Journal of Systems Engineering and Electronics
Vol. 37, No. 3, June 2026, pp.933 – 951

An MBPLE-enabled architecture design method for remote sensing satellites

WANG Yaodong[1,2], CAO Yue[3], YU Houman[2], and LIU Yusheng[1,*]

1. State Key Laboratory of Computer-Aided Design and Computer Graphics, Zhejiang University, Hangzhou 310058, China;
2. China Academy of Space Technology, Beijing 100094, China;
3. College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China

Abstract: Remote sensing satellites (RSS) are highly complex and customized from a common product family. It makes the traditional model-based system engineering (MBSE) method, which lacks architecture-level reusability, difficult to apply to their architecture design. Considering RSS has a relatively fixed common architecture from which the various design solutions are customized specific to different missions, the model-based product line engineering (MBPLE) methodology can be leveraged. In this paper, based on the analysis of the current RSS development process in practice and the main issues of current MBPLE methods, an RSS-specific MBPLE approach is proposed. Firstly, the RSS domain terminology is consolidated into the RSS-MBPLE SysML profile to support the construction of models in the design process. Then, the two key steps, i.e., architecture configuration and standalone product selection are efficiently conducted following the MBPLE principles supported by plugins of the mainstream SysML platform. Finally, a typical RSS control subsystem is illustrated as the case study to demonstrate the effectiveness of the proposed method. The results show that the proposed approach improves architecture-level reusability and design automation compared with traditional methodology, thereby reducing manual clone-and-modify efforts and enhancing the efficiency of RSS architecture design.

Keywords: remote sensing satellites, model-based product line engineering (MBPLE), domain modeling, architecture design.

DOI: 10.23919/JSEE.2026.000127

1. Introduction

With the rapid development of large and medium-sized remote sensing satellites (RSS), the traditional document-based systems engineering methods are gradually becoming difficult to adapt to the system development process with high quality, high speed, and high efficiency. On the other hand, the model-based system engineering (MBSE) method has begun to attract more and more attention from both industry and academia [1]. In this paradigm, the complex systems are described by semi-formal structured models from various viewpoints so that engineers can capture a complete and integrated view. Such integrated models can maintain traceability among different assets so that changes can be propagated efficiently. Besides that, the semi-formal semantics can avoid ambiguity to a certain extent when the models are communicated among stakeholders from various domains [2].

For large and medium-sized RSS, the system models are highly complex and customized. Although RSS is diverse, the basic functions of the satellite platforms and their components are consistent. Only due to differences in features such as payloads, working modes, orbits, etc., are a variety of RSS architecture variants formed. Such highly product family-based characteristics of RSS mean the architecture design of RSS should heavily rely on the reuse of the common basic architecture. However, in the traditional MBSE development process, the reuse of existing architectures is mainly achieved by directly cloning or specializing from existing models and then modifying the elements on this basis. It is not only error-prone but also has low efficiency. Therefore, due to the lack of an architecture-level reuse mechanism, it is obviously not suitable for the RSS development process in practice.

Manuscript received July 31, 2024.
*Corresponding author.
This work was supported by the National Key Research and Development Program of China (2023YFB3307201), the National Natural Science Foundation of China (62102367), and the Natural Science Foundation of Zhejiang Province (LQ22F020019).

It can be seen that the main issue that hinders architecture-level reuse is that the common and variable parts in the architecture models cannot be clearly distinguished and hence cannot efficiently configure the variable parts. This issue can be solved using the product line engineering (PLE) methodology. The PLE concept was first proposed in the field of software engineering [3] and is being gradually extended to the field of system engineering and combined with MBSE to initiate a new research domain, i.e., model-based PLE (MBPLE) [4].

Although there are related studies in the field of MBPLE, existing MBPLE methods cannot be applied to the RSS domain due to below issues:

(i) In RSS design, not only the architecture variant but also the design solutions including concrete standalone products should be issued to the downstream departments. Therefore, the RSS-specific MBPLE process should include two phases, i.e., architecture configuration and standalone product selection instead of only one phase like existing studies.

(ii) The two phases in the MBPLE process are not independent, instead, the problem and solution domains of phase 2 are constrained by phase 1. This relationship should be considered when applying MBPLE in the two phases.

(iii) Since the MBPLE methodologies should be applied in the two phases, it is necessary to differentiate their domain terminologies not only for human comprehension but also for computer comprehension to provide foundations for design automation.

In response to the above-mentioned issues, an RSS-MBPLE approach is proposed in this study. It first defines the RSS-MBPLE profile based on existing MBPLE terms to differentiate the domain terminologies of the two phases, i.e., architecture configuration and standalone product selection. Then, the RSS-MBPLE process is introduced, which not only streamlines the entire architecture design process by applying MBPLE in the two phases but also provides computer-aided design tools to automate certain steps to improve design efficiency.

2. Related works

2.1 Architecture design in MBSE

Architecture design is an important activity in the system engineering process. Many classic MBSE methodologies, e.g., object-oriented systems engineering method (OOSEM) [5], Harmony SE [6], Magic Grid [7], provide guidance for architecture design tasks. Based on these general methodologies, some in-depth research has been conducted on specific tasks in architecture design, such as architecture modeling, component retrieval and reuse, and architecture evaluation. Chen et al. [8] proposed a method for automated generation and evaluation of logical architectures. This method retrieves and combines components into feasible architecture solutions and selects the optimal one using multi-objective optimization. Gao et al. [9] proposed a set of domain metamodels and modeling patterns for communication satellite networks and optimized and improved the system architectures based on simulation. Fu et al. [10] imported SysML system design models into knowledge graphs to achieve model fusion and reuse and realized automated configuration of system architectures based on semantic retrieval. Cao et al. [11] proposed a software and physical synergetic architecture design method, in which the physical and software conceptual models are automatically generated from hybrid functional models based on ontological reasoning.

However, most of the architecture design methods in the MBSE originate from the top-down conceptual design methodology of engineering design, that is, the top-down functional decomposition structure is used as input, and the system architecture is generated from the bottom-up through component selection and integration [12]. However, these top-down architecture design methods are mainly suitable for design from scratch and hence cannot apply to the highly platform and product family-based design of RSS.

2.2 Architecture design in MBPLE

To further improve the reuse of existing design assets and shorten the delivery cycle, PLE has gradually attracted the attention of researchers in the MBSE field and formed a new research direction, MBPLE. Hugo et al. [13] divided the asset reuse strategy into three generations and gave the implementation framework of MBPLE using the Alstom Rolling Stock product line as an example. Young et al. [14] demonstrated the application of MBPLE method in Raytheon. The method defines various configurations using the PLE tool and transforms them into variant architecture models in the MBSE tool to support the subsequent design process. Bilic et al. [15] constructed a toolchain framework for integration of the vari-

ant management tools and MBSE modeling tools and verified its efficiency in the Volvo construction equipment product line. Tekinerdogan et al. [16] summarized the application experience and lessons of PLE in the field of physical protection systems (PPS) and proposed an MBPLE workflow suitable for the PPS product life cycle. Forlingieri [17] defined four dimensions of variability, namely co-variability in product, manufacturing and services, variability in the development lifecycle, variability in layers of abstraction, and variability in system hierarchy levels and analyzed their impact on the MBPLE process. Colletti et al. [18] proposed two variant modeling patterns and four comparison criteria to determine which pattern is best suited to model a certain product family. Forlingieri et al. [19] defined two variant modeling methods in SysML based on the application of MBPLE in Airbus. Arifin et al. [20] proposed a complete architecture design process combining MBPLE with genetic algorithms. This method generates a specific physical architecture model based on MBPLE configuration and uses a genetic algorithm to solve the component selection problem in this model. Wagner et al. [21] proposed a method to manage the variability inherent in application scenarios, system architecture, and the structure of the digital twin using MBPLE. Lameh et al. [22] integrated configuration management into feature models so that a versioned feature model is proposed. Wager et al. [23] extended the RFLP-framework by the viewpoint of variability so that variability models on different hierarchy levels can be linked to each other. Essoussi et al. [24] proposed an approach to migrate large legacy systems with variants designed in document-based ways to digital assets based on SysML extensions and plugins. Colletti et al. [25,26] reviewed recent work on variant management of simulation models and proposed an approach leveraging SysML and its variant modeling capabilities.

It can be seen that MBPLE is an emerging technology that has attracted extensive attention from researchers. However, existing methods mainly focus on how to apply MBPLE to individual design steps instead of streamlining the architecture design process. In addition, most of the work remains at the methodology level without computer-aided design tool support.

Since many domain-specific terms are introduced in MBPLE, extensions to the current general-purpose MBSE languages are required. Actually, in SysML v2, fundamental variability concepts have already been provided to facilitate model reuse at any level of design and maintenance of multiple variant configurations [27]. Epp et al. [28,29] analyzed the feasibility of using SysML v2 as a variant modeling language and proposed a variant modeling metamodel based on it. Besides the language architecture standardized by object management group (OMG) including unified modeling language (UML)/SysML, meta-object facility (MOF), etc., another important MBSE language is kombination of architecture model specificAtion (KARMA) [30,31], which provides a semantic specification to construct domain-specific metamodels and languages. Its main purpose is to formalize the semantics of various models to support their interoperability across the entire lifecycle. Since the proposed RSS-MBPLE approach is based on SysML v1.6, these advanced formalisms and languages can be investigated as our future work to migrate and adapt the approach to future demands.

3. Problem statement and method overview

With the rapid development of remote sensing technology, various types of RSS are invented to meet the needs of different missions. For example, they can be classified into visible light, infrared, laser, and microwave synthetic aperture radar (SAR) according to the payloads, agile and non-agile according to the working modes, and high-orbit, medium-orbit, and low-orbit according to the orbits. Although RSS types are diverse, the composition of the satellite platform is essentially the same. Different specific architectures are generated from this common platform by configuring the numbers, types, and specific standalone products to implement the components according to different missions and indicators [32]. The typical development process of RSS in practice is summarized in Fig. 1. This study mainly focuses on the architecture design of the solution development stage. The two core tasks of this activity are the architecture configuration and standalone product selection. RSS architecture configuration mainly forms the basic architecture according to different missions (such as working mode, orbital altitude, etc.), whereas standalone product selection is to compare and select concrete standalone products constituting the architecture according to multiple functional and non-functional requirements and indicators.

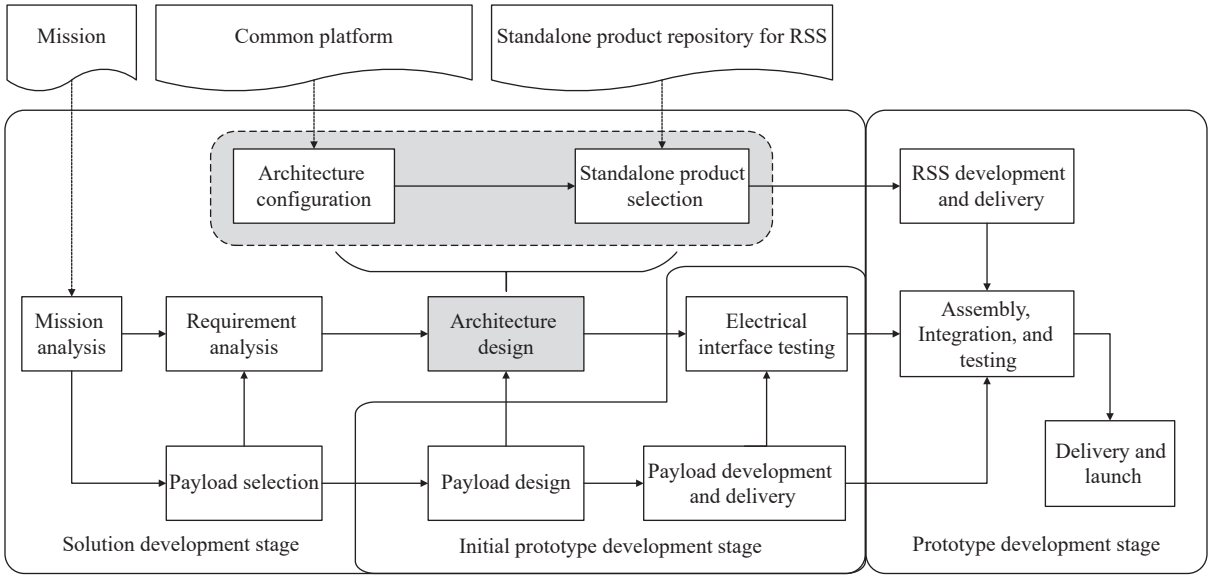


Fig. 1 Typical development process of RSS

Based on traditional MBSE methodologies, the architecture design should be conducted from scratch or reuse existing architecture design by "clone-and-modify". Either way is error-prone and inefficient. This issue can be solved by MBPLE, whose basic idea is shown in Fig. 2.

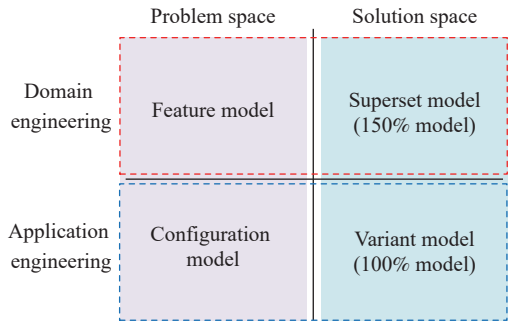


Fig. 2 MBPLE basic idea

The entire product family is defined as a superset model or a 150% model since it represents the product family whose scope is normally wider than a single product, i.e., 100% model [13]. It contains the common parts of all the system design models and identifies the variable parts as variation points (VPs). The specific values of the VPs are determined by the feature model, which describes the externally visible properties of the system. The relationships between the features and VPs are formally defined by impact rules. A set of specific values of the features in the feature model is so-called a configuration (in other words, a configuration is an instance of the feature model). Based on the impact rules between the feature model and the superset model, a configuration can uniquely generate a certain variant model, that is, a 100% model. The process of constructing feature models, superset models, and impact rules between them is called domain engineering and the process of generating variant models from configurations is called application engineering. The feature model and its configurations constitute the problem domain and the superset model and variant models constitute the solution domain.

Based on the analysis of the RSS development process in practice, this paper proposes an RSS-specific MBPLE approach. The approach is summarized in Fig. 3 and consists of the following two main parts.

(i) RSS-MBPLE metamodel

To represent the models involved in the RSS-specific MBPLE process, e.g., feature models, superset models, etc., domain-specific terminology should be formally defined as the domain-specific metamodel according to the domain metamodeling process. These RSS-MBPLE model elements can clearly identify the semantics of the models, which can facilitate both the engineers and computer programs to understand the models.

(ii) RSS-MBPLE process

Based on the typical RSS development process shown in Fig. 1, an RSS-specific MBPLE process to enable its architecture design is proposed, which covers the two core steps, i.e., architecture configuration and standalone product selection.

Step 1 Based on the typical missions and types of RSS, an architecture configuration space including the

architecture feature model, architecture superset model, and the impact rules between the features and VPs is constructed. On this basis, specific architecture variant models can be automatically generated according to the given specific feature configurations (e.g., low-orbit stable or high-orbit agile).

Step 2 Based on the generated architecture variant and the RSS model repository, the standalone product selection space including the solution feature model, solution superset model, and the impact rules between them is constructed. By calculating the indicators and comparing the feasible solutions that represent different combinations of the standalone products, the optimal solution can be selected and a specific solution variant model can be automatically generated through the solution selection space.

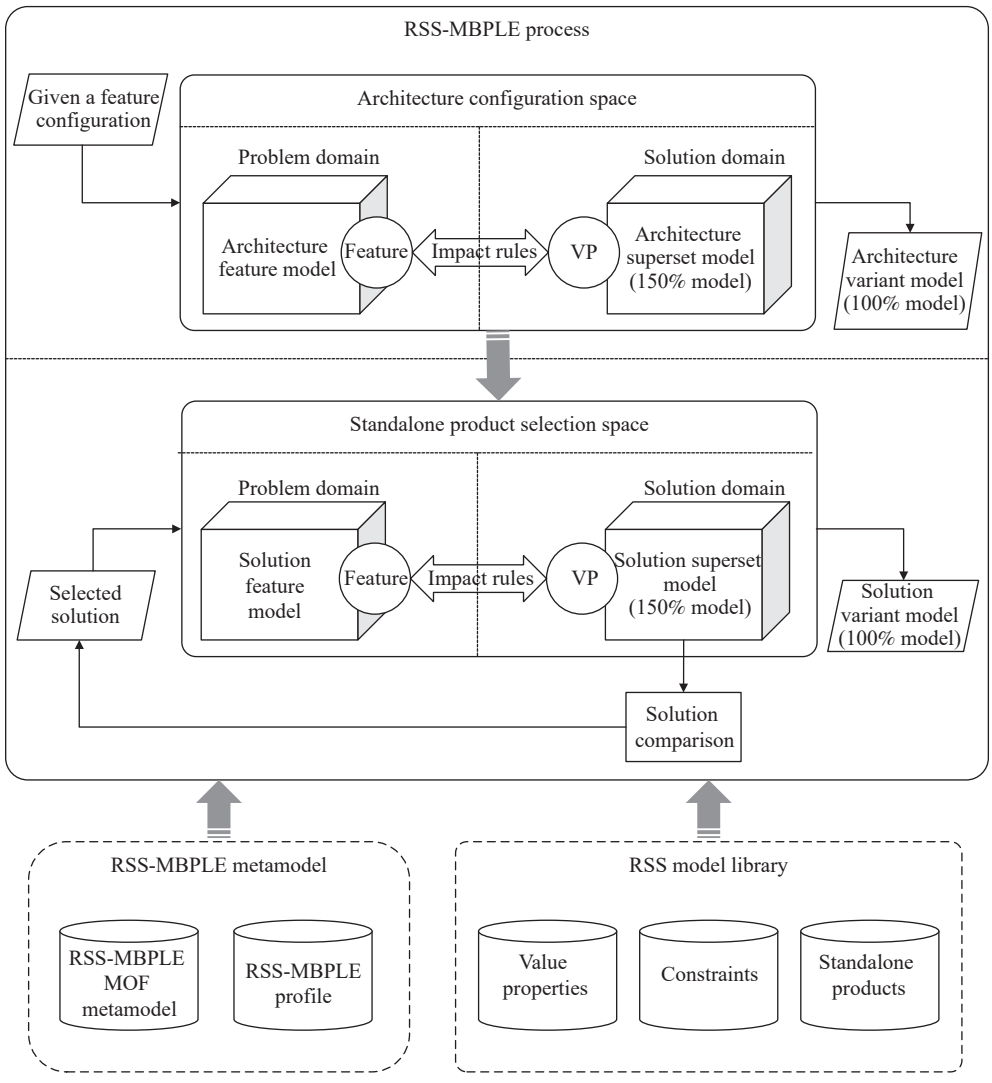


Fig. 3 Method overview

4. RSS-MBPLE metamodel

To describe the concepts in the RSS-specific MBPLE process, especially considering there are two separated MBPLE phases in the whole architecture design process, an RSS-MBPLE metamodel is proposed based on the domain metamodeling process [33] as shown in Fig. 4. In addition, to ensure the portability of the modeling elements among various mainstream SysML modeling platforms, the UML/SysML lightweight extension mechanism is leveraged to define the concrete model elements using stereotypes and profiles.

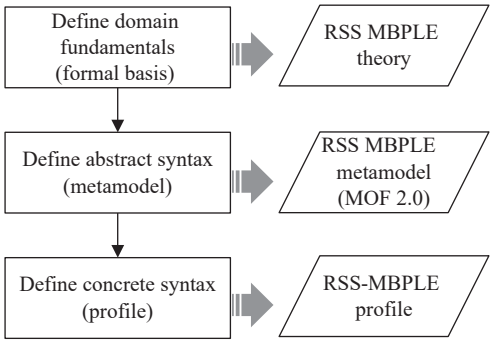


Fig. 4 Overview of the domain modeling process

The process consists of three steps: (i) Define domain fundamentals, which determines its formal foundation of the metamodel; (ii) Define abstract syntax, which defines the metaclasses and relationships using the meta-object facility (MOF 2.0); (iii) Define concrete syntax, which defines SysML stereotypes that can be directly used in the modeling tools by comparing and analyzing the similarities and differences of the syntax and semantics between the domain metamodel and UML/SysML metamodel.

Based on this process, an RSS-MBPLE metamodel is constructed and defined in MOF 2.0 as shown in Fig. 5. The metaclasses are detailed as follows.

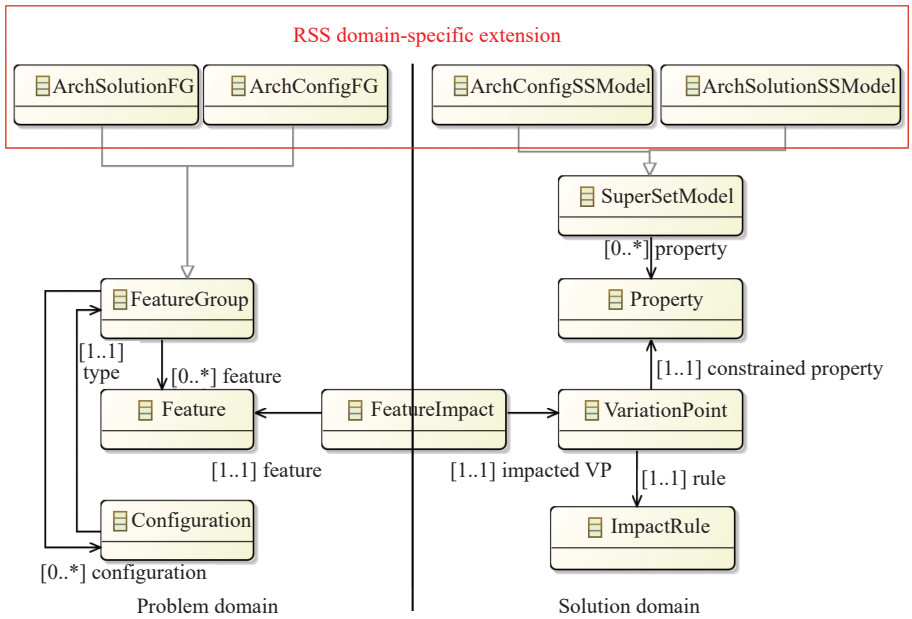


Fig. 5 RSS-MBPLE metamodel

Feature is the core metaclass of the feature model, which represents the externally visible properties of the system that stakeholders are interested in. A group of related features constitutes a FeatureGroup. Configuration is an instance of a feature group, which contains a set of specific values of features. The above-mentioned metaclasses together constitute the problem domain of MBPLE. The solution domain of MBPLE consists of superset models and variant points. SuperSetModel represents the full set of target models to be constructed. Since the superset model describes a model family containing a set of models, it is necessary to distinguish between the common parts and variable parts among these models. The variable parts of the superset models are noted by VariationPoint, which can be used to constrain any property of the superset model, indicating that the property will be set according to certain feature values. The impact relationships between the problem domain and the solution domain are represented by FeatureImpact, which connects the variant points and the features affecting them. How the feature values affect the properties constrained by the variant points is specified by ImpactRule.

Since MBPLE is used for two tasks in the RSS development process, i.e., architecture configuration and standalone product selection, it is important to extend the above-mentioned basic concepts to distinguish the two different problem-solution spaces. ArchConfigFG and ArchConfigSSModel represent the feature group and superset model for the RSS architecture configuration. The features in the feature group include features that affect the overall architecture of the satellite, such as orbital altitude and maneuverability. The superset model

describes the model family of different satellite architecture configurations, such as low-orbit stable and high-orbit agile. ArchSolutionFG and ArchSolutionSSModel represent the RSS standalone products selection feature group and superset model. The features in the feature group indicate the concrete products of the components in the architecture, such as the sun sensor products, star sensor products, etc. The superset model represents the model family of all specific product selection solutions.

Based on this metamodel, by extending UML/SysML model elements, corresponding stereotypes of all metaclasses in the metamodel can be constructed. These stereotypes constitute the RSS-MBPLE profile, which is shown in Fig. 6.

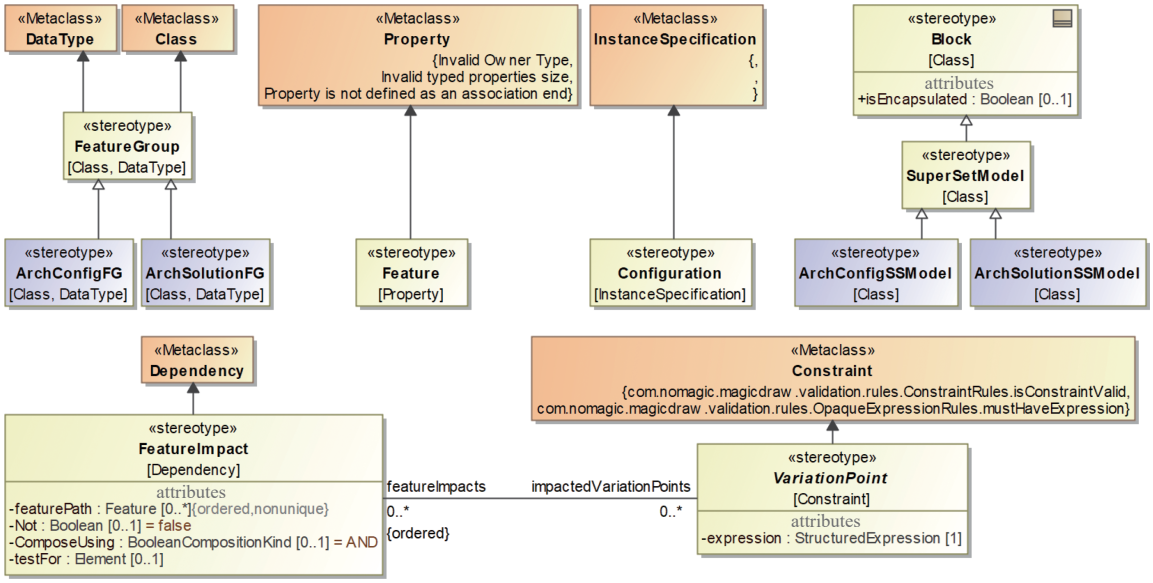


Fig. 6 RSS-MBPLE profile

5. RSS-MBPLE process

The two core steps for RSS architecture design in the typical development process shown in Fig. 3, i.e., RSS architecture configuration and standalone product selection are both supported by MBPLE to improve the design efficiency and maximize the reuse of existing design knowledge. The two phases are explained in detail in the following subsections.

5.1 Phase 1: architecture configuration

The process of the MBPLE-based architecture configuration is shown in Fig. 7. The process is divided into two major sub-processes, i.e., domain engineering and application engineering. Domain engineering mainly constructs feature models, superset models, and the impact rules between them, thereby providing a basis for the generation of architecture variants. During the application engineering, users can give concrete configurations according to specific tasks and then the architecture variants can be automatically generated through the variant generation. The following is a detailed description of the steps in the process.

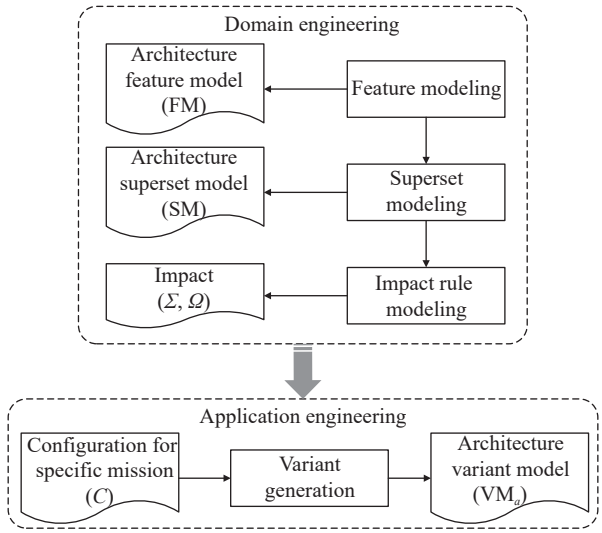


Fig. 7 Architecture configuration process

(i) Feature modeling. This step analyzes and abstracts the missions that RSS needs to achieve and builds a feature model based on the RSS-MBPLE profile. The architecture feature model $\mathrm{FM} = \{f_0, f_1, \cdots\}$ contains a set of features that can describe the missions of the RSS. Each feature $f = \langle \mathrm{name}, \mathrm{type} \rangle$ is described by its name and value type. The value type of a feature can be Boolean,

Integer, Real, Enumeration, etc.

(ii) Superset modeling. Superset models are built upon the general architecture of RSS platforms using RSS-MBPLE and RSS-arch profiles. The basic structure of the superset model $\mathrm{SM}=(M,\mathrm{VP},\Phi)$ is consistent with the usual SysML system model $M=\{e_0,e_1,\cdots\}$ which can be regarded as a set of model elements. The only difference between SM and M is that any model element in SM can be attached to a set of VPs $\mathrm{VP}=\{\mathrm{vp}_0,\mathrm{vp}_1,\cdots\}$ to indicate that it is a variable part of the model. $\Phi=\{(\mathrm{vp},e)|\mathrm{vp}\in\mathrm{VP}\wedge e\in M\}$ represents the constraint relationships between the VPs and the model elements.

(iii) Impact rule modeling. The impact rules describe how the properties of the model elements in the superset model are determined by the values of the features. They essentially reflect the design knowledge and experience of designers. The impact relationships $\Sigma=\{(\mathrm{vp},f)|\mathrm{vp}\in\mathrm{VP}\wedge f\in\mathrm{FM}\}$ associate the features with the variant points. Since the variant points are associated with the model elements in SM through the constraint relationships Φ, the feature values can determine the properties of the variable model elements in the superset model. The impact rules $\Omega=\mathrm{FM}\times\Sigma\rightarrow\mathrm{SM}$ describe how the features determine the model element properties in the superset model through the impact relationships.

(iv) Variant generation. Based on the models and impact rules constructed in the domain engineering, designers can set specific feature values according to specific missions. A configuration $C=\{c_0,c_1,\cdots\}$ is a set of feature values $c=\langle\mathrm{name},\mathrm{value}\rangle$. The architecture variant model VM_a can be automatically generated from the configurations by setting the values of the variable parts of SM according to the impact rules. This automated generation is supported by MBPLE tools.

5.2 Phase 2: standalone product selection

The architecture model of RSS generated above only describes the types, numbers, and connection relationships of the system components but does not specify the concrete standalone products used in each component. Therefore, it is necessary to generate different combinations of the standalone products from the architecture variant model and compare and select the optimal one according to the system indicators. One combination of the standalone products is the so-called solution. The process of solution selection based on MBPLE is shown in Fig. 8. Since it also follows the MBPLE idea, the overall workflow is similar to Fig. 7.

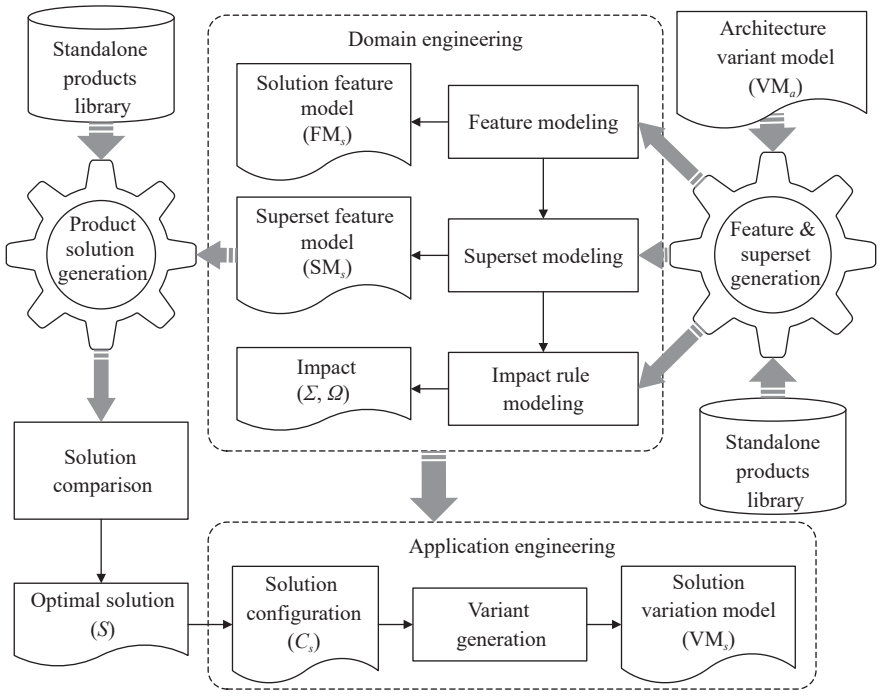


Fig. 8 Solution selection process

The process contains four major activities.

Step 1 Feature and superset generation. The designers build a solution feature model and the superset model based on the architecture variant model and the standalone product models in the reusable model repository;

Step 2 Product solution generation. The designers generate multiple feasible solutions based on the standalone products in the repository and calculate the indica-

tors of each solution;

Step 3 Solution comparison. Through the solution comparison based on multi-indicators, the optimal solution can be selected by designers manually;

Step 4 Variant generation. The solution configuration is generated based on the optimal solution and the variant model of the final product design solution is automatically generated through the variant generation step.

Thanks to the aforementioned architecture variant model and the standalone products model repository, Step 1 and Step 2 can be automatically completed through interactive algorithms, which are explained in detail in the following sections. And since Step 4 is already implemented by MBPLE tools, the whole workflow is highly automated and efficient.

5.2.1 Automated generation of feature and superset models

The algorithm for automatically generating the product selection space for solution selection including the solution feature models, superset models, and impacts between them is shown in Algorithm 1. The input of the algorithm is the architecture variant model VM_a generated during the architecture configuration and the reusable standalone product model repository, and the output is the product solution selection feature model FM_s, the superset model SM_s, and the impact relationship Σ_s between them. Since the main task of product solution selection is to select the concrete products realizing the components in the architecture variant model, the algorithm (Lines 5−9) first creates SM_s based on VM_a. The VPs in SM_s are all components whose concrete standalone products have not been determined yet and their types are set as generic types so far. In the variant models, the specific subclasses of the generic types will be selected as the final types of the components. Users can also split components based on specific needs. For example, the gyroscope can be split into a mechanical gyroscope and fiber optic gyroscope to support independent selection of products. After that, the feature model FM_s is generated based on the superset model and the reusable standalone product repository. Each feature in FM_s controls the product selection corresponding to the VPs in SM_s.

Algorithm 1 Component selection space generation algorithm

1 **Input** the architecture variant model VM_a, the component model repository Lib
2 **Output** the feature model FM_s, super set model SM_s for product selection, and impact rules Σ_s between them.
3 **foreach** component e_a=(name, type, multiplicity) in VM_a **do**
4 // populate SM_s by copying VM_a
5 clone $e_s = e_a$
6 add vp_s to e_s
7 add e_s to SM_s
8 (users can split certain c_s to multiple components on demand)
9 // populate FM_s by creating features whose values are from the component repository
10 **foreach** e_s in SM_s **do**
11 create $f_s = (e_s.\text{name}, t_s)$ in which t_s is an Enumeration
12 **foreach** e in Lib **do**
13 **if** $e == e_s.\text{type}$ **then** add all leaf types t_l of e as literals of t_s
14 add f_s to FM_s
15 //create impact rules
16 create ir = $(e_s.vp_s, f_s)$
17 add ir to Σ_s
18 **foreach** l in t_s **do**
19 create con = $\{\text{if } f_s = l \Diamond e_s.\text{type} = t_l\}$
20 add con to $e_s.vp_s$

Therefore, $f_s \in FM_s$ and $e_s \in SM_s$ have a one-to-one correspondence and the value range of f_s is all optional products of component e_s in the model repository. Based on the above rules, f_s can be created (Lines 14−17). The mapping rule creation process between f_s and the VP vp_s of e_s is very intuitive, that is, any value of f_s determines the corresponding product type of e_s (Lines 18−23). Based on the solution selection space generated by this algorithm, the selection of specific products in the superset model is directly controlled by feature values, thereby enabling the fast generation of a specific solution model.

5.2.2 Automated generation of feasible product solutions

The algorithm for automatically generating optional product solutions is shown in Algorithm 2. The input of the algorithm is the superset model SM_s of the solution selection space, the standalone product model repository Lib, and the condition set F that is used as the filters to the feasible standalone products. The output is the set of all feasible solutions noted as PPS. The algorithm (Lines 5−13) first searches for all standalone products p_i of all e_s with VPs in SM_s in the model repository Lib and adds them to P_c, which contains the feasible standalone products of all elements with VPs. After that, by calculating the Cartesian product of all optional sets of components in P, the set PSS containing all the optional product solutions is generated (Lines 15−17).

Algorithm 2 Product solutions generation algorithm

1 **Input** the superset model SM_s for product selection, the component model repository Lib, the filter of properties F

2 **Output** the set of product solutions PSS

3 create $P = \varnothing$ // the set of all the possible products of each component which has an VP

4 // populate P by scanning the model repository

5 **foreach** component e_s in SM_s and e_s.vp_s is not null **do**

6 create $P_c = \varnothing$ // the set of all products of component e_s in Lib

7 **foreach** e in Lib **do**

8 **if** $e == e_s$.type **then**

9 **foreach** p which is a descendant of e **do**

10 **if** p is a leaf and p satisfies F **then** add p to P_c

11 add P_c to P

12 //populate PSS by combining all the products in P

13 **foreach** P_c in P **do**

14 PSS = PSS × P_c

6. Case study

In this section, a typical RSS control subsystem is used as an example to illustrate the proposed method. This method is implemented based on a mainstream MBSE platform Cameo System Modeler 19.0 and its plugin MBPLE.

6.1 RSS control subsystem

The RSS control subsystem is a typical complex system. It is mainly involved in the attitude and orbit control of the entire satellite. It has complex structures and functions, fixed common architectures, and rich specific configurations [34]. It works with the propulsion subsystem to complete the attitude and orbit control of the satellite in various working modes during the on-orbit period and other related tasks in accordance with the payload working requirements. The general configuration of the RSS control subsystem is shown in Fig. 9. It is mainly composed of three parts, i.e., controllers, sensors, and actuators. The identifier after the name of each component indicates the number of configurable units. For example, 0-1 Sun Sensor[0-*] means there can be none or many 0-1 Sun Sensors in the system.

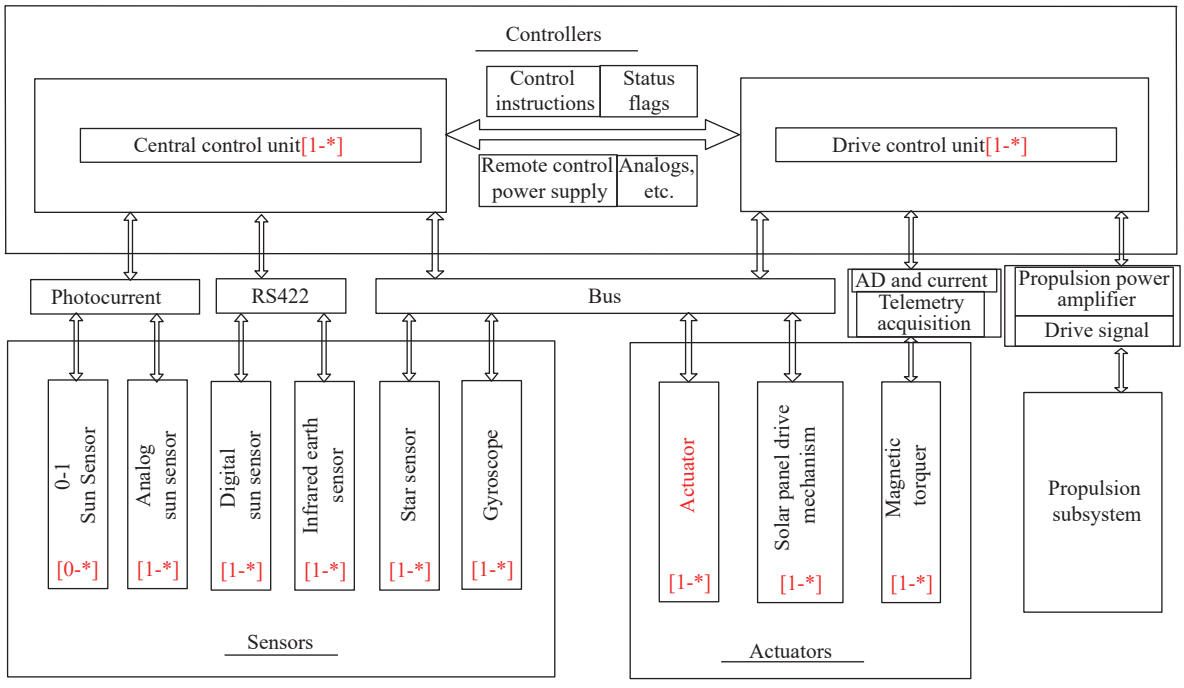


Fig. 9 General architecture of the control subsystem

Based on the general architecture, as shown in Table 1, according to specific application scenarios of RSS, they can be divided into several typical configurations, such as low-orbit stable, low-orbit agile, high-medium-orbit stable, and high-medium-low-orbit agile. Taking the low-orbit stable RSS as an example, its typical configuration is shown in Fig. 10. The typical configuration of sensors is star sensor×3, gyroscope×3, digital sensor×1, analog sensor×5, infrared

earth sensor×2; the typical configuration of actuators is momentum wheel×5, solar panel drive mechanism×2, magnetic torquer×3; the typical configuration of controllers is central control unit×1 and drive control unit×1.

Table 1 Typical configurations of RSS

Typical configuration	Application description
Low orbit stable	Applicable to satellites working in low orbit in steady state or with general attitude maneuvering/side swinging capability
Low orbit agile	Applicable to satellites with high requirements for low orbit attitude maneuvering capability
Medium and high orbit stable	Applicable to satellites working in medium and high orbit in steady state or with general attitude maneuvering/side swinging capability
Medium, high, and low orbit agile	Applicable to medium, high, and low orbits, with high system integration, high-precision measurement, pointing, and rapid maneuvering capability can be achieved through the optional three-supervision platform, achieving comprehensive performance improvement

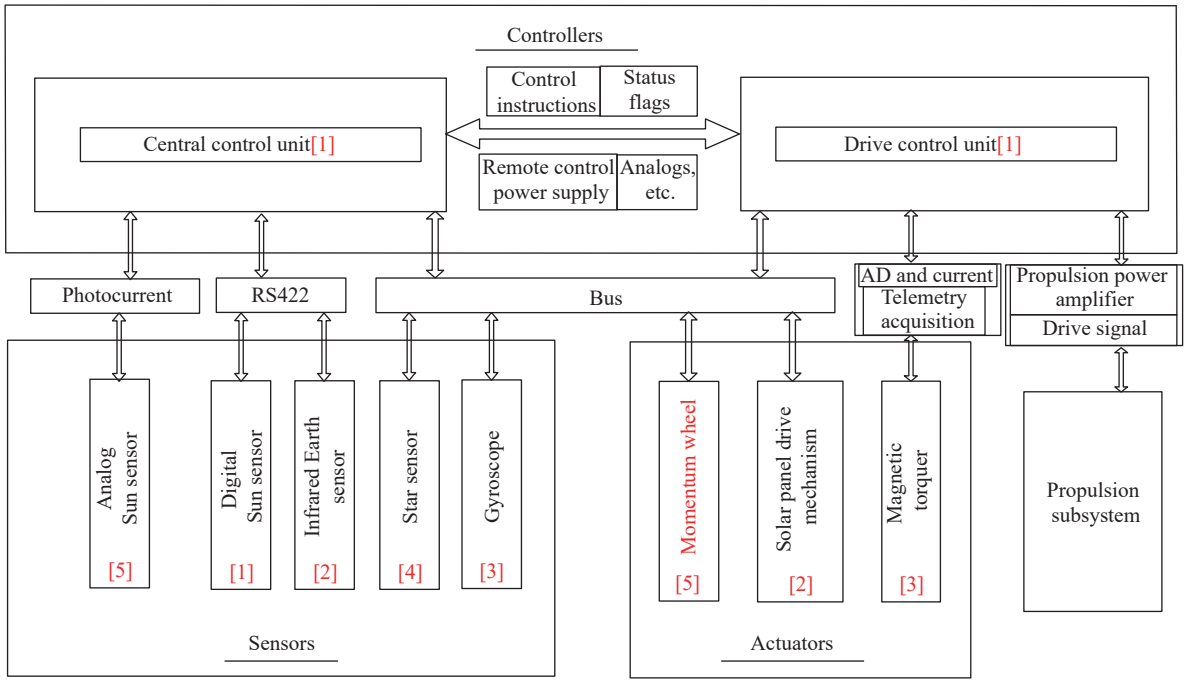


Fig. 10 Typical configuration of low-orbit stable RSS control subsystem

After determining the architecture configuration of the control subsystem, it is also necessary to select a concrete standalone product for each component. For example, Table 2 shows a list of available star sensor products for low-orbit RSS.

Table 2 List of available star sensor products

Number	Name	Product technical status	Application
1	High-precision star sensor APS VI type	Measurement accuracy (3σ): 3″(x/y), 25″(z) Optical axis thermal drift better than 0.3″/℃ Stray light suppression angle: 35° Dynamic performance: 3°/s Data update rate: 12 Hz Weight: probe 800g; line box 700g Power consumption: probe 1.8 W; line box 5W Power supply: 20−50 V, relay mode configurable Data interface: RS422, 1 553	Applicable to low, medium, and high orbits
2	High-precision star sensor APS II type	Measurement accuracy (3σ): When the absolute value of angular velocity is less than 0.1°/s, single head: x/y≤1"; z≤30" Two probes: When the installation angle is 90°, ≤0.7"(x/y); ≤0.7"(z) When the installation angle is 60°, ≤0.9"(x/y); ≤1.0"(z) Three probes: When the installation angle is 90°, ≤0.6"(x/y); ≤0.6"(z) When the installation angle is 60°, ≤0.8"(x/y); ≤1.0"(z) Optical axis thermal drift is better than 0.3″/℃ (15℃−25℃) Sunlight suppression angle 30° Earth-air light suppression angle 30° Dynamic performance: 2°/s Data update rate: 8 Hz Weight: Probe ≤3.7 kg, circuit box ≤7 kg Dimensions: Probe 156 mm×170 mm×392 mm Line box 144 mm×196 mm×258.5 mm Power consumption: Probe ≤4 W, Line box ≤45 W. Data interface: 1553B interface Power supply: +28−+42 V	Applicable to low, medium, and high orbits

Continued

Number	Name	Product technical status	Application
3	Ultra-high precision star sensor APS I type	Field of view (°): 2×2; Sensitive magnitude (Mv): +10.5; Optical axis pointing accuracy (3σ): 0.3"; Update rate (Hz): 8; Dynamic performance (°/s): 0.2; Working temperature (℃): −30−+40; Sunlight suppression angle (°): 35; Body size (mm×mm×mm): Probe: 360×360 ×655; Line: 190×110 ×157; Weight (kg): Probe: to be determined; Line: to be determined; Electrical interface: RS422/1553B/LVDS; Power consumption (W): Probe: 2; Line: 10; Design life (a): 15;	Applicable to low, medium, and high orbits
4	Ultra-high precision star sensor APS II	Measurement accuracy (3σ): When angular velocity ≤0.1/s, better than 0.5″ (*X*/*Y*); Angular velocity 0.1−1 /s, better than 1″ (*X*/*Y*) Angular velocity 1−2/s, better than 2″ (*X*/*Y*) Angular velocity 2−5/s, better than 15″ (*X*/*Y*) Optical axis thermal drift is better than 0.1″/℃ (15℃−25℃) Sunlight suppression angle 30° Earth-air light suppression angle 30° Dynamic performance: 5°/s Data update rate: 8 Hz Weight: probe ≤5 kg, circuit box ≤10 kg Dimensions: probe 156 mm×170 mm×392 mm Circuit box 144 mm×196 mm×258.5 mm Power consumption: probe ≤4 W, circuit box ≤45 W. Data interface: 1553B interface Power supply: +28−+42 V	Applicable to low, medium orbits

The basis for selecting a standalone product is the typical indicators of the control subsystem, such as weight, attitude determination accuracy, attitude pointing accuracy, maneuverability, and stability. The decomposition of each system indicator is shown in Fig. 11.

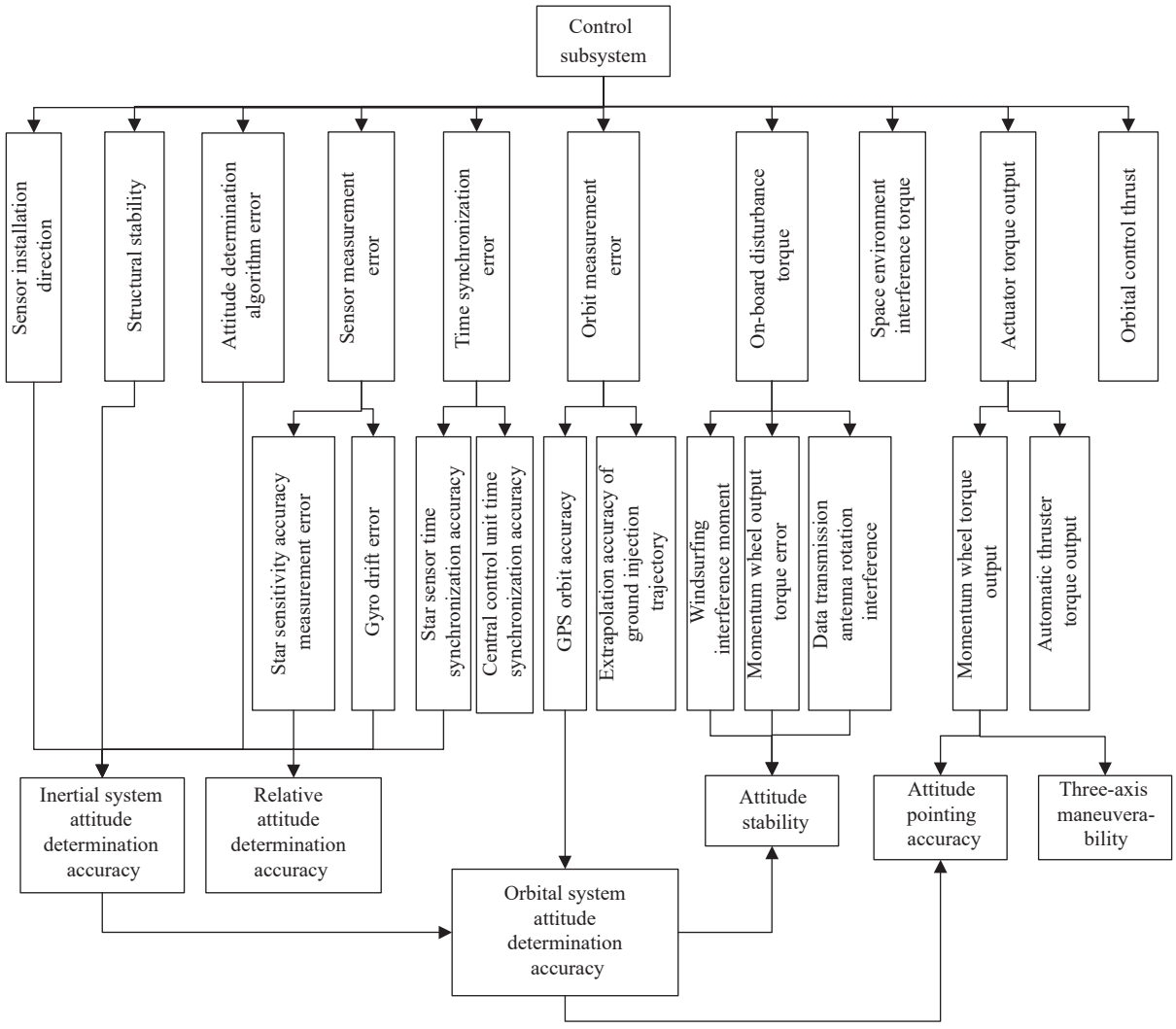


Fig. 11 Decomposition of control subsystem indicators

It can be seen that the system indicators are mainly impacted by the indicators of the standalone products. Therefore, using the decomposition relationship and control theory, the mathematical relationships between the system-level indicators and the standalone product indicators can be given. Based on these mathematical equations, the indicators of various product design solutions can be calculated and compared so that the optimal one can be selected.

From the analysis of the RSS control subsystem, it can be seen that it has the characteristics of complex architecture and diverse constraints. Considering it has a strong inheritance but rich specific configurations, it is suitable to leverage the MBPLE principles to support its architecture design. The RSS-MBPLE approach including two phases is illustrated in the following two sections. The SysML platform used in the case study is Cameo System Modeler 19.0. The two algorithms are also implemented as plugins to be seamlessly embedded in the design process.

6.2 Architecture configuration

The purpose of the control subsystem architecture configuration is to generate architecture variants based on the typical configurations of the control subsystem according to the missions, e.g., the low-orbit stable RSS architecture. The process is shown in Fig. 12.

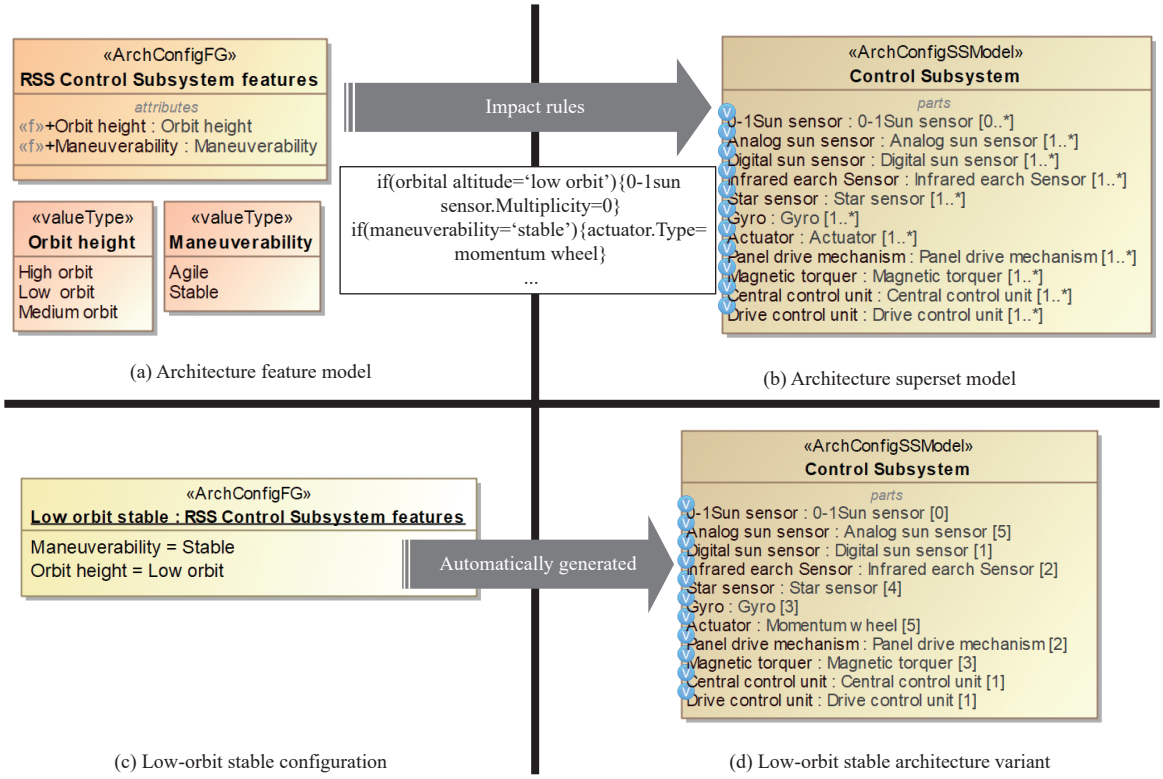


Fig. 12 Control subsystem architecture configuration process

The steps of architecture configuration are as follows:

(i) According to the missions of the RSS, extract the two features, i.e., orbital altitude and maneuverability and construct the feature model shown in Fig. 12(a);

(ii) Construct the superset model based on the general configuration of the control subsystem shown in Fig. 9 and create variant points according to the variable parts as shown in Fig. 12(b);

(iii) Establish the impact relationship and impact rules between features and variant points. For example, the rule if(orbitalaltitude='loworbit'){0−1solarsensor.Multiplicity=0} means that when the orbital altitude in the feature model is low orbit, the number of 0-1 solar sensors in the architecture variant is 0; the rule if(maneuverability{=} 'stable'){actuator.Type=momentumwheel} means that when the maneuverability in the feature model is stable, the actuator type in the architecture variant is momentum wheel.

(iv) Since the above process establishes the impact

rules between the feature model and the architecture superset model, when a specific configuration is given, the corresponding architecture variant can be automatically generated following the rules. For example, given the low-orbit stable configuration in Fig. 12(c), the architecture variant shown in Fig. 12(d) can be automatically generated.

6.3 Standalone product selection

Based on the low-orbit stable architecture, designers can make specific selections for the standalone products from the standalone product model repository to generate the final product design solution.

(i) Automatic generation of feature and superset models for product selection

Since the standalone product selection still adopts the MBPLE concept, it is necessary to construct the feature model and superset model for this process. The feature and superset models can be automatically generated following Algorithm 1. The whole process is illustrated in Fig. 13. The input of the algorithm is the low-orbit stable architecture variant model shown in Fig. 12(d) and the standalone product model and constraint repository. They are also illustrated in the upper part of Fig. 13.

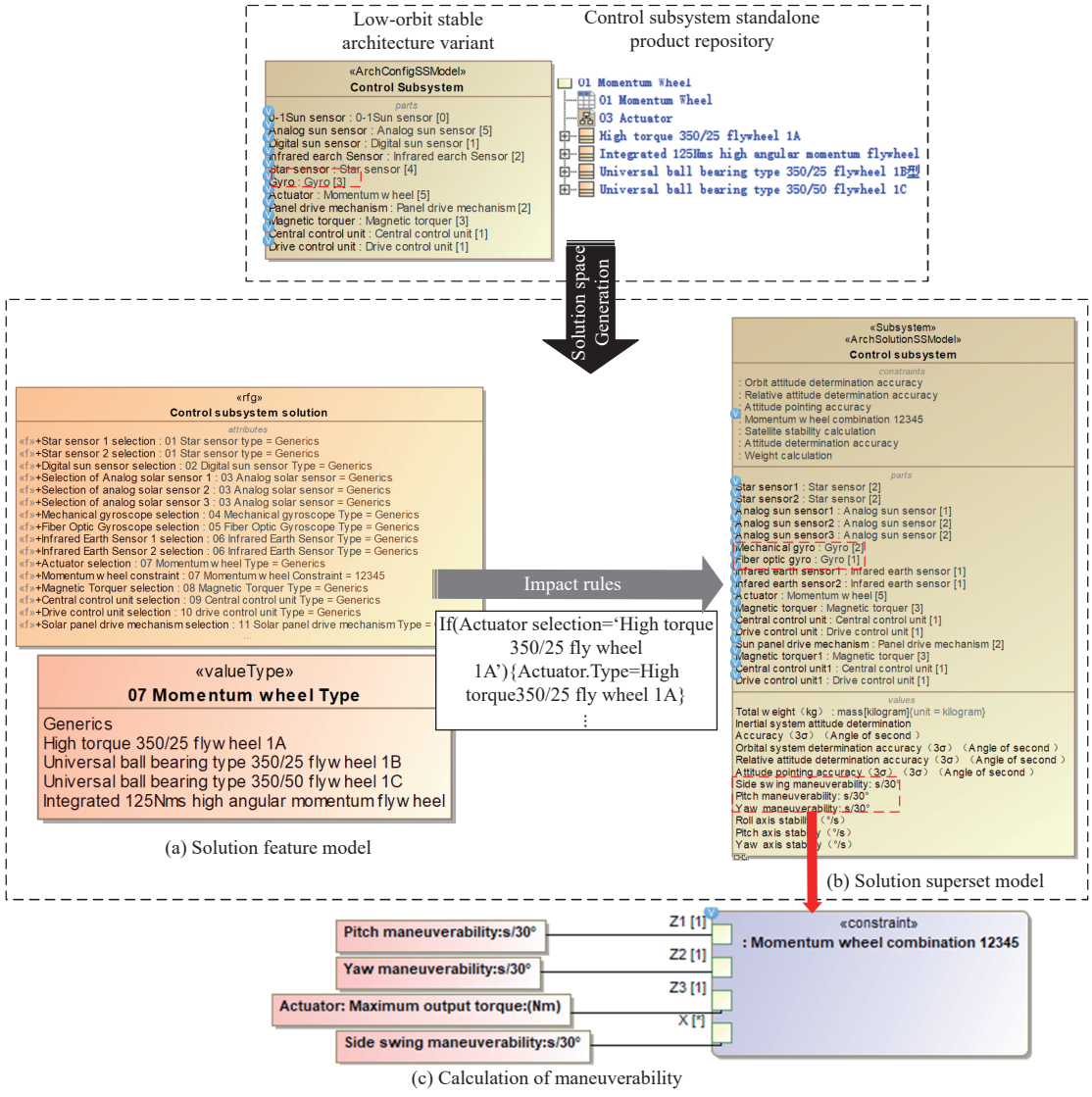


Fig. 13 Control subsystem solution feature model and superset model

The first output model is the Solution superset model as shown in Fig. 13(b). It is identical to the architecture variant model except that some components whose multiplicity is greater than 1 are split into multiple compo-

nents since they need to be realized by different standalone products. For example, Gyro whose multiplicity equals 3 is split into two components, i.e., Fiber optic gyro and Mechanical gyro to support the independent selection of the two. In addition, constraints (such as momentum wheel combination 12345, etc.) and indicators (such as pitch maneuverability, yaw maneuverability, and roll maneuverability, etc.) are manually added to the superset model. The final solution superset model contains 17 VPs, used to support the selection of different components and constraints.

The second output is the Solution feature model as shown in Fig. 13(a). The features have one-to-one correspondences with the VPs in the superset model. The possible values of each feature are generated by scanning the standalone product repository. For example, there are four products that can implement Momentum wheel, e.g., High torque 350/25 fly wheel 1A, Integrated 125 Nms high angular momentum flywheel, etc. They constitute an enumeration 07 Momentum wheel Type, which is set as the type of the feature Actuator selection.

The third output is the impact rules between features and VPs. They simply reflect the mappings between feature values and standalone products, e.g., If(Actuator selection='High torque 350/25 fly wheel 1A'){Actuator. Type=High torque 350/25 fly wheel 1A}.

(ii) Product solution generation and comparison

Following Algorithm 2, by inputting the Solution superset model shown in Fig. 13(b) and the standalone product repository, all the feasible standalone product combinations can be automatically generated. They are represented as instances of the solution superset model. Fig. 14 shows five solutions with parameters.

#	Name	: Momentum wheel combination 12345	Magnetic torquer : 200An2Magneti torque devic	Total weight (kg) : mass [kilogram]	Inertial system attitude determinatic Accuracy (3σ) (Angle of second)	Orbital system determination accuracy (3σ) (Angle of second)	Relative attitude determination accuracy (3σ) (Angle of second)	Attitude pointing accuracy (3σ) (3σ) (Angle of second)	Side swing maneuverability: s/30°	Pitch maneuverability: s/30°	Yaw maneuverability: s/30°	Roll axis stability (° /s)	Pitch axis stability (° /s)	Yaw axis stability (° /s)
1	Solution1	Momentum whe	: 100An2Magn : 100An2Magn : 100An2Magn	166.38 kg	0.7769	4.0747	0.002	0.0757	307.4771	316.6221	303.8597	0.0002	0.0003	0.0003
2	Solution2	Momentum whe	: 200An2Magn : 200An2Magn : 200An2Magn	224.62 kg	4.6614	6.1424	0.002	0.0757	251.054	265.0509	268.459	0.0002	0.0003	0.0003
3	Solution4	Momentum whe	: 200An2Magn : 200An2Magn : 200An2Magn	205.43 kg	0.3574	4.0159	0.002	0.0757	251.054	265.0509	268.459	0.0002	0.0003	0.0003
4	Solution5	Momentum whe	: 200An2Magn : 200An2Magn : 200An2Magn	114.67 kg	0.9323	4.1072	1.3333	0.0757	307.4771	265.0663	285.276	0.0034	0.0035	0.0035
5	Solution3	Momentum whe	: 100An2Magn : 100An2Magn : 100An2Magn	133.01 kg	0.4661	4.0271	3.6667	0.0757	307.4771	316.6221	303.8597	0.001	0.0011	0.0011

Fig. 14 Five solutions of the control subsystem

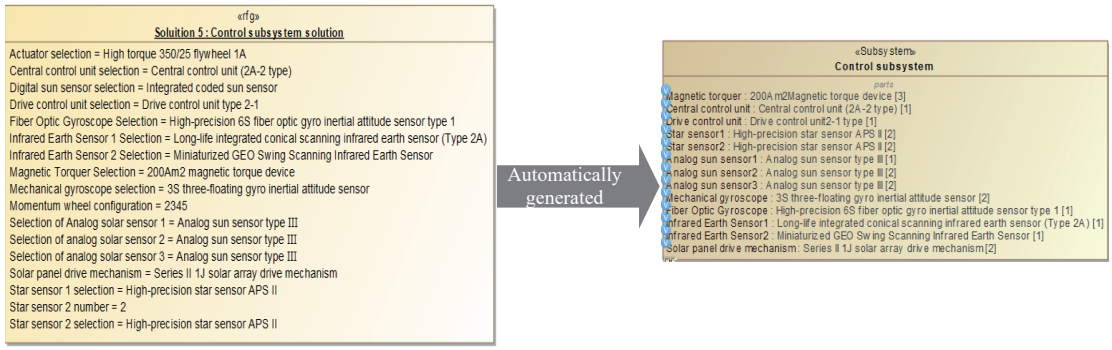
The key indicators of the solutions can be calculated according to the parameters of standalone products. This calculation is automated by running the instance table against the parametric diagrams defining the calculation formulas as shown in Fig. 13(c). As shown in Fig. 15, the key indicators of various solutions are calculated. By comparing the indicators, Solution 5 is selected as the optimal solution.

#	Name	Total weight (kg) mass [kilogram]	Inertial system attitude determination Accuracy (3σ) (Angle of second)	Orbital system determination accuracy (3σ) (Angle of second)	Relative attitude determination accuracy (3σ) (Angle of second)	Attitude pointing accuracy (3σ) (Angle of second)	Side swing maneuverability: s/30°	Pitch maneuverability: s/30°	Yaw maneuverability: s/30°	Roll axis stability (°/s)	Pitch axis stability (°/s)	Yaw axis stability (°/s)
1	Solution1	166.38 kg	0.7769	4.0747	0.002	0.0757	307.4771	316.6221	303.8597	0.0002	0.0003	0.0003
2	Solution2	224.62 kg	4.6614	6.1424	0.002	0.0757	251.054	265.0509	268.459	0.0002	0.0003	0.0003
3	Solution4	205.43 kg	0.3574	4.0159	0.002	0.0757	251.054	265.0509	268.459	0.0002	0.0003	0.0003
4	Solution5	114.67 kg	0.9323	4.1072	1.3333	0.0757	307.4771	265.0663	285.276	0.0034	0.0035	0.0035
5	Solution3	133.01 kg	0.4661	4.0271	3.6667	0.0757	307.4771	316.6221	303.8597	0.001	0.0011	0.0011

Fig. 15 Comparison of indicators of the five solutions

(iii) Solution variant generation

After obtaining the optimal solution, i.e., Solution 5, the feature model can be instantiated and configured according to this solution as shown in Fig. 16(a). According to this configuration, the solution superset model can be transformed into a specific solution variant corresponding to Solution 5 as shown in Fig. 16(b).



(a) Solution 5 configuration (b) Solution 5 architecture

Fig. 16 Configuration and solution variant of Solution 5

7. Discussions

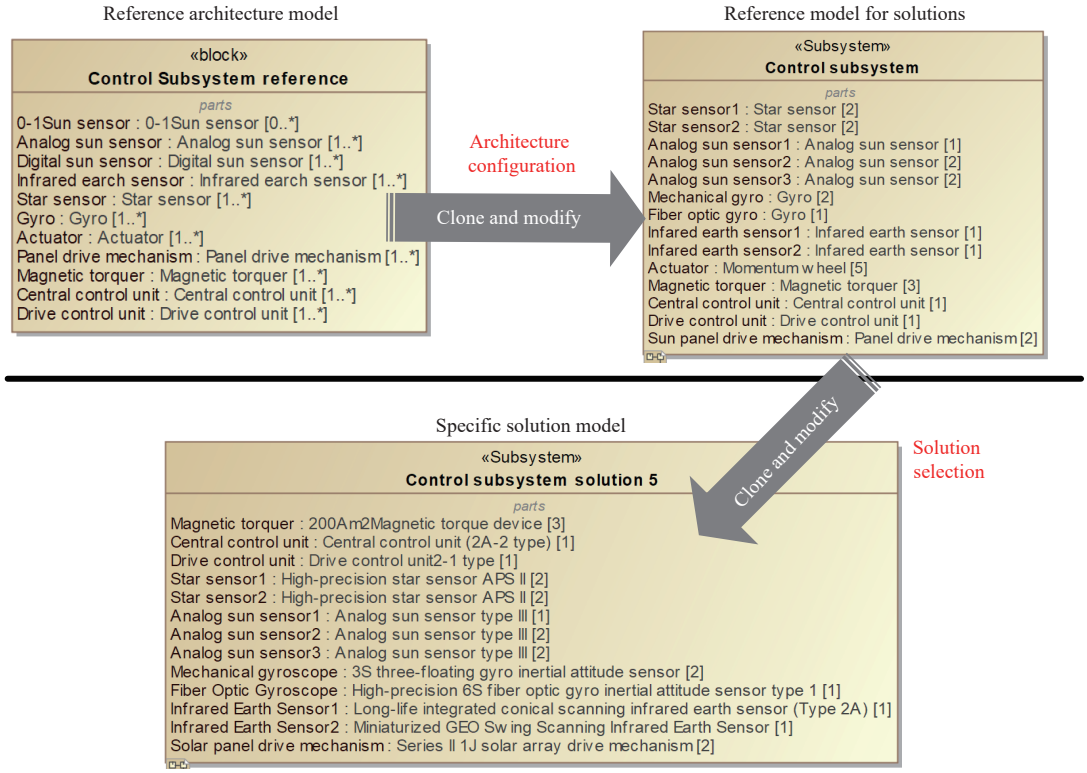
To demonstrate the advantages of the proposed RSS-MBPLE approach over the traditional MBSE methodology, Fig. 17 shows the comparison between the two workflows.

It can be seen that the traditional MBSE methodology shown in Fig. 17(a) heavily relies on the manual action "clone-and-modify" due to the lack of efficient architecture-level reusability. Specifically, to create a specific architecture for low-orbit stable satellites, the designers have to clone the existing reference architecture model and then modify it manually according to their design knowledge, e.g., the low-orbit stable satellite should have five analog sun sensors and three gyros, the gyros should be split to two separate components to hold different standalone products. Then, after the concrete standalone products are determined, the specific solution model should be cloned and modified from the reference model.

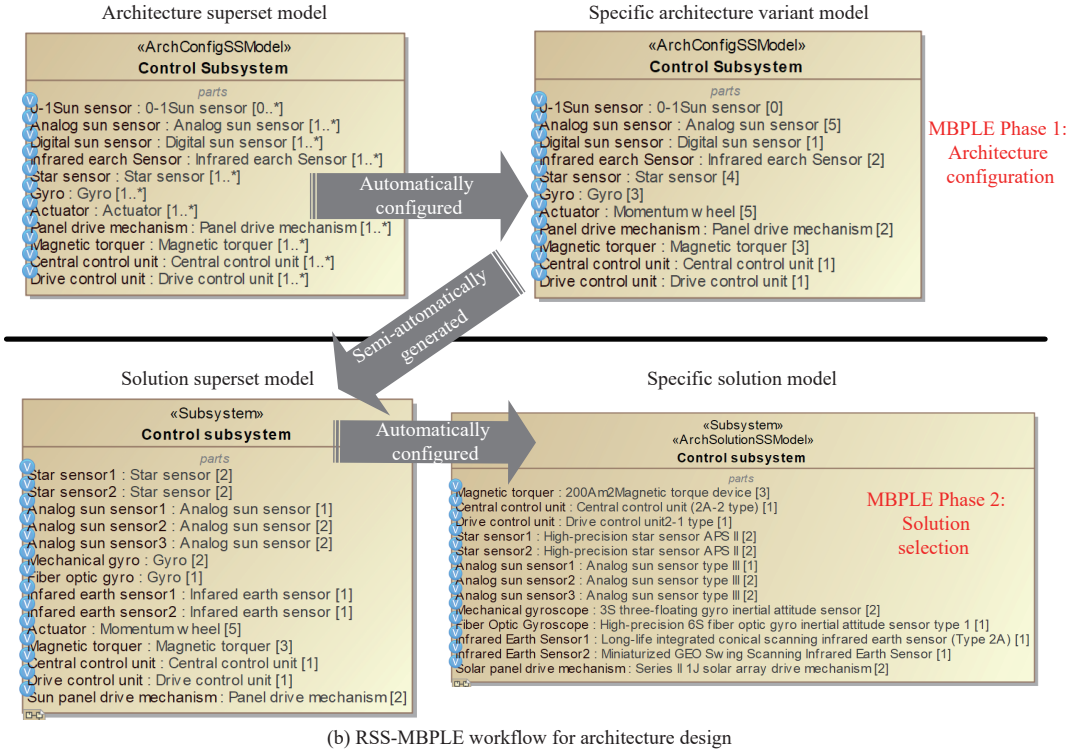
On the contrary, the RSS-MBPLE replaces the "clone-and-modify" actions by design automation. In MBPLE phase 1, the low-orbit stable architecture variant can be automatically generated from the superset model based on the configurations of the simple and user-visible external features. Then, the solution superset model can be generated semi-automatically by Algorithm 1 shown in Subsection 5.2.1. Finally, the specific solution model can be automatically generated from this superset model through feature configuration again in MBPLE phase 2.

From the comparison, it can be seen that the proposed RSS-MBPLE can dramatically improve the design efficiency. The main enablers are the reusable design knowledge consolidated in the MBPLE core assets. Unlike traditional MBSE, which only supports component-level reusability based on component repositories, the reusable assets in MBPLE are feature models, superset models, and impact rules between them, which enable architecture-level reusability. Take the design knowledge "the low-orbit stable satellite should have five analog sun sensors and three gyros" as an example, this knowledge no longer stays in designers' minds but is explicitly defined as impact rules between the features "Orbit height" and "Maneuverability" and the components "analog sun sensor" and "gyro". Thanks to the formally defined design knowledge, automation tools and algorithms can be developed to enable design automation and eliminate manual effort.

Another observation is that from Fig. 17(b), it can be seen clearly that the proposed RSS-MBPLE workflow contains two phases. Moreover, the two phases are closely related because the feature and superset models of Phase 2 are determined by the architecture variant generated in Phase 1. Such a relationship makes the manual construction of the problem and solution space of MBPLE Phase 2 tedious and non-feasible. This issue was discussed in Subsection 5.2 and the corresponding automation algorithms are proposed, which demonstrates the advantage of the proposed RSS-MBPLE approach over existing MBPLE methods.



(a) Traditional MBSE architecture design workflow



(b) RSS-MBPLE workflow for architecture design

Fig. 17 Comparison between traditional MBSE and RSS-MBPLE workflows

8. Conclusions

RSS is a typical complex system whose architecture design is challenging. By observing the characteristics of common platforms with various configurations specific to different missions, the MBPLE methodology can be

leveraged to support its architecture design. Through the analysis of the development process of RSS in practice, an RSs-MBPLE approach is proposed. The main contributions of this study can be summarized as follows.

(i) The RSS-MBPLE profile is defined to represent the RSS-specific MBPLE concepts in SysML. Based on this profile, the semantics of the models involved in the RSS-MBPLE process can be explicitly indicated, which can be understood by both human-beings and computer programs.

(ii) A practical tool-supported RSS-MBPLE process is proposed, which can be seamlessly integrated into the RSS development process in practice. The RSS-MBPLE process includes two phases to accomplish the two key tasks of architecture design, i.e., architecture configuration and standalone product selection with high quality and efficiency.

The proposed method is implemented in a mainstream SysML platform with plugins. A case study of the low-orbit stable RSS control subsystem has demonstrated the efficiency of the method.

Future works include two folds. The domain-specific modeling profile and model repository can be further enriched by analyzing more types of RSS. The degree of automation and intelligence of the MBPLE-enabled method can be improved from various aspects such as intelligent learning and generation of feature models and automated selection of optimal solutions.

References

[1] RAMOS A L, FERREIRA J V, BARCELÓ J. Model-based systems engineering: an emerging approach for modern systems. IEEE Trans. on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2011, 42(1): 101–111.

[2] MADNI A M, SIEVERS M. Model-based systems engineering: motivation, current status, and research opportunities. Systems Engineering, 2018, 21(3): 172–190.

[3] KANG K C, LEE J, DONOHOE P. Feature-oriented product line engineering. IEEE Software, 2002, 19(4): 58–65.

[4] HAUSE M, HUMMELL J. Model-based product line engineering–enabling product families with variants. INCOSE Insight, 2019, 22(2): 43–48.

[5] LYKINS H, FRIEDENTHAL S, MEILICH A. Adapting UML for an object oriented systems engineering method (OOSEM). INCOSE International Symposium, 2000, 10(1): 490–497.

[6] DOUGLASS B P. Agile systems engineering[M]. Waltham: Morgan Kaufmann, 2015.

[7] MORKEVICIUS A, ALEKSANDRAVICIENE A, et al. Towards a aommon systems engineering methodology to cover a complete system development process. INCOSE International Symposium, 2020, 30(1): 138–152.

[8] CHEN R R, LIU Y S, FAN H R, et al. An integrated approach for automated physical architecture generation and multi-criteria evaluation for complex product design. Journal of Engineering Design, 2019, 30(2−3): 63–101.

[9] GAO S, CAO Y, LI Y Q, et al. Reusability, reconfigurability and efficiency optimization of satellite network modeling and simulation. IEEE Access, 2023: 13: 122035−122058.

[10] FU C, LIU J B, WANG S D. Building SysML model graph to support the system model reuse. IEEE Access, 2021, 9: 132374–132389.

[11] CAO Y, LIU Y S, YE X P, et al. An automated approach for execution sequence-driven software and physical co-design of mechatronic systems based on hybrid functional ontology. Computer Aided Design, 2021, 131: 102942.

[12] PAHL G, BEITZ W. Engineering design-a systematic approach. 3rd ed. London: Springer-Verlag, 2007.

[13] GÓNGORA H G C, FERROGALINI M, MOREAU C. How to boost product line engineering with MBSE-a case study of a rolling stock product line. Proc. of the 5th International Conference on Complex Systems Design & Management, 2015: 239–256.

[14] YOUNG B, CLEMENTS P. Model based engineering and product line engineering: combining two powerful approaches at raytheon. INCOSE Insight, 2019, 22(2): 67–75.

[15] BILIC D, BROSSE E, SADOVYKH A, et al. An integrated model-based tool chain for managing variability in complex system design. Proc. of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, 2019: 288–293.

[16] TEKİNDERDOĞAN B, ÖZCAN M K, YAKıN İ, et al. Model-based systems product line engineering of physical protection systems. INCOSE International Symposium, 2021, 31(1): 1–15.

[17] FORLINGIERI M. The four dimensions of variability and their impact on MBPLE: how to approach variability in the development of aircraft product lines at Airbus. Proc. of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems, 2022. DOI: 10.1145/3510466.3511275.

[18] COLLETTI R A, QAMAR A, NUESCH S P, et al. Best practice patterns for variant modeling of activities in model-based systems engineering. IEEE Systems Journal, 2020, 14(3): 4165–4175.

[19] FORLINGIERI M, WEILKIENS T. Two variant nodeling nethods for MBPLE at airbus. INCOSE International Symposium, 2022, 32(1): 1097–1113.

[20] ARIFIN H H, KIT H, DAI J, et al. Model-based product line engineering with genetic algorithms for automated component selection. Proc. of the 4th International Conference on Complex Systems Design & Management Asia and of the 12th Conference on Complex Systems Design & Management, 2021: 287−310.

[21] WAGNER H, ZUCCARO C. Managing variability in digital twins and system development through product line engineering. Proc. of the IEEE International Symposium on Systems Engineering, 2024. DOI: 10.1109/ISSE63315.2024.10741090.

[22] LAMEH J, DUBRAY A, JANKOVIC M. Towards a configuration management integration to feature models in model-based product line engineering. Proceedings of the Design Society, 2023, 3(7): 3581–3590.

[23] WAGNER H, RÖSSLER W, ZUCCARO C. How to structure variability in large-scale complex engineered systems.

Proc. of the Tag des Systems Engineering, 2023: 272–279.

[24] ESSOUSSI M H, VIVOT P, DELOYE L, et al. Large legacy systems design maintainability through modeling. Proc. of the ERTS 2024, 2024: 1–10.

[25] COLLETTI R A, QAMAR A, NUESCH S, et al. Variant modeling for multi-perspective, multi-fidelity systems simulation. Recent Trends and Advances in Model Based Systems Engineering. Cham: Springer, 2022.

[26] COLLETTI R A, QAMAR A, BAILEY W, et al. A variable reference architecture for the management and configuration of ground vehicle simulations. Innovations in Systems and Software Engineering, 2024, 20(3): 185–207.

[27] BAJAJ M, FRIEDENTHAL S, SEIDEWITZ E. Systems modeling language (SysML v2) support for digital engineering. INCOSE Insight 2022, 25(1): 19–24.

[28] EPP J, ROBERT T, RUCH O, et al. Towards SysML v2 as a variability modeling language. Proc. of ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, 2023: 251–256.

[29] EPP J, ROBERT T, RUCH O, et al. Transitioning towards SysML v2 as a variability modeling language. Innovations in Systems and Software Engineering, 2024, 20(2): 585–596.

[30] LU J Z, WANG G X, YAN Y, et al. Semantic model-based systems engineering based on KARMA: a research and practice roadmap 2025. INCOSE International Symposium, 2022, 32(1): 706–720.

[31] LU J Z, WANG G X, MA J D, et al. General modeling language to support model-based systems engineering formalisms (Part 1). INCOSE International Symposium, 2020, 30(1): 323–338.

[32] WADA A, KOMATSU Y, TATE D, et al. Phased demonstrations of MBSE in small demonstration satellite series: development of system model and environment for full application of MBSE. INCOSE International Symposium, 2023, 33(1): 1188–1202.

[33] SELIC B. A Systematic approach to domain-specific language design using UML. Proc. of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007: 2–9.

[34] ZHANG B, WU Y F, ZHAO B Y, et al. Progress and challenges in intelligent remote sensing satellite systems. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2022, 15: 1814–1822.