engine designed based on DEVS, which can support cross-domain modeling in multiple domains including continuous, discrete, and agent models. The simulation result can be directly fed back to the designer to verify the system function.

3.1 Definition

The definition part defines the elements and their relationships from the system structure layer. These elements are the basis of other section and commonly composed of parameter properties, composition properties, value properties, port properties, etc., which are used to define the parameters and their types to be instantiated, the internal structure of the module, state variables, the external module to be called, and the behavior sequence of the agent, respectively. In addition, modules that have been defined somewhere can be imported or inherited in this section.

The following is the grammar of the definition part described in the extended BNF, where {A} means 0 or more As and [A] means an optional A.

```
definition_section ::={(import_clause | extends_clause
   | class_definition
   | parameter_component_clause)';'}
   { port_section
   | part_section
   | value_section
   | plan_definition}
import_clause ::= 'import' (IDENT '=' name | name ('.' ( '*' | '{' import_list '}' ) )? )
extends_clause ::= 'extends' type_specifier [class_modification]
class_definition  ::= ('encapsuate')?  class_prefixes class_specifier
parameter_component_clause ::= 'parameter' type_specifier component_list
port_section ::= 'port:'{port_component_clause}
part_section ::= 'part:'{component_clause';'}
value_section ::= 'value:'{component_clause';'}
import_list ::= IDENT {',' IDENT}
component_clause ::=['replaceable'] type_prefix type_specifier component_list
```

3.2 Connection

The connection part defines the connection relationship among the components in the couple model from the structural level. This connection is realized through ports, which transfer events, energy, and data, as well as services provided and requested by the connection. The extended BNF of the connection part is as follows.

```
connection_section ::= 'connection:'{connect_clause';'}
connect_clause ::= 'connect' '(' component_reference ',' component_reference ')'
```

3.3 State Machine

The state machine part is used to describe the discrete behavior of the model and express the changes of the system states as events occur. In each state, it generally includes the duration of the state, the behavior generated by the internal/external event, and its output. Specifically, in the *when_entry* area, the duration of a state is defined by the function *statehold*. In the *when_receive* area, the model receives external event triggers and generates output. The *when timeover* statement is used to describe the behavior triggered by internal events and the resulting output. The extended BNF of the state machine part is as follows.

```
state_section ::= 'state:' {state_definition}
state_definition ::= 'initial' 'state' IDENT  (state_statement)*  'end';'
```

```
    | 'state' IDENT  (state_statement)*  'end';'
    | 'state' IDENT  catch_clause  ((when_receive_clause';')|(when_statement';'))* 'end";'
state_statement ::= (when_entry_clause
    | when_receive_clause
    | when_goto_out_clause)';'
```

3.4 Equation

In the equation part, systems with continuous behavior are described based on mathematical equations with declarative and non-causal characteristics. Specifically, a set of ordinary differential/partial differential equations is used to describe a model with continuous behavior, among which, equations can be divided into *simple equations*, *if-equations*, *when-equations*, *when-receive* equations, *for-equations*, etc. The *if-equation* is used to define the behavior triggered by constraint conditions. The w*hen-equation* is used to define the behavior triggered by an instantaneous state. The *for-equation* is used to define the behavior that is cyclically executed when a certain condition is met. The extended BNF of the equation part is as follows.

```
equation_section ::= 'equation:' {equation ';'}
equation ::= simple_equation
    | if_equation
    | for_equation
    | when_receive_equation
    | when_equation
```

3.5 Action

The action part mainly describes the behavior of the agent model, as well as intelligent behavior. The agent model is constructed based on the BDI architecture, from which the plan of each agent can be extracted. On this basis, the execution logic of multiple defined plans is sorted to form a plan sequence, which can effectively control the behavior of the agent. The extended BNF of action part is as follows.action_section ::= 'action:' {statement';'}

```
statement ::= send_clause
    | simple_statement
    | function_call
    | break_statement
    | continue_statement
    | return_statement
    | if_statement
       | for_statement
    | while_statement
    | when_statement
    | statehold_clause
    | run_statement
    | agentover_statement
    | transition_clause
```

The above 5 parts are the basic components of model description, when modeling, the structure and behavior of the model are described in the forms of classes, including continuous, discrete, couple, agent, record, function, and connector class. A class is composed of one or more parts, as shown in Figure 3.

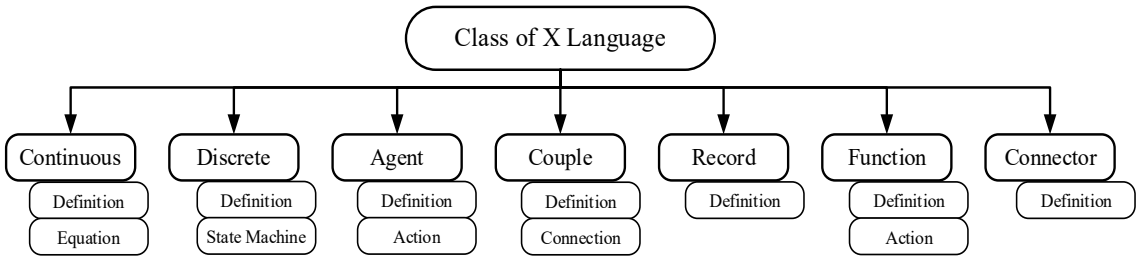

Figure 3: Composition of X language classes.

4 THE INTERPRETER AND ENGINE FOR X LANGUAGE

X language is an object-oriented multi-domain system modeling language. The elements of the language include not only the characteristics of traditional programming language, but also the characteristics of equation-based modeling languages. These two kinds of languages usually adopt different routes in interpretation, which leads to different technical routes in X language interpretation.

The interpretation of X language is divided into two stages, namely the front-end and the back-end. In the front-end, the interpreter will perform lexical analysis and syntax analysis on the source code to obtain an abstract syntax tree, and finally, make a symbol table by traversing and collecting the information of the elements in the model to prepare for the subsequent interpretation. In the back-end part, the interpreter will process the abstract syntax tree that has been processed by the front-end.

When building the engine for X language, it is necessary to consider the simulation capabilities for the domains that need to be supported and the interaction among them. DEVS is a commonly used multi-domain modeling and simulation framework, which can support continuous, discrete event, and agent simulation (Zeigler and Sarjoughian 2017), therefore, it is chosen for the X language engine.

In the X language engine, the communication between the adjacent *simulator* and *coordinator* is based on the message. Each time an event occurs (internal or external), the *coordinator* will send a transformation message to its child nodes to inform them to perform the transformation. When the *simulator* executes its internal or external events, it will calculate its next state. If it performs internal transformation, it also needs to send the output to its parent *coordinator*. There are three types of communications between the *simulator* and the *coordinator*, which are the internal and external event notification from the *coordinator* to the *simulator* and the output from the *simulator* to the *coordinator*.

5 CASE STUDY

The three components of the tank system-the tank, the PI controller, and the source of liquid-are explicit in Figure 4. The tank is connected to the controller and liquid source through connectors. Liquid enters the tank from the source, and leaves the tank at a rate controlled by the valve.The tank has four connectors: *qIn* for input flow, *qOut* for output flow, *tSensor* for providing fluid level measurements, and *tActuator* for setting the position of the valve at the outlet of the tank.

The liquid level *h* in the tank must be maintained at a fixed level as closely as possible. The flow of the liquid source increases sharply at time = 150 to the factor of three of the previous flow level, which creates an interesting control problem that the controller of the tank has to handle.

The X code of this case is demonstrated in Appendix. In order to facilitate readers to understand the modeling method of X language, combined with the content of Chapter 3, a brief description of the case code is given. We simulate the tank system and obtain the response as shown in Figure 5.

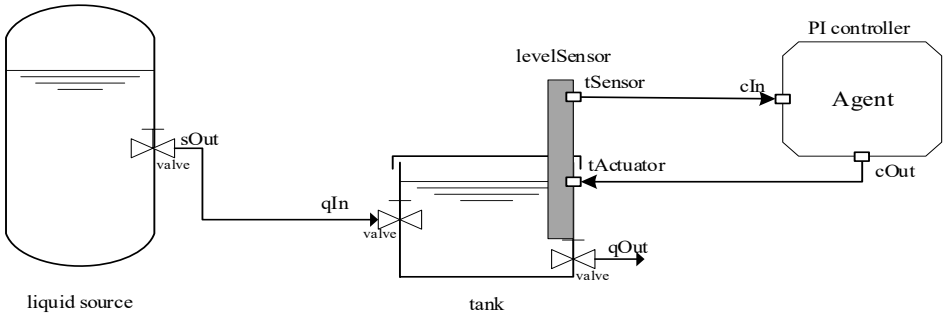

Figure 4: A tank system with a tank, a source for liquid, and a controller.

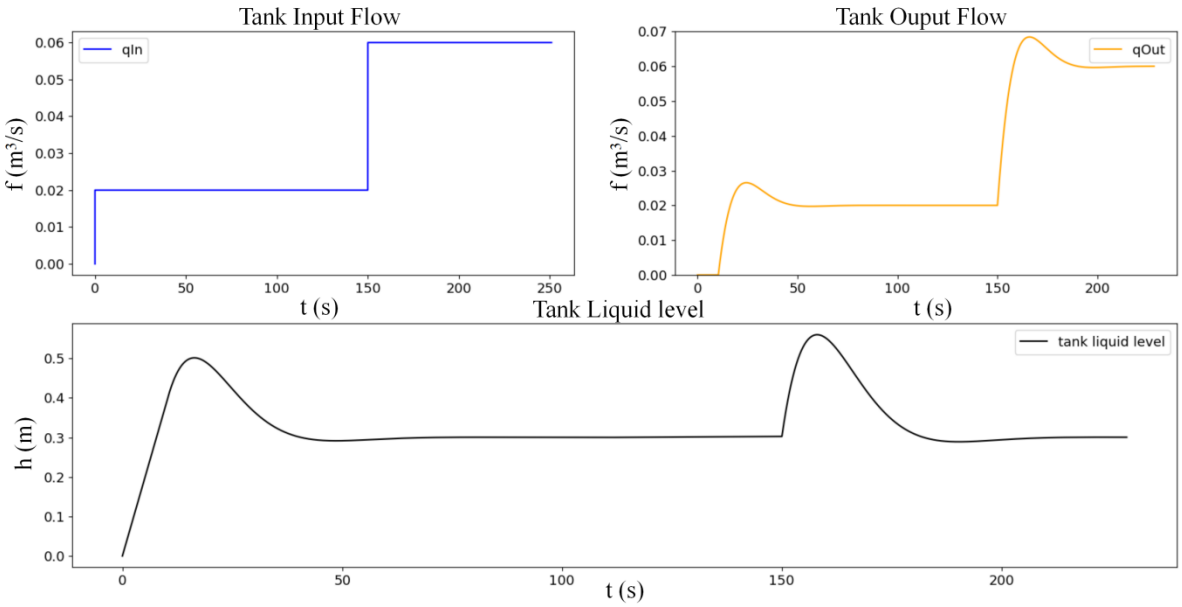

Figure 5: System simulation result.

Split the tank system into 5 models, namely *topmodel*, *LiquidSource*, *TankPI*, *Tank,* and *PIcontinuousController*, where *topmodel* is a couple model, including *LiquidSource* and *TankPI*, and *TankPI* is also a couple model, including *Tank* and *PIcontinuousController*, *Tank* and *PIcontinuousController* are simple models.

The definition part is used in all these 5 models, such as the *part* in the *topmodel* and the *parameter*, *value,* and *port* in the *Tank*. The *topmodel* is a couple model, the connection part needs to be used to define the relationship between different components. For example, the *qOut* of *ls* and the *qIn* of *tpi* are connected to each other to indicate the direction of liquid flow. Since the flow rate of the *LiquidSource* is discrete, it is described by state machine. The *LiquidSource* has three states including *init*, *pass,* and *idle*. Among them, the duration of the state *init* is 0, and the next transition state is *pass*. When a state transition occurs, the current *Flowlevel* is assigned to the output event port *qOut*. The duration of the state *pass* is 150, and the next transition state is *idle.* When a state transition occurs, 3 times the *Flowlevel* is assigned to the output event port *qOut,* then enters the state *idle* and remains the state. The liquid level *h* in the tank changes continuously with time, so its behavior can be described by equations as demonstrated in *Tank*. The controller is an agent, which is used to control the liquid level *h* in the tank to be maintained at a fixed level

as closely as possible. In the *PIcontinuousController,* there is only one task of controlling the liquid level, so only one plan is needed. The controller takes the value of the current level as input, calculating the level at the next moment through the equation, and transmits the result to the *Tank* to complete the entire control process. After the controller model completes a loop, it will wait for the next input information.

6 CONCLUSION

Modeling and simulation have become an important means of supporting analyses and development of a complex product. However, the existing modeling languages lack full-process modeling and simulation capabilities, and it is often necessary to complete integrated modeling through multiple language conversions or tool integration. Therefore, this paper proposes a new integrated intelligent modeling and simulation language - X language, which supports the description of system-level structure and physical behavior, as well as modeling of complex agent models. Interpreter and engine are developed to enable X language to support the simulation of continuous, discrete event and agent models. Finally, the tank system was taken as a case to verify the modeling and simulation capabilities of the X language. Due to space limitations, this paper does not elaborate enough on the X language, and the authors will further introduce it in subsequent papers.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China, No.2018YFB1701600. The authors hereby would also like to thank Dr. Tingyu Lin and Dr. Guoqinag Shi from Beijing Institute of Electronic System Engineering, for participating in the discussions and providing valuable suggestions.

A APPENDICES

Please enter the following link to view the code: https://github.com/ffsuiyue/xcode

REFERENCES

Aertgeerts, A., B. Claessens, R. D. Coninck, and L. Helsen. 2015. “Agent-Based Control of A Neighborhood: A Generic Approach by Coupling Modelica with Python”. *In Proceedings of Building Simulation 2015*, pp. 456-463. Hyderabad, India.

Akplogan, M., G. Quesnel, F. Garcia, A. Joannon, and R. Martin-Clouaire. 2010. “Towards A Deliberative Agent System Based on DEVS Formalism for Application in Agriculture”. In *Proceedings of the 2010 Summer Computer Simulation Conference*, pp. 250-257. Ottawa, Canada.

Beltrame, T., and F. E. Cellier. 2006. “Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism”, In *Proceedings of the 5th International Modelica Conference*, pp. 73–82.

Elmqvist, H., F. Gaucher, S. E. Mattsson, and F. Dupint. 2012. “State Machines in Modelica”. In *Proceedings of the 9th International MODELICA Conference*, pp. 37-46. Munich, Germany。

Friedenthal, S., A. Moore, and R. Steiner. 2008. “OMG Systems Modeling Language (OMG SysML™) Tutorial”. *INCOSE International Symposium*, vol.18, pp. 1731-1862.

Fritzson, P. 2011. “Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica”. Wiley-IEEE Press. New York, USA.

INCOSE 2012. “Systems Engineering Handbook, version 3.2.2 Edition”, *International Council on Systems Engineering*. San Diego, CA, USA.

Kapos, G. D., V. Dalakas, A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos. 2014. “Model-Based System Engineering Using SysML: Deriving Executable Simulation Models with QVT”. In *Proceedings of the 2014 Systems Conference*, pp.531-538. Ottawa, Canada.

Li, T. , B. H. Li, X. D. Chai, and X. F. Yan. 2011. "Meta Modeling Framework for Complex Product Multidiscipline Virtual Prototyping". *Computer Integrated Manufacturing Systems*, 17(6), pp. 1178-1186.

Maheshwari, A., C. R. Kenley, and D. A. Delaurentis. 2015. "Creating Executable Agent-Based Models Using SysML". *INCOSE International Symposium* vol.25, pp.1263-1277.

Müller, J. P. 2008. "Towards A Formal Semantics of Event-based Multi-Agent simulations". *International Workshop on Multi-Agent Systems and Agent-Based Simulation,* pp. 110-126.

Nutaro, J. J., P. T. Kuruganti, V. A. Protopopescu, and M. Shankar. 2012. "The Split System Approach to Managing Time in Simulations of Hybrid Systems Having Continuous and Discrete Event Components". *SIMULATION* vol. 88, pp. 281−298.

Peak, R. S., R. M. Burkhart, S. A. Friedenthal, M. W. Wilson, M. Bajaj, and I. Kim. 2007. "Simulation-Based Design Using SysML part 1: A Parametrics Primer". *INCOSE International Symposium* vol.17, pp.1516-1535.

Ramos, A. L., J. V. Ferreira, and J. Barcelo. 2012. "Model-Based Systems Engineering: An Emerging Approach for Modern Systems". *IEEE Transactions on Systems Man & Cybernetics Part C* vol.42, pp. 101-111.

Schaub, A., M. Hellerer, and T. Bodenmüller. 2012. "Simulation of Artificial Intelligence Agents Using Modelica and the DLR Visualization Library". In *Proceeding of the 9th International Modelica Conference*.

Sha, Z., Q. Le, and J. H. Panchal. 2011. "Using SysML for Conceptual Representation of Agent-Based Models". In *Proceedings of the 2011 Computers & Information in Engineering Conference* vol.54792, pp. 39-50. Washington, DC.

Zeigler, B. P., and Sarjoughian H. S. 2017. "Modeling and Simulation of Systems of Systems". *Guide to Modeling and Simulation of Systems of Systems,* pp. 3-11. Springer, Cham.

Zhang, M. 2013. "Constructing A Cognitive Agent Model Using DEVS Framework for Multi-agent Simulation". *In Proceeding of the 15th Eur. Agent Syst. Summer School*, pp. 1-5.

Zhao, J. J., J. W. Ding, F. L. Zhou, and L. P. Chen. 2006. "Modelica and Its Mechanism of Multi-domain Unified Modeling and Simulation". *Journal of System Simulation* 18(S2), pp. 570-573.

AUTHOR BIOGRAPHIES

LIN ZHANG is a Professor at Beihang University, China, Past President and Fellow of SCS. His research interests include modeling and simulation, cloud manufacturing, and model engineering. His email address is zhanglin@buaa.edu.cn.

FEI YE is a Ph.D. student of Automation Science and Electrical Engineering at Beihang university. He received his master's degree from Beihang University in 2018. His research interests include modeling and simulation, system engineering, and intelligent optimization. His email address is yefei@buaa.edu.cn.

YUANJUN LAILI is an Assistant Professor at Beihang University. She is also a member of SCS and an Associate Editor of "International Journal of Modeling, Simulation, and Scientific Computing". Her main research interests are in the areas of intelligent optimization, modeling and simulation of manufacturing systems. Her email address is lailiyuanjun@buaa.edu.cn.

KUNYU XIE is a Ph.D. student of Automation Science and Electrical Engineering at Beihang University. He received his bachelor's degree from Beihang University in 2019. His research interests include modeling and simulation of continuous, hybrid, and discrete systems. His email address is zy1903114@buaa.edu.cn.

PENGFEI GU is a Ph.D. student of Automation Science and Electrical Engineering at Beihang University. He received his master's degree from university of science and technology Beijing in 2019. His research

interests include modeling and simulation, system engineering and evolutionary game theory. His email address is by2003151@buaa.edu.cn.

XIAOHAN WANG is a Ph.D. student at Beihang University. His research interests include agent-based modeling and simulation, reinforcement learning, and discrete system simulation. His email address is by1903042@buaa.edu.cn.

CHUN ZHAO is an Assistant Professor at Beijing Information Science and Technology University, China. He focuses on the Cloud Manufacturing, Modeling & Simulation of complex system, and FPGA based cloud-edge systems. His email address is zhao_chun@189.cn.

XUESONG ZHANG is an Assistant Professor of Jilin University. His main research interests include parallel computing, parallel simulation, and smart manufacturing system scheduling. His email address is xs_zhang@126.

MINJIE CHEN is a senior engineer of Beijing Huaru Technology Co., Ltd. His research interests include modeling and simulation, reinforcement learning. His email address is jimi_chen@163.com.