information

Article

SES-X: A MBSE Methodology Based on SES/MB and X Language

Kunyu Xie [1], Lin Zhang [1,*], Xin Li [2], Pengfei Gu [1] and Zhen Chen [1]

[1] School of Automation Science and Electrical Engineering, Engineering Research Center of Complex Product Advanced Manufacturing Systems Ministry of Education, Beihang University, No. 37, Xueyuan Road, Beijing 100191, China
[2] Department of Information Systems, College of Business/School of Data Science, City University of Hong Kong, Lau Ming Wai Academic Building, Hong Kong, China
* Correspondence: zhanglin@buaa.edu.cn

Abstract: Model-based systems engineering (MBSE) is a leading paradigm for the analyses and development of complex systems. However, the development of modeling and simulation infrastructure supporting MBSE is lacking, which limits the application of MBSE. To address this problem, this paper proposes an SES-X methodology that integrates system modeling (following SES philosophy) with system simulation (supported by X language) to support the full lifecycle of MBSE modeling, including system analysis, architecture decomposition, physical modeling, and simulation. In the process, SES-X performs two levels of model pruning for model verification and simulation efficiency. This paper also conducts a case study on a car model to illustrate the effectiveness of the SES-X methodology.

Keywords: model-based systems engineering; system entity structure; X language; complex systems

check for updates

Citation: Xie, K.; Zhang, L.; Li, X.; Gu, P.; Chen, Z. SES-X: A MBSE Methodology Based on SES/MB and X Language. *Information* **2023**, *14*, 23. https://doi.org/10.3390/info14010023

Academic Editor: Tuncer Ören

Received: 30 November 2022
Revised: 23 December 2022
Accepted: 26 December 2022
Published: 29 December 2022

1. Introduction

The ever-growing complexity of modern systems creates significant challenges to their design and development. The mechanics, electronics, and even humans would all affect the stability of a system, which needs to be considered in the design stage. Therefore, model-based systems engineering (MBSE) is becoming popular in the design and development of complex systems. As compared with document-centric engineering, MBSE puts models at the center of system design to support the development of complex systems [1]. The MBSE modeling process consists of the following steps: del validation. The purpose is to construct physical models based on logical models following the requirements and to verify the physical models through simulation. Thus, one key problem of MBSE is to combine system architecture (model) design capabilities, with model verification and validation (simulation) capabilities, into a unified process [2].

Being a general approach, there are multiple solutions for MBSE modeling. These solutions generally depend on three pillars: modeling methodology, modeling language, and modeling tools, where the modeling methodology regularizes the design process, and the modeling language and modeling tools support the design process [3]. Previously, many studies have taken a modeling language-centered approach to adopt simulation methods into existing modeling methodologies [4,5]. However, in such solutions, the simulation functionalities are often tailored to fit into the system modeling framework, which limits their simulation abilities [6]. Other studies took a simulation-centered approach by allowing existing simulation software to handle the model established by system modeling language [7,8]. However, this approach requires parameter transformation or model transformation between the two paradigms, which limits its scalability for complicated models.

To address the limitations of these two approaches, the further development of MBSE requests native unified frameworks (including methodology, language, and tools) that can support the entire modeling process of requirement analysis, model design, and model verification. In our previous research, we proposed an X language based on SysML [9], Modelica [10], and DEVS [11], which provides integrated support for logical modeling and physical modeling [12,13]. We also build modeling software XLab [14], which supports the simulation of X language. In this study, we further complement X language and XLab with the system modeling abilities for MBSE development.

Specifically, this paper proposes combining SES/MB, a popular methodology for complex system architecture analysis and modeling, with X language to build an SES-X (SES eXtended) methodology. With this methodology, we can systematically design simulation models aligning with system requirements. SES-X provides a unified framework with top-down system design abilities (following SES philosophy) and bottom-up system simulation abilities (enabled by X language). During the modeling process, our methodology contains two levels of model pruning processes that simplify the model to facilitate verification and simulation. In this paper, we provide an illustrative example to showcase the effectiveness of the SES-X in applications. To the best of our knowledge, it is one of the first MBSE solutions that can provide native support to both modeling and verification functions for the entire process of complex system model development.

The rest of this paper is organized as follows. In Section 2, we review the literature on existing MSBE solutions, and we introduce the SES framework and the X language. Section 3 proposes the SES-X methodology for MSBE development. Section 4 conducts a case study to illustrate the effectiveness of our proposed SES-X methodology. Section 5 concludes the paper.

2. Related Works

2.1. Existing MBSE Modeling Solutions

2.1.1. Modeling Language-Centered Solutions

The majority of MBSE modeling solutions have used SysML as the modeling language [3] and built up the system on the SysML graphs. Examples of such modeling language-centered methodologies include Harmony-SE [15], MagicGrid [16], and OOSEM [17]. For example, Harmony-SE starts with requirement analysis to build functional descriptions and translates functional descriptions into system functions to guide the building of executable models. However, these modeling language-centered solutions have a natural deficiency. Since SysML does not have simulation functionalities, it is difficult for it to validate the (multidisciplinary) models constructed.

To address this issue, a number of studies leverage SysML's profile-extension mechanism to connect with simulation languages and gain simulation abilities, such as SysML4Simulink [18] and SysML4Arena [19]. These profiles have external semantics that allow representations in SysML to simulate specific model artifacts. Although different SysML profiles have been developed, it is very difficult to develop a profile that can fully cover all simulation semantics of a simulation language [1]. As a result, the modeling language-centered solutions often have restricted support for simulations. Furthermore, the extension profiles must be properly updated when interfacing with different versions of simulation tools, which leads to significant maintenance work.

2.1.2. Simulation-Centered Solutions

Another approach tries to integrate MBSE modeling into simulation infrastructures. Zeigler advocated the use of DEVS as a system modeling and simulation framework for MBSE and reviewed the DEVS simulation tools being developed for MBSE [2]. Alshareef incorporated DEVS into MBSE modeling, focusing on the simulation verification process by UML and SysML activity diagram specification [20]. Wach et al., merged the tricotyledon theory of system design [21] with DEVS, which equipped MBSE engineers with modeling and simulation analysis capabilities for multiple system models [22].

There are also some studies using parameter transformation through interfaces, such as FMI [23], or model transformation, using model-to-model (M2M) and model-to-text (M2T) methods [24], which realize the simulation of the model established by SysML. Typical models that establish the model transformation mechanism are Modelica [7], Simulink [7], and DEVS [8]. The limitation of this approach is that the solution will become difficult to maintain when model complexity increases.

While the above two approaches provide a foundation for integrating the simulation into MBSE modeling, there is a lack of native solutions that can simultaneously support system modeling and system simulation. To fill the gap in the literature, we want to develop a methodology for the full lifecycle of MBSE modeling.

2.2. SES/MB Framework

SES/MB framework [25] is a modeling framework based on system entity structure (SES) and model bases (MBs). As illustrated in Figure 1, SES describes the hierarchical composite of a system, which is represented as a tree structure. The tree is built on the basic models in MB as leaf nodes, which link to each other to form complicated systems. The basic models in MB have well-defined input and output interfaces, and they are configurable with parameters. For example, in the SES shown in Figure 1, the blocks represent entity nodes, which represent real-world objects. Furthermore, the tree has three types of connectors, which are suffixed with Dec (Aspect), MultiA (MultiAspect), and Spec (Specialization), respectively. A Dec connector links a node to its decomposition of multiple nodes. A MultiA connector decomposes one node into multiple entities of the same type. A Spec connector describes the specialization and choices of an entity among multiple, according to selection rules, which need to be carried out during the pruning process.

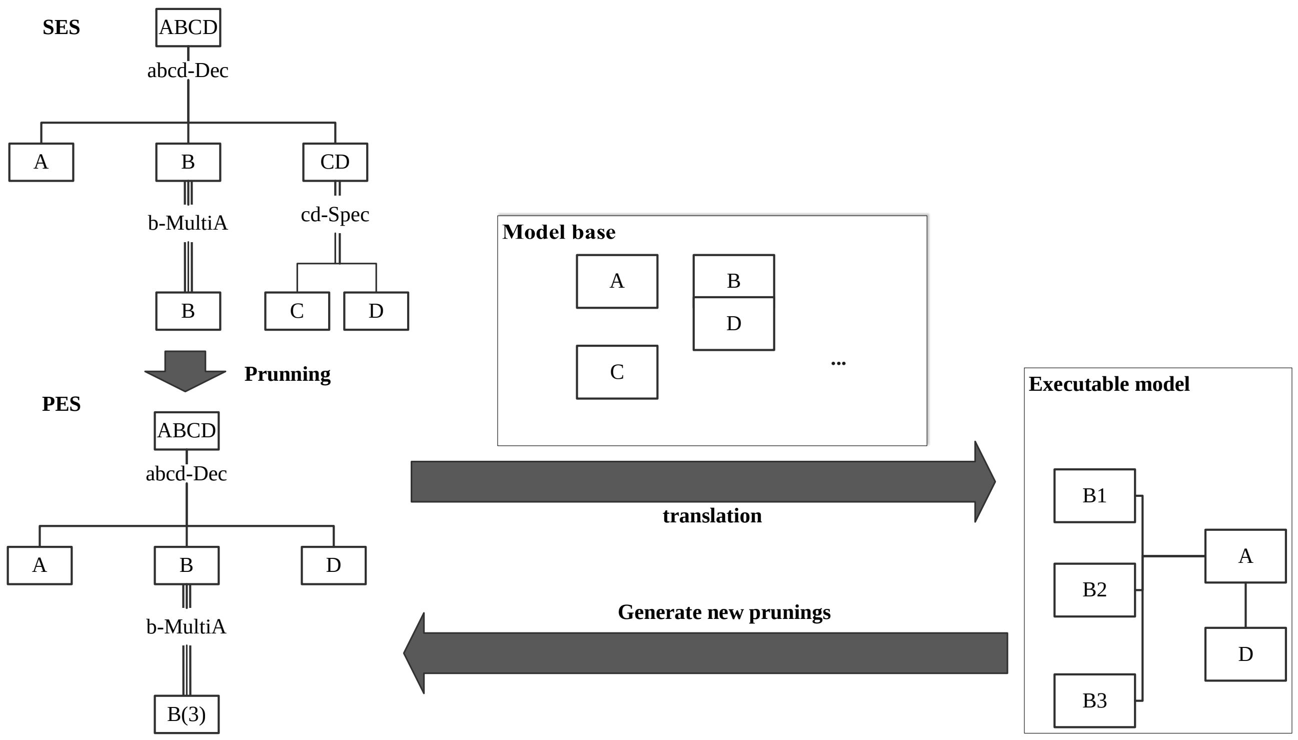


Figure 1. SES/MB Modeling and Simulation Process.

In SES modeling, a modeler needs to prune a system-decomposition tree by selecting a unique system configuration from all possible system architectures. The result is a tree known as the pruned entity structure (PES). After that, models in the MB are synthesized with the PES to compose an executable model for simulation. The simulation results can then assist the modeler in the tradeoff of different design features. If necessary, multiple PESs and multiple rounds of simulations can be conducted to reach an optimal solution.

2.3. X Language and XLab

X language [12,13,26] is an integrated modeling and simulation language that supports object-oriented modeling and has the ability to (1) support graphical modeling and code modeling, as well as (2) support continuous, discrete, agent, and hybrid modeling.

As illustrated in Figure 2, the X language has three analysis diagrams and five types of couple classes. The analysis diagrams include the requirement diagram for requirement description, analysis, and tracking, the use-case diagram for functional description, and the sequence diagram to describe the behavior of the system. In X language, these three diagrams cannot be converted into codes and are only used for analysis.

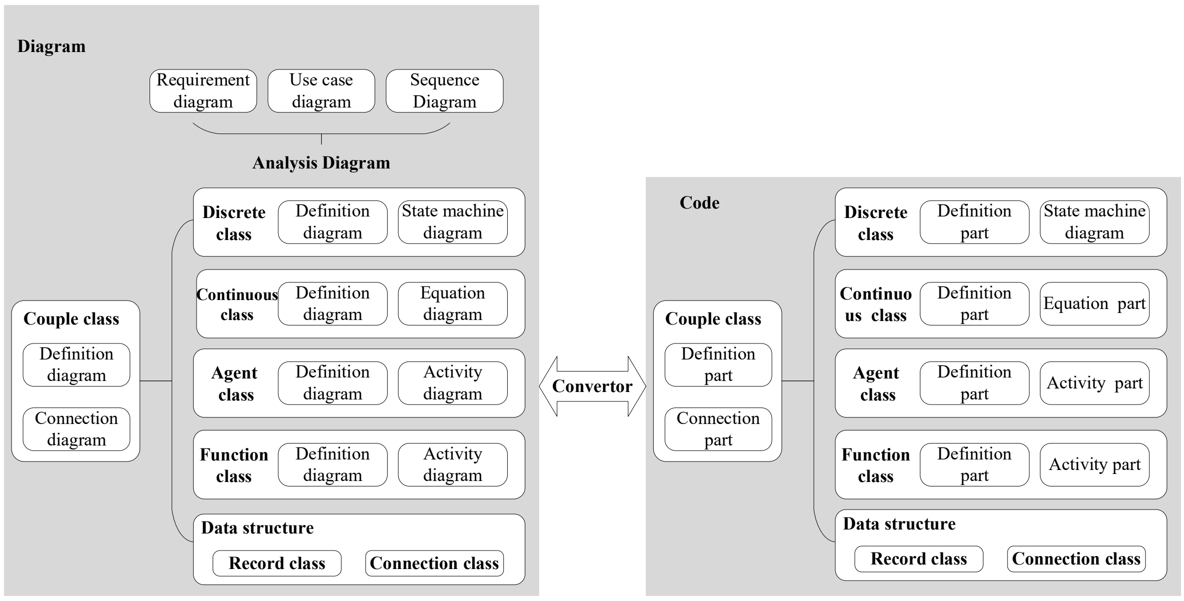


Figure 2. Class and elements of X language.

The five types of couple classes, however, have two formats: diagram format and code format, which can be transformed into each other. The graphic model is composed of diagrams, and the code format model is composed of parts. Each class has a definition diagram/part and a connection diagram/part, which define the system model at the level of the system structure, explaining which components the system contains and the connections between them. In the current X language, the connection diagrams/parts include an equation, activity, and state machine, which describe the system behavior with mathematical equations, assignment processes, and the DEVS atomic model, respectively.

XLab [14] is a web-based software that supports X language modeling and simulation, which supports both X language diagrams and X language coding. The diagram and code format of models can be converted into each other in XLab, and the code format model can be compiled into a simulation model by the XLab compiler and executed by a simulation engine written in C++ [13].

3. SES-X Methodology

Following the philosophy of SES, we propose the SES-X methodology that aims to model the entire process of system modeling and verification as a tree. As shown in Table 1, the major difference between SES and SEX-X is that SEX-X is a process-modeling methodology rather than a system-modeling methodology, as SES is. Its nodes are models or documents rather than entities. When using SES-X, the main process is to first derive modeling based on requirements, as specified by the upper-level systems. The SES-X method contains four stages—the analysis stage, the architecture decomposition stage, the physical modeling stage, and the validation/simulation stage—corresponding to this process. In the middle of the four stages, we have two processes to prune the non-computational models/documents to conduct model verification and simulation, as illustrated in Figure 3. These two pruning steps not only make the simulation feasible but also limit the search space for modeling and simulation. Below, we specify the detailed procedure of each step.

Table 1. Difference between SES and SES-X.

	SES	**SES-X**
Purpose	System architecture design	MBSE process design
Nodes	Entities that form a system	Models/Modules in a modeling processes
Decompose	Detail components of a system	Detail models meet (upper level) requirements
Prune	On the entire tree of system models	On subtrees of the modeling process
	One round of pruning	Two rounds of pruning
	Select one specialization of an entity	Select specialization of a system architecture or specification of an entity

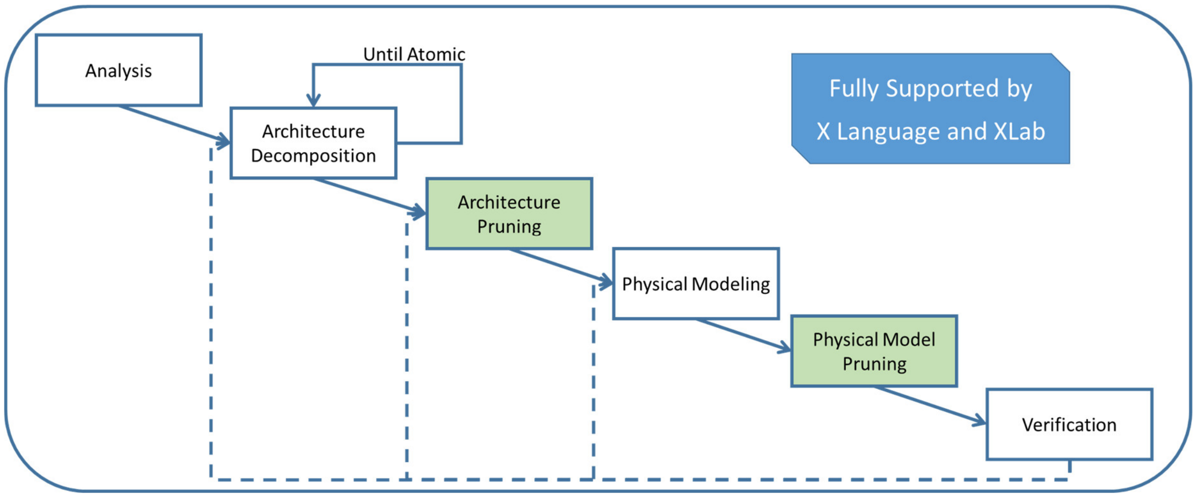


Figure 3. SES-X Modeling and Simulation Process.

3.1. Analysis Stage

The purpose of this stage is to abstract the system model and establish the overall system architecture rather than the specific logic and implementation of the model.

First, the modeler builds a requirements model by analyzing stakeholder requirements and clarifying the system functional specification, such as through the use case diagram. Thus, the first level of SES-X contains three elements: requirement model, use case model, and system architecture. The requirement model describes the requirements and constraints that stakeholders have on the system. The use case model reflects the functions that the system must provide to the user.

Second, the modeler needs to build the system architecture. This part of SES-X is composed of two nodes: functional architecture and entity structure. The functional architecture describes the system's processing logic. The entity structure describes the entity and relations of the data. At this stage, the modelers need to establish operation scenarios for each function of the system, identify the operation relations or constraint relations between each function, and design new functions based on interactions of functions. Further, according to the function interaction relations, the modeler establishes the entities supporting the designed functions.

After this stage, the system-level model of SES-X will look like Figure 4.

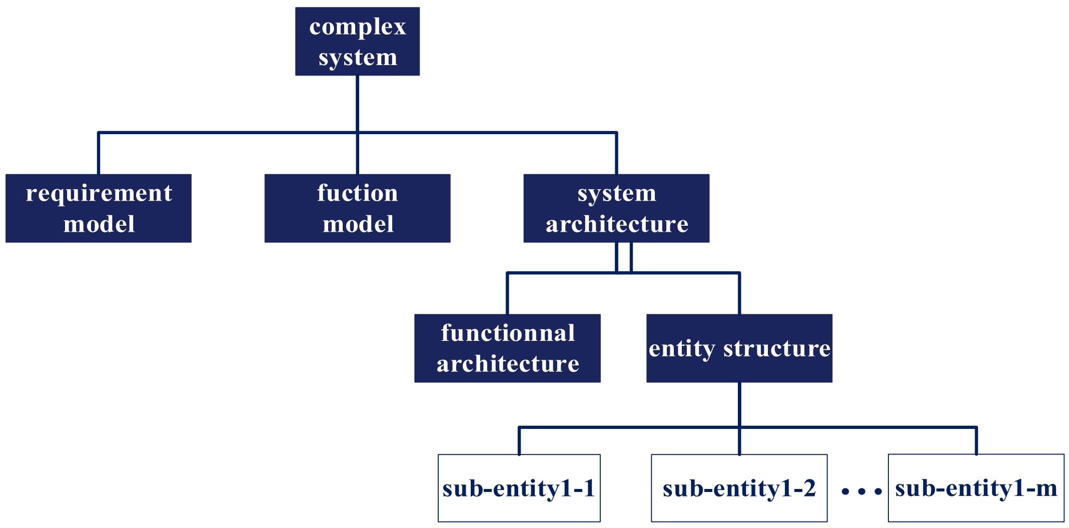


Figure 4. The Analysis Model of SES-X.

3.2. Architecture Decomposition

The second stage is the architecture-decomposition stage, which decomposes the entities constructed in the first stage into sub-entities. As with the entire system, the sub-entities are also composed of requirements, use cases, and system architectures. The decomposition follows a top-down process and is conducted recursively until non-divisible atomic entities are reached.

In SES-X, each sub-entity is responsible for a portion of the functions and entities of the upper-level system. Coupling and interactions also exist among its functions and entities. Thus, the sub-entities are connected with requirement models and use-case models at a higher level. This stage also needs to sort out the logic of each entity, such as the running sequence and state transition relationship, according to the allocated functions, so as to optimize the function and modeling of each sub-entity.

After this stage, one will be able to obtain the SES-X model shown in Figure 5.

3.3. Architecture Pruning

After architecture decomposition, we will have a complicated SES-X tree. To manage the scale of the model for functional verification (i.e., to verify whether the logic of the system can meet the design requirements), we need to prune the tree. At this stage, by coupling the function of sub-entities, their alignment with the requirements of a higher-level system is tested. Note that this tree-pruning step is different from SES in the sense that we not only prune entities but also prune models and a sub-tree from the entire tree. We need to prune all supporting elements that cannot be executed in simulation, such as the requirement model and use case model, in SES-X. The testing is currently performed by the modeler, while automated algorithms for functionality integration verification and model pruning will be investigated in the future.

As illustrated in Figure 6, the pruned model only contains entity elements that form a PES (as in SES). This design allows us to test if the process of the sub-entity meets the design requirements before lower-level entities are integrated.

3.4. Physical Modeling

In the third stage, appropriate physical models need to be built on entities to enable the simulation. Unlike in the previous stage, the physical models at this stage will be used in simulation and need to be designed based on the nature of the component/sub-entity. Thus, a bottom-up design process will be used in this phase.

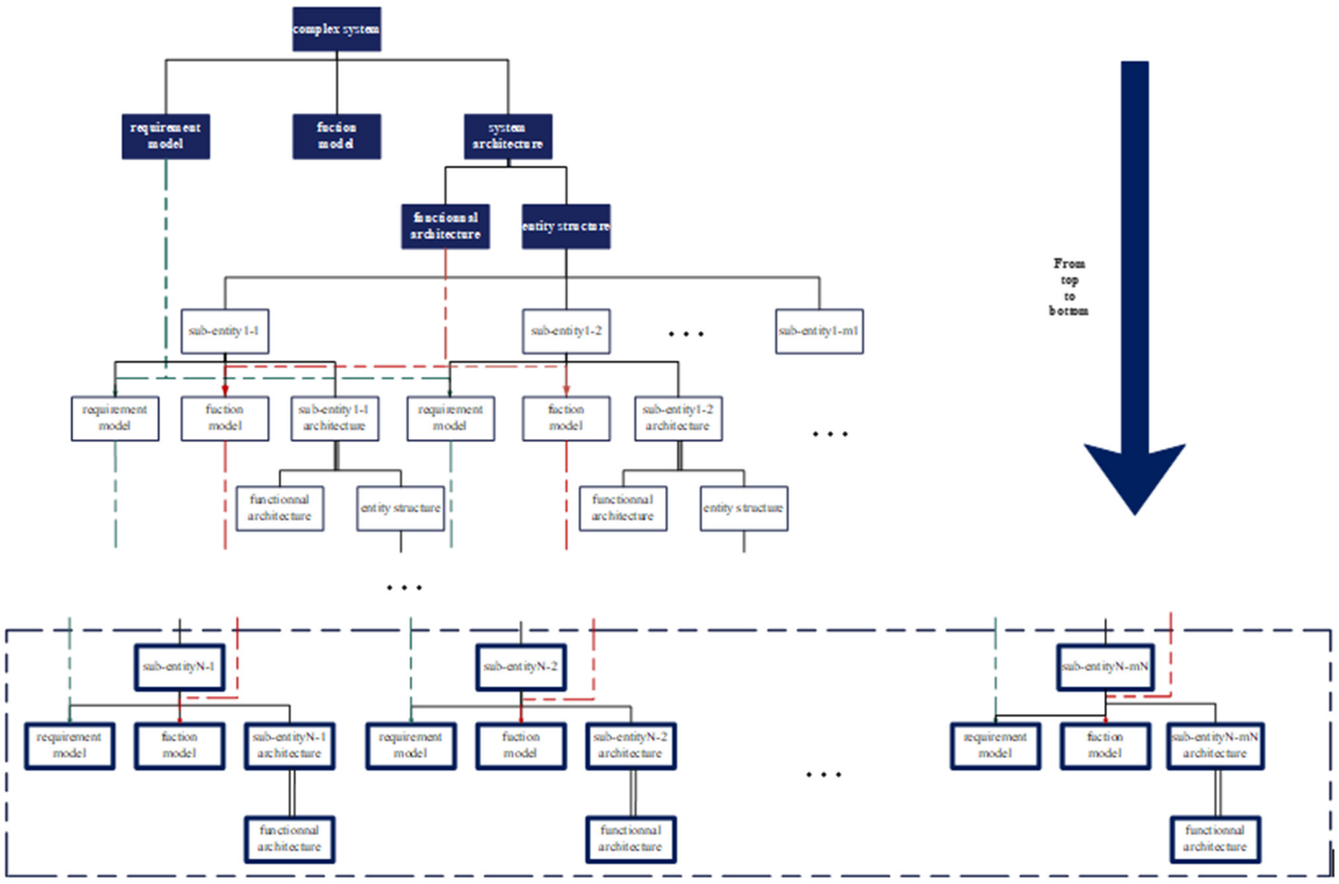


Figure 5. Architecture Deposition of SES-X.

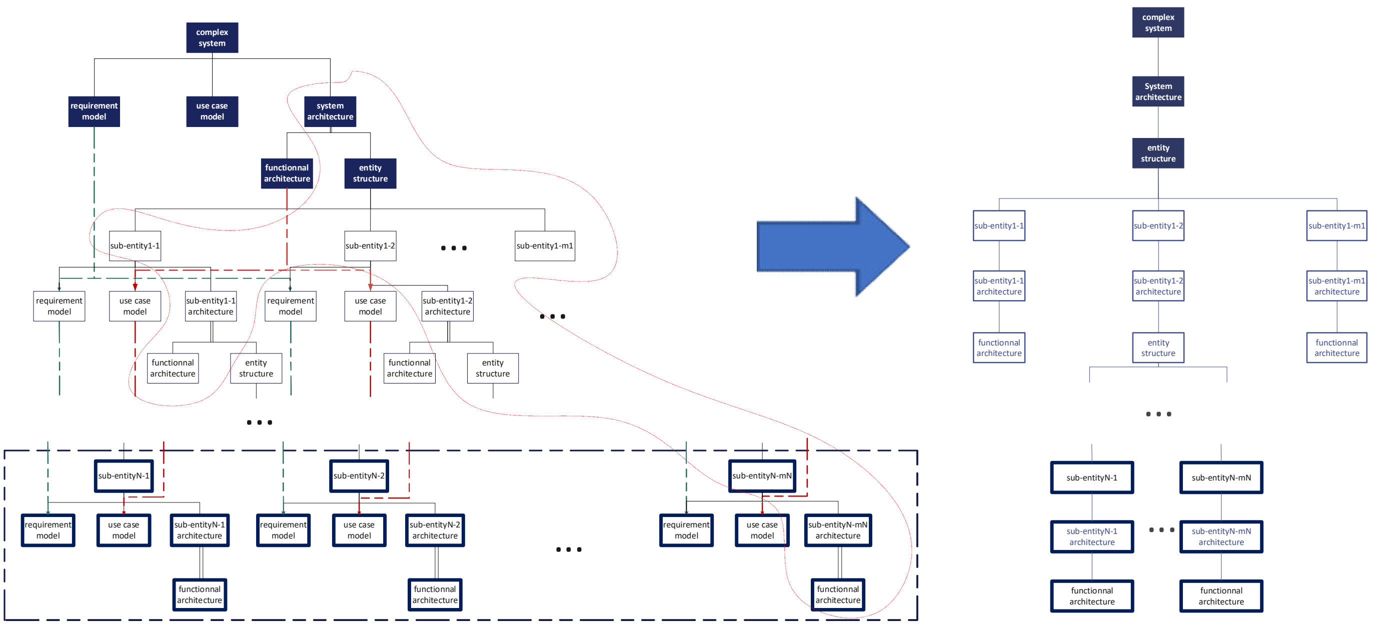


Figure 6. Architecture pruning of SES-X.

This phase focuses on the physical mechanisms of the entities that make up the system. It is possible that the physical models of atomic entities are already defined in the model base. Meanwhile, since each entity may be modeled by different dynamic models, the systems that meet the same function requirements may have a different physical model. Starting from each atomic model, the relationship between models is updated recursively to the upper levels. Eventually, physical model connections are mapped with the system model specification established before.

After this stage, one will be able to obtain the SES-X model, as shown in Figure 7 (middle part). The SES-X model obtained at this stage is an SES, which describes a set of possible system structures for the system model. This SES can be used, as in SES/MB, to simulate and validate the system.

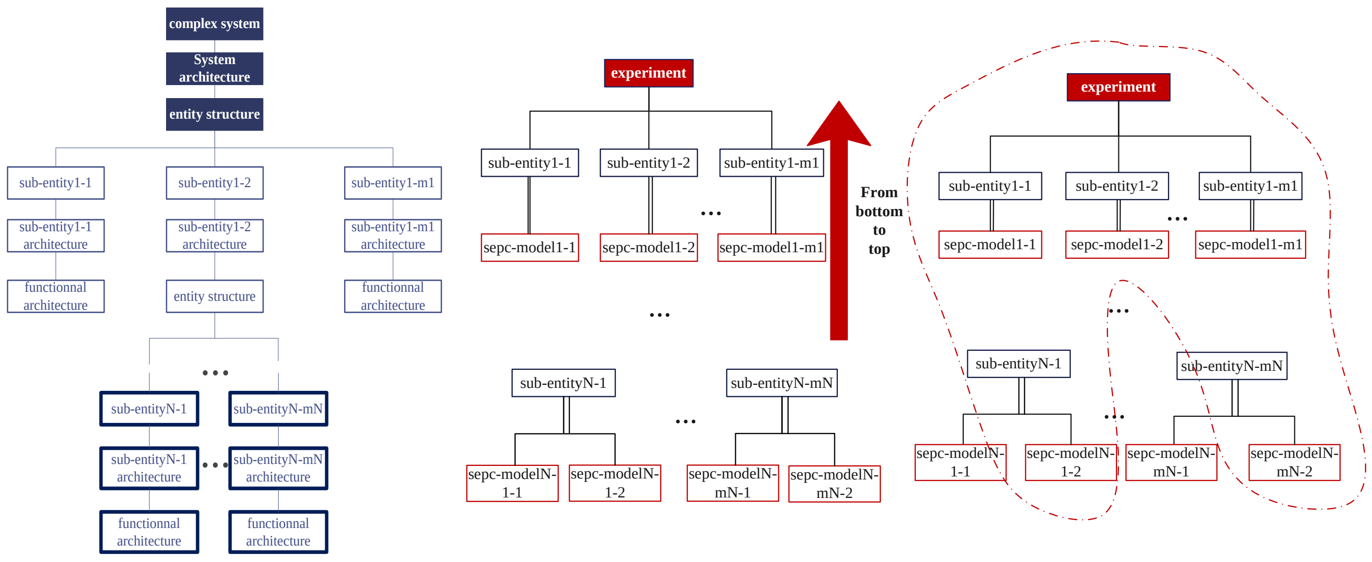


Figure 7. Physical model and pruned physical model of SES-X.

3.5. Physical Model Pruning

At this point, we would have a physical model aligning with the SES-X framework. However, to conduct the simulation, we need to prune further to obtain the prototype model for simulation. In this step, the model specializations are determined for the entities of the system, and a system model is finally obtained. Note that this physical model pruning step is based on system models that support the simulation of system dynamics. This is different from architecture pruning in Section 3.3, which is only for functionality checks. The goal of this stage is to make a determination from multiple alternatives of system models and obtain a prototype entity structure that supports the simulation of the system.

After this stage, one will be able to obtain the SES-X model shown in Figure 7 (right part).

3.6. Simulation Verification

The last stage is system simulation verification. At this stage, the interface of the model has been matched, the physical models are specified, and we will have an integrated multi-level system model. After the simulation parameters are specified, the system model can be transformed into a simulation model. The simulation process can be implemented with X language and XLab.

As illustrated in Figure 3, our proposed SES-X is an iterative process. After simulation, if the outcome of the system cannot meet the requirements, we may need to revise the pruning of model design for the physical model or architecture model and reconduct the simulation. The final solution will be reached when system requirements are met.

4. Case Study

In this section, we illustrate the use of the SES-X methodology, through a toy application, to build a car model. When designing a car, a car manufacturer needs to take into account various factors. For the purpose of this paper, we only consider the most basic requirements, in which a car has a frame and tires, and we only consider the dynamics of the frame under the action of the driver (i.e., turning the steering wheel).

4.1. System Analysis

In the first stage, we can build a system architecture, as in Figure 8. Its requirements diagram specifies the most basic requirement for car modeling: the car is able to respond to the driver's action. To meet this requirement, two sub-requirements are needed:

1. The frame can report the speed of the car during the steering process.
2. The tires can reflect the speed, force, and slip rate.

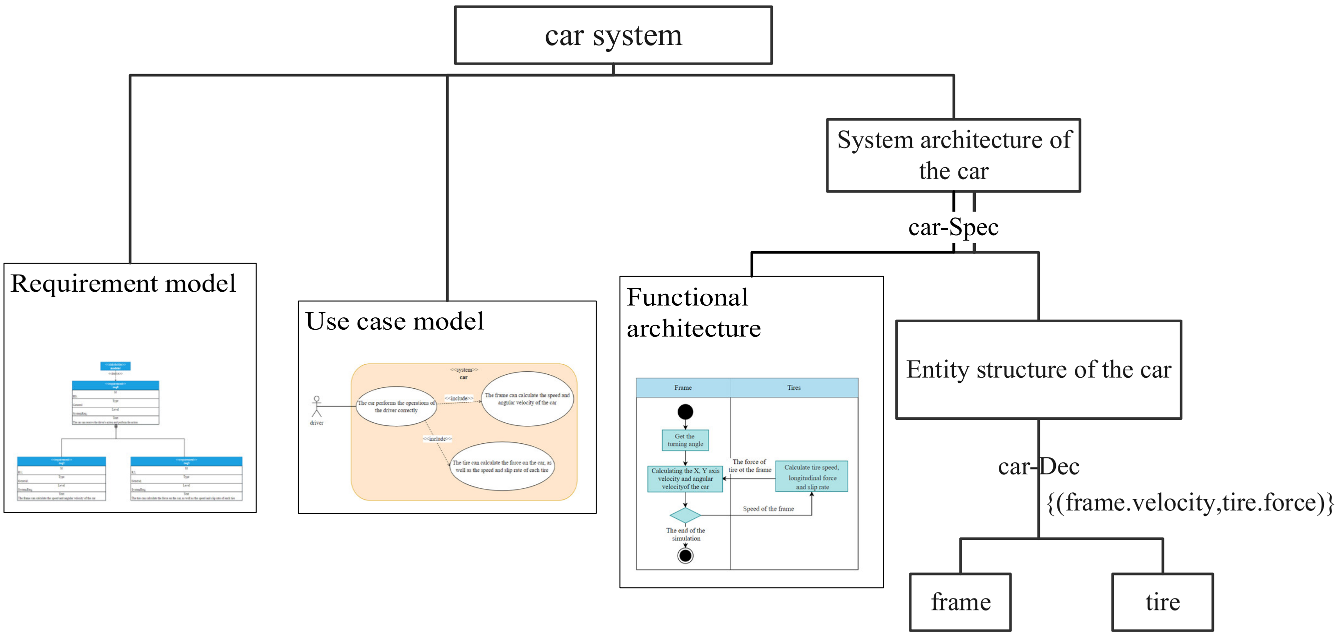

car system
System architecture of the car
car-Spec
Requirement model
Use case model
<<system>>
car
driver
The car performs the operations of the driver correctly
<<include>>
The frame can calculate the speed and angular velocity of the car
<<include>>
The tire can calculate the force on the car, as well as the speed and slip rate of each tire
Functional architecture
Frame
Tires
Get the turning angle
Calculating the X, Y axis velocity and angular velocityof the car
The force of tire ot the frame
Calculate tire speed, longitudinal force and slip rate
Speed of the frame
The end of the simulation
Entity structure of the car
car-Dec
{(frame.velocity,tire.force)}
frame
tire

Figure 8. The Analysis Model of A Car in SES-X.

To meet these two requirements, we can build the use case model in the use case diagram and the system architecture. The functional architecture of the system is mainly concerned with the interaction between the body and the wheel, where the vehicle speed is affected by the angle of the front wheel and the force of the wheel on the body. The speed of the wheel is mechanically determined by the speed of the body. Thus, we derive functional architecture using the activity diagram. Accordingly, the automobile system can be considered to be composed of two sub-entities, namely the frame and the tire.

4.2. Architecture Decomposition

Then, we further design the two sub-entities: frame and tire. Among them, the frame model is relatively easy since it does not have any moving parts and cannot be further divided into multiple entities. The frame model can be considered an atomic entity, as shown in Figure 9.

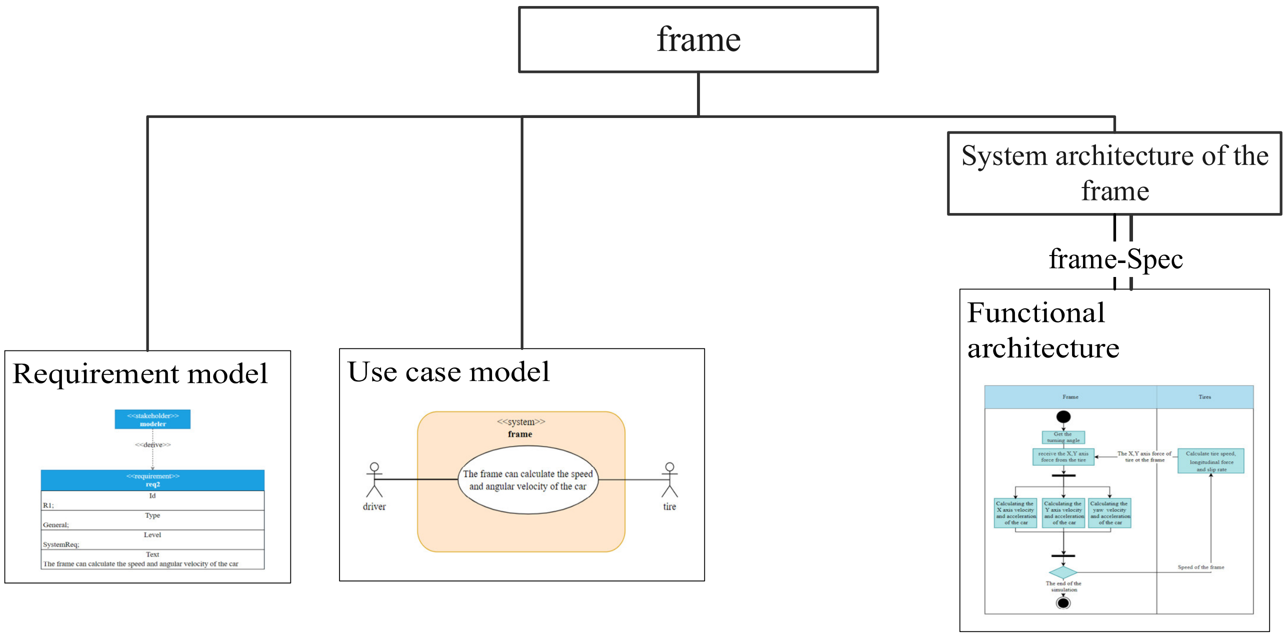

frame
System architecture of the frame
frame-Spec
Functional architecture
Requirement model
Use case model
driver
tire
The frame can calculate the speed and angular velocity of the car

Figure 9. The SES-X Model of The Frame.

The tire model has a few requirements, including:

1. The calculation of the vertical load of the wheel,
2. The calculation of the wheel side angle,
3. The calculation of the speed of each wheel in the wheel coordinate system,
4. The calculation of the wheel slip rate,
5. The calculation of the lateral and longitudinal forces of the wheel, and
6. The slip rate, which should not exceed 10%.

Based on these requirements, Figure 10 shows the requirement model and use case model of the tires, including vertical load calculation, wheel side angle calculation, wheel coordinate system speed calculation, wheel slip rate calculation, and wheel lateral and longitudinal force calculation.

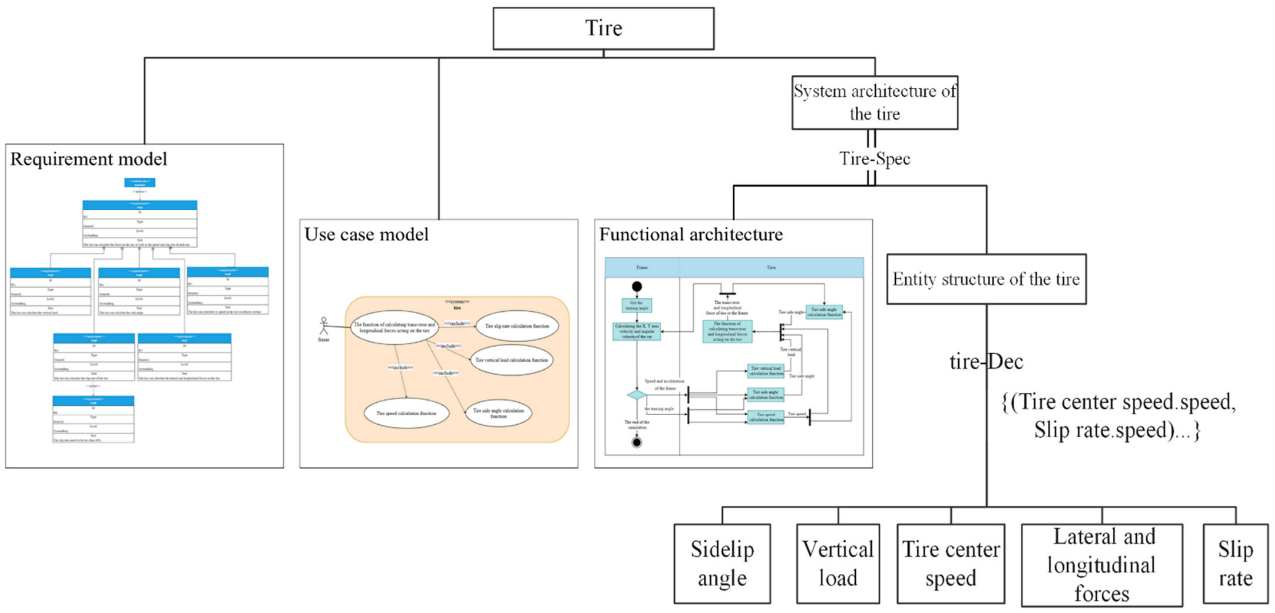

Tire
System architecture of the tire
Tire-Spec
Requirement model
Use case model
Functional architecture
Entity structure of the tire
tire-Dec
{(Tire center speed.speed, Slip rate.speed)...}
Sidelip angle
Vertical load
Tire center speed
Lateral and longitudinal forces
Slip rate

Figure 10. The SES-X model of the tires.

Based on the analysis of these functions, the solid structure of the tire model can be divided into a vertical load calculation module, a wheel side angle calculation module, a wheel coordinate system speed calculation module, a wheel slip-rate calculation module, and a wheel lateral and longitudinal force calculation module. These modules are (assumed to be) atomic entities that do not need to be further divided and, together, form the sub-entity of ties in the architecture part in Figure 10. At this stage, the system architecture can be pruned for testing purposes. Since the model is very simple, the pruned model is the same as the original model and is not reported here.

4.3. Physical Model

Then, we take a bottom-up approach to build a physical model in alignment with the logical model. Specifically, the physical model of the frame can be built through partial differential equations, as follows:

$$m\left(\dot{V}_x - r\cdot V_y\right) = \left(F_{xfl} + F_{xfr}\right)\cos(\delta) - \left(F_{yfl} + F_{yfr}\right)\sin(\delta) + F_{xrl} + F_{xrr} \tag{1}$$

$$m(\dot{V}_y + r\cdot V_x) = (F_{xfl} + F_{xfr})\sin(\delta) + (F_{yfl} + F_{yfr})\cos(\delta) + F_{yrl} + F_{yrr} \tag{2}$$

$$\begin{aligned} &I_z\cdot\dot{r} = \Big[(F_{xfl} + F_{xfr})\sin(\delta) + (F_{yfl} + F_{yfr})\cos(\delta)\Big]l_a + \\ &\Big[(F_{xfr} - F_{xfl})\cos(\delta) + (F_{yfl} - F_{yfr})\sin(\delta)\Big]\tfrac{t_{w1}}{2} + \\ &(F_{xrr} - F_{xrl})\tfrac{t_{w2}}{2} - (F_{yrl} + F_{yrr})l_b \end{aligned} \tag{3}$$

where Equation (1) models the longitudinal force balance for the frame, Equation 2 models the lateral force balance for the frame, and Equation 3 models the torque balance for the frame around the Z-axis. In these equations, δ is the steering angle of the front wheel of the car; m is the mass of the frame; V_x, V_y, and r are the longitudinal, lateral, and yaw velocities, respectively; F_{xi}, F_{yi}, and F_{zi} are the longitudinal, lateral, and vertical force of the tire, respectively, where *fl* is left front wheel, *fr* is right front wheel, *rl* is left rear wheel, and *rr* is right rear wheel; l_a and l_b are the distances from the leading and trailing axes to the center of mass; t_{w1} is the front axle wheel pitch, and t_{w2} is the rear axle wheel pitch; I_z is the moment of inertia of the vehicle with respect to the Z-axis. These equations can be constructed with the X-language shown in Figure 11. The physical models of the tires can be built in a similar fashion.

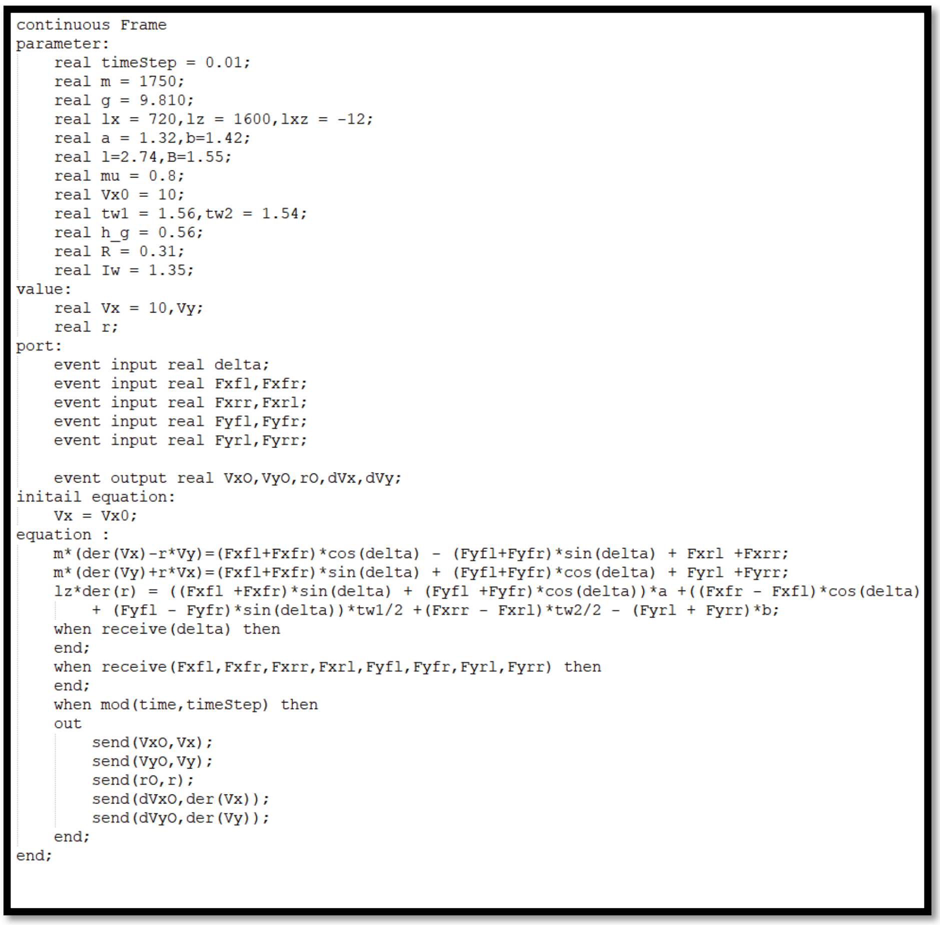
After this stage, one will be able to obtain an SES-X of the car, as shown in Figure 12. Each leaf node of this SES-X corresponds to a model constructed using the X language, such as the specialized node of the frame corresponding to the *Frame* model shown in Figure 11.

4.4. Model Pruning

After the SES of the car is built, it can be pruned for effective modeling. Since our model includes only one level of composition, an entity has only one specialization. The PES is simple, as illustrated in Figure 13.

4.5. Simulation

Then, XLab allows us to connect the frame model with the tire model based on the system architecture model (as shown in Figure 14), which can be put forward for simulation. In our particular example, Figure 15 reports the dynamic process of making a turning angle of 0.1 radians. It can be seen that the slip rate does not exceed 0.01 in the simulation process, which meets requirement 6 of the tire model. Therefore, the established car model design would meet the system requirements and can allow us to move to the next stage of the development process.



```
continuous Frame
parameter:
    real timeStep = 0.01;
    real m = 1750;
    real g = 9.810;
    real lx = 720,lz = 1600,lxz = -12;
    real a = 1.32,b=1.42;
    real l=2.74,B=1.55;
    real mu = 0.8;
    real Vx0 = 10;
    real tw1 = 1.56,tw2 = 1.54;
    real h_g = 0.56;
    real R = 0.31;
    real Iw = 1.35;
value:
    real Vx = 10,Vy;
    real r;
port:
    event input real delta;
    event input real Fxfl,Fxfr;
    event input real Fxrr,Fxrl;
    event input real Fyfl,Fyfr;
    event input real Fyrl,Fyrr;

    event output real VxO,VyO,rO,dVx,dVy;
initail equation:
    Vx = Vx0;
equation :
    m*(der(Vx)-r*Vy)=(Fxfl+Fxfr)*cos(delta) - (Fyfl+Fyfr)*sin(delta) + Fxrl +Fxrr;
    m*(der(Vy)+r*Vx)=(Fxfl+Fxfr)*sin(delta) + (Fyfl+Fyfr)*cos(delta) + Fyrl +Fyrr;
    lz*der(r) = ((Fxfl +Fxfr)*sin(delta) + (Fyfl +Fyfr)*cos(delta))*a +((Fxfr - Fxfl)*cos(delta)
        + (Fyfl - Fyfr)*sin(delta))*tw1/2 +(Fxrr - Fxrl)*tw2/2 - (Fyrl + Fyrr)*b;
    when receive(delta) then
    end;
    when receive(Fxfl,Fxfr,Fxrr,Fxrl,Fyfl,Fyfr,Fyrl,Fyrr) then
    end;
    when mod(time,timeStep) then
    out
        send(VxO,Vx);
        send(VyO,Vy);
        send(rO,r);
        send(dVxO,der(Vx));
        send(dVyO,der(Vy));
    end;
end;
```

Figure 11. The Frame Model in X language.

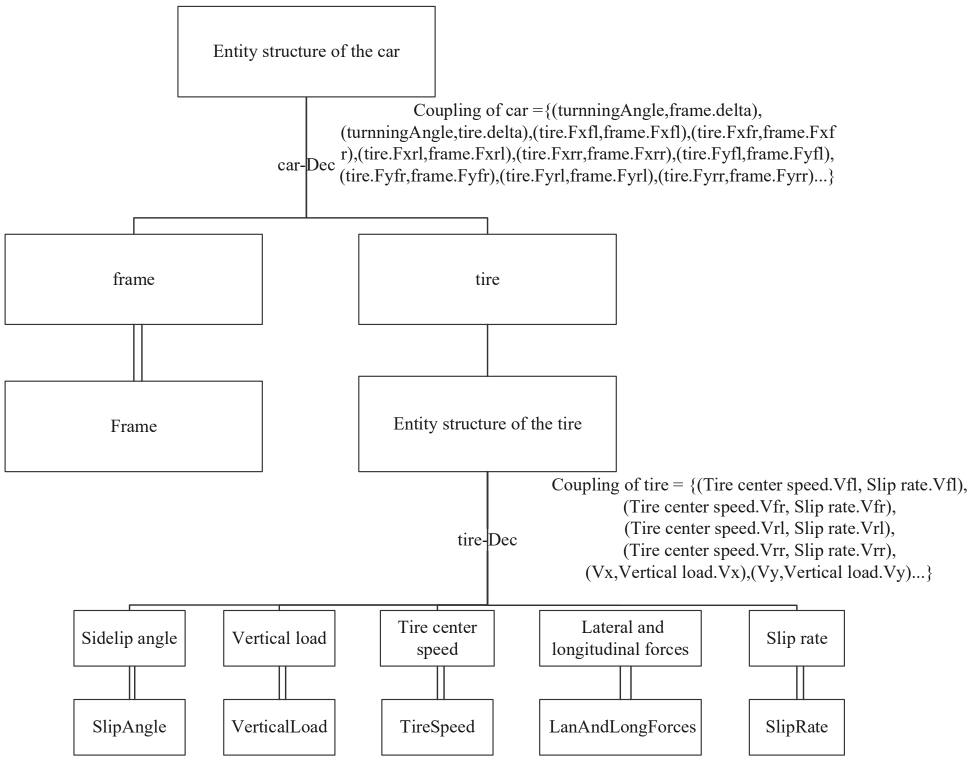

Entity structure of the car
Coupling of car ={(turnningAngle,frame.delta), (turnningAngle,tire.delta),(tire.Fxfl,frame.Fxfl),(tire.Fxfr,frame.Fxfr),(tire.Fxrl,frame.Fxrl),(tire.Fxrr,frame.Fxrr),(tire.Fyfl,frame.Fyfl), (tire.Fyfr,frame.Fyfr),(tire.Fyrl,frame.Fyrl),(tire.Fyrr,frame.Fyrr)...}
car-Dec
frame
tire
Frame
Entity structure of the tire
Coupling of tire = {(Tire center speed.Vfl, Slip rate.Vfl), (Tire center speed.Vfr, Slip rate.Vfr), (Tire center speed.Vrl, Slip rate.Vrl), (Tire center speed.Vrr, Slip rate.Vrr), (Vx,Vertical load.Vx),(Vy,Vertical load.Vy)...}
tire-Dec
Sidelip angle
Vertical load
Tire center speed
Lateral and longitudinal forces
Slip rate
SlipAngle
VerticalLoad
TireSpeed
LanAndLongForces
SlipRate

Figure 12. The SES-X of a car.

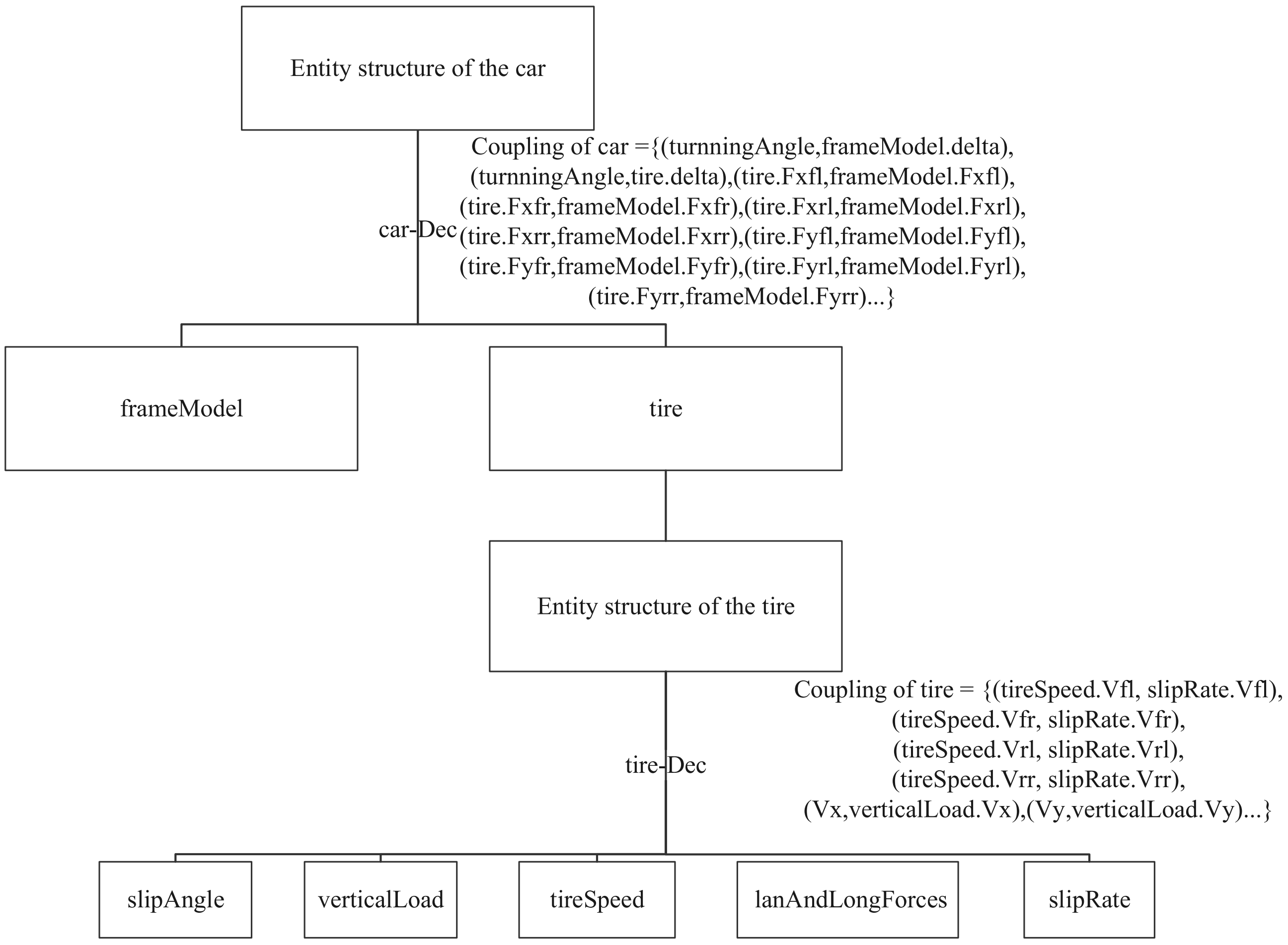

Entity structure of the car
car-Dec
Coupling of car ={(turnningAngle,frameModel.delta),(turnningAngle,tire.delta),(tire.Fxfl,frameModel.Fxfl),(tire.Fxfr,frameModel.Fxfr),(tire.Fxrl,frameModel.Fxrl),(tire.Fxrr,frameModel.Fxrr),(tire.Fyfl,frameModel.Fyfl),(tire.Fyfr,frameModel.Fyfr),(tire.Fyrl,frameModel.Fyrl),(tire.Fyrr,frameModel.Fyrr)...}
frameModel
tire
Entity structure of the tire
tire-Dec
Coupling of tire = {(tireSpeed.Vfl, slipRate.Vfl),(tireSpeed.Vfr, slipRate.Vfr),(tireSpeed.Vrl, slipRate.Vrl),(tireSpeed.Vrr, slipRate.Vrr),(Vx,verticalLoad.Vx),(Vy,verticalLoad.Vy)...}
slipAngle
verticalLoad
tireSpeed
lanAndLongForces
slipRate

Figure 13. Pruned SES-X model of a car.

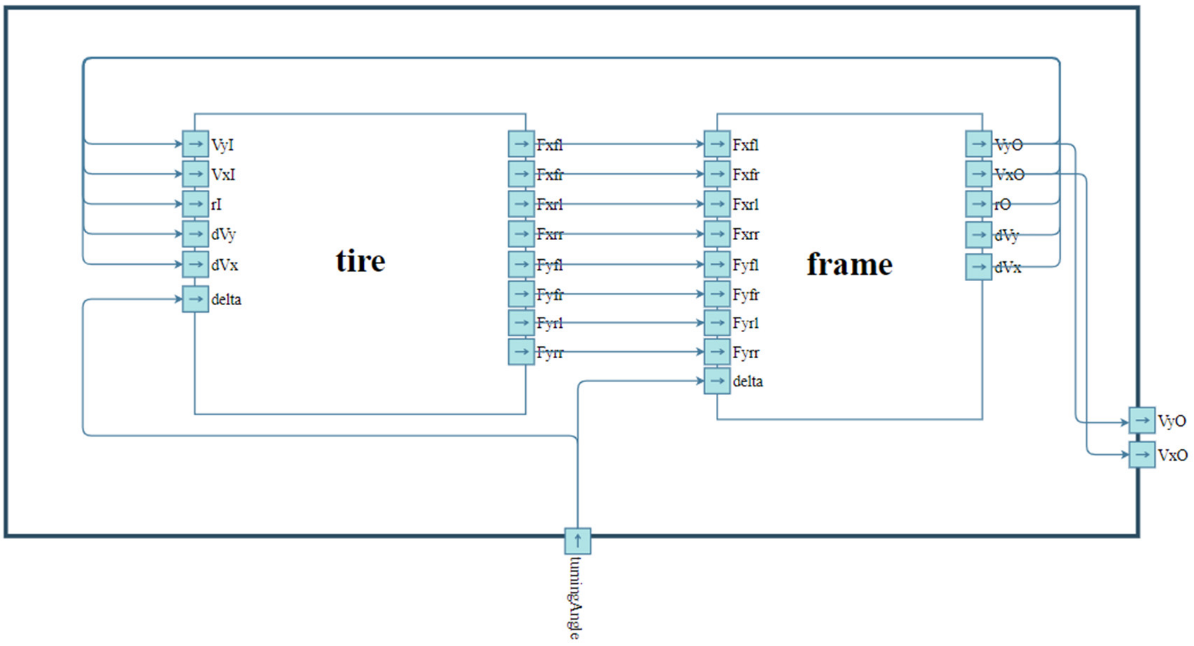


Figure 14. Car model connecting the frame with tires in X language.

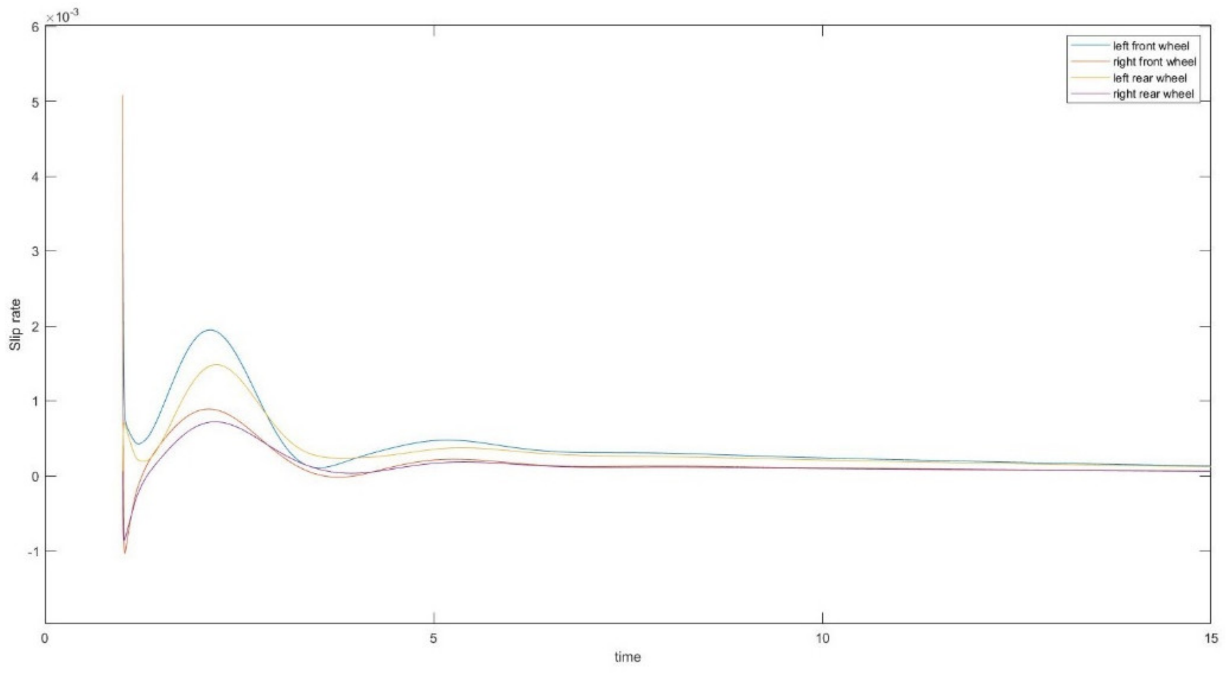


Figure 15. Simulation result of tire slip rate.

5. Conclusions

This paper proposes an SES-X methodology that integrates system modeling (following the SES philosophy) with system simulation (supported by the X language). SES-X supports the entire process of modeling, including system analysis, architecture decomposition, physical modeling, and simulation. In the process, SES-X was used to conduct two levels of model pruning for the efficiency of modeling and simulation. This paper uses a car model to demonstrate the effectiveness of SES-X.

Our proposed SES-X methodology has significant theoretical and practical implications. Theoretically, it is a native modeling framework incorporating system modeling with system simulation functionalities. To the best of our knowledge, it is the most comprehensive MBSE modeling framework so far, and it can provide complete modeling and verification functions for the entire process of the system model construction. Practically, SES-X could significantly smooth the design and development process of complicated systems.

Our proposed SES-X methodology can be further improved in the future. For example, the SES-X process is not yet fully coded into X language (at the syntax level). X language can only support a limited number of functions and physical verification methods needed by SES-X. In the future, we will continue to work on SES-X and provide a set of user-friendly solutions that fully support MBSE modeling with more advanced simulation capabilities. We also look forward to large-scale applications of this solution in practice.

Author Contributions: Conceptualization, K.X. and L.Z.; methodology, K.X., L.Z., X.L., P.G. and Z.C.; software, K.X.; validation, K.X. and X.L.; formal analysis, K.X.; investigation, K.X., P.G. and Z.C.; resources, K.X. and L.Z.; data curation, K.X.; writing—original draft preparation, K.X.; writing—review and editing, K.X., L.Z. and X.L.; visualization, K.X.; supervision, L.Z.; project administration, L.Z.; funding acquisition, L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by China’s National Key R&D Program, No. 2018YFB1701600.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zeigler, B.P.; Mittal, S.; Traore, M.K. MBSE with/out Simulation: State of the Art and Way Forward. *Systems* **2018**, *6*, 40. [CrossRef]
2. Zeigler, B.P. DEVS and MBSE: A review. *Int. J. Model. Simulat. Sci. Comput.* **2022**, *13*. [CrossRef]
3. Delligatti, L. *SysML Distilled: A Brief Guide to the Systems Modeling Language*; Addison-Wesley Professional: Boston, MA, USA, 2013.
4. Amissah, M.; Toba, A.-L.; Handley, H.A.H.; Seck, M. Towards a Framework for Executable Systems Modeling: An Executable Systems Modeling Language (ESysML). In Proceedings of the Model-driven Approaches for Simulation Engineering Symposium, Baltimore, MD, USA, 15–18 April 2018; pp. 1–12. [CrossRef]
5. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. Model Transformation Services for MSaaS platforms. In Proceedings of the Model-driven Approaches for Simulation Engineering Symposium, Baltimore, MD, USA, 15–18 April 2018; pp. 1–12. [CrossRef]
6. Aliyu, H.O.; Maïga, O.; Traoré, M.K. The high level language for system specification: A model-driven approach to systems engineering. *Int. J. Model. Simulat. Sci. Comput.* **2016**, *7*, 1641003. [CrossRef]
7. Bock, C.; Barbau, R.; Matei, I.; Dadfarnia, M. An Extension of the Systems Modeling Language for Physical Interaction and Signal Flow Simulation. *Syst. Eng.* **2017**, *20*, 395–431. [CrossRef]
8. Kapos, G.-D.; Dalakas, V.; Nikolaidou, M.; Anagnostopoulos, D. An integrated framework for automated simulation of SysML models using DEVS. *Simulation* **2014**, *90*, 717–744. [CrossRef]
9. Hause, M. OMG Systems Modeling Language (OMG SysML™) Tutorial. *INCOSE Int. Symp.* **2009**, *19*, 1840–1972. [CrossRef]
10. Fritzson, P.; Engelson, V. Modelica—A unified object-oriented language for system modeling and simulation. In *ECOOP'98—Object-Oriented Programming*; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1445, pp. 67–90. [CrossRef]
11. Zeigler, B.P.; Praehofer, H.; Kim, T.G. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*; Academic Press: San Diego, CA, USA, 2000.
12. Zhang, L.; Ye, F.; Laili, Y.; Xie, K.; Gu, P.; Wang, X.; Zhao, C.; Zhang, X.; Chen, M. X Language: An Integrated Intelligent Modeling and Simulation Language for Complex Products. In Proceedings of the 2021 Annual Modeling and Simulation Conference (ANNSIM), Fairfax, VA, USA, 19–22 July 2021. [CrossRef]
13. Zhang, L.; Ye, F.; Xie, K.; Gu, P.; Wang, X.; Laili, Y.; Zhao, C.; Zhang, X.; Chen, M.; Lin, T.; et al. An Integrated Intelligent Modeling and Simulation Language for Model-based Systems Engineering. *J. Ind. Inf. Integr.* **2022**, *28*, 100347. [CrossRef]
14. Gu, P.; Zhang, L.; Chen, Z.; Ye, J. Collaborative Design and Simulation Integrated Method of Civil Aircraft Take-off Scenarios Based on X Language. *J. Syst. Simul.* **2022**, *34*, 929–943. [CrossRef]
15. Yang, Z.; Du, H.; Liu, Y.; Liu, R.; Liu, Y. Use the Harmony-SE Approach to Extend the Advantages of MBSE. In Proceedings of the 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA), Chengdu, China, 1–4 August 2021; pp. 223–227. [CrossRef]
16. Morkevicius, A.; Aleksandraviciene, A.; Armonas, A.; Fanmuy, G. Towards a Common Systems Engineering Methodology to Cover a Complete System Development Process. *INCOSE Int. Symp.* **2020**, *30*, 138–152. [CrossRef]
17. Kaindl, H.; Huber, S.; Karacan, Ö.; Kondo, I.; Schreiner, H.; Süß, H.-W. ooSEM (poster session): A process model for object-oriented development in an industrial environment. In Proceedings of the OOPSLA '00: Addendum to the 2000 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum), New York, NY, USA, 1 January 2000; pp. 99–100. [CrossRef]
18. Chabibi, B.; Douche, A.; Anwar, A.; Nassar, M. Integrating SysML with Simulation Environments (Simulink) by Model Transformation Approach. In Proceedings of the 2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, 13–15 June 2016; pp. 148–150. [CrossRef]
19. Batarseh, O.; McGinnis, L.F. System modeling in SYsML and system analysis in Arena. In Proceedings of the 2012 Winter Simulation Conference (WSC), Berlin, Germany, 9–12 December 2012. [CrossRef]
20. Alshareef, A.; Seo, C.; Kim, A.; Zeigler, B.P. DEVS Markov Modeling and Simulation of Activity-Based Models for MBSE Application. In Proceedings of the 2021 Winter Simulation Conference (WSC), Phoenix, AZ, USA, 12–15 December 2021. [CrossRef]
21. Wymore, A.W. *Model-Based Systems Engineering*; CRC Press: Boca Raton, FL, USA, 2018.
22. Wach, P.; Zeigler, B.; Salado, A. Conjoining Wymore's Systems Theoretic Framework and the DEVS Modeling Formalism: Toward Scientific Foundations for MBSE. *Appl. Sci.* **2021**, *11*, 4936. [CrossRef]
23. Blochwitz, T. Functional Mock-Up Interface for Model Exchange and Co-Simulation. 2014. Available online: https://fmi-standard.org/Downloads (accessed on 1 January 2016).
24. Miller, J.; Mukerji, J. MDA Guide Version 1.0.1. 2003. Available online: http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf. (accessed on 30 November 2022).
25. Kim, T.; Lee, C.; Christensen, E.; Zeigler, B. System entity structuring and model base management. *IEEE Trans. Syst. Man. Cybern.* **1990**, *20*, 1013–1024. [CrossRef]
26. Xie, K.; Zhang, L.; Laili, Y.; Wang, X. XDEVS: A hybrid system modeling framework. *Int. J. Model. Simulation. Sci. Comput.* **2022**, *13*. [CrossRef]