Automated Design Space Exploration for Resource Allocation in Software-Defined Vehicles

Fengjunjie Pan, Jianjie Lin, Markus Rickert, and Alois Knoll

***Abstract*— Modern vehicles include an increasing amount of software, e.g., for autonomous driving capabilities, connectivity, and personalized user experience. The capabilities in current vehicles are still mostly provided by multiple separated embedded systems, while the current trend goes toward purely software-defined vehicles (SDV). Traditional distributed electrical/electronic (E/E) architectures have tightly coupled hardware/software, and the computational power is optimized for the included feature set. For SDVs, a centralized E/E architecture utilizing high-performance computers has been proposed. In contrast to individual embedded systems with limited and fixed functionality, combing a large set of individual software components in a single system leads to a high complexity in the proper allocation of resources. Model-based system engineering (MBSE) has been promoted in the automotive industry to handle complex system design. However, existing MBSE approaches focus mainly on traditional E/E architectures. In this work, we propose an automated and model-based approach that can address the resource allocation problem in SDVs. Users can formally describe the vehicle's resources, safety/non-safety requirements, and optimization objectives based on existing software engineering standards. The proposed method is not restricted to specific system models, requirements, or optimization goals and is, therefore, compatible with other E/E architectures. By introducing a model-independent transformation from the model information to solver-independent optimization formulas, the resource allocation problem can be solved automatically by a wide range of state-of-the-art solvers. We demonstrate the applicability of this approach in a SDV scenario with a high-performance computer and multiple applications.**

I. Introduction

Intelligent vehicles are drawing increasing attention from the automotive industry. Current cars and automotive electrical and electronic (E/E) architectures are primarily based on embedded systems with separated functionality and optimized hardware. Their functionality cannot be modified or updated on demand in the same scope as other devices such as computers or mobile phones. Domain and zone-oriented E/E architectures are currently investigated, where ECUs are grouped based on related function or locations in the car. The increasing complexity of autonomous driving functionality and the demand for up-to-date software in areas such as entertainment leads to the discussion of Software-Defined Vehicles (SDVs). In such a platform, software installation and updates can happen frequently and change the functionality of the whole system. To achieve maximum flexibility and scalability in SDVs, centralized architectures with high-performance central computers are discussed [1].

F. Pan, J. Lin, M. Rickert, and A. Knoll are with Robotics, Artificial Intelligence and Real-Time Systems, School of Computation, Information and Technology, Technical University of Munich, Munich, Germany. {panf, jianjie.lin, rickert, knoll}@in.tum.de

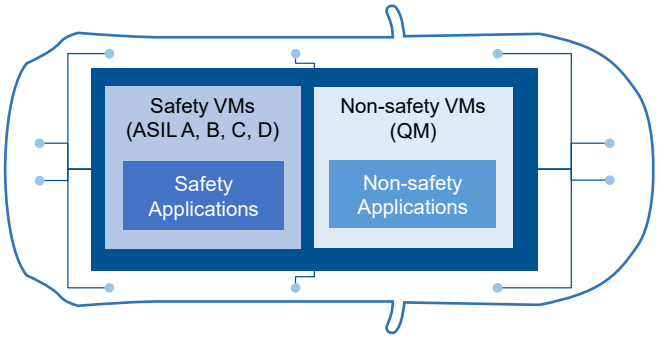

Fig. 1: Centralized E/E architecture in SDVs with support for applications and their resource and safety requirements.

Fig. 1 illustrates a possible SDV architecture based on a central computer, where virtualization technology is utilized to create independent virtual machines (VMs). Hardware resources and mixed-criticality applications are allocated to individual VMs for the safe execution of vehicle functions.

Designing resource allocation in SDVs is a design space exploration (DSE) process involving various (non-)safety requirements and optimization objectives. Consider the high complexity and the agile development cycle of SDVs, performing the DSE cannot remain a purely manual procedure. In our previous work [2], we formalized the resource allocation as a mathematical problem and utilized state-of-the-art solvers to perform the design task. This requires expertise in describing the problem in a mathematical format suitable for the solver. Therefore, we propose a model-based approach to further automate and simplify the design process of SDVs. Model-based system engineering (MBSE) has a proven track record in the aerospace industry and has received a lot of attention in the automobile industry in recent years [3], [4]. It leverages formally-described information for system design, analysis, verification, and validation. In the automotive domain, existing model-based design approaches, e.g., [5] and [6], mainly target distributed E/E architectures and only support limited types of requirements/objectives via custom domain-specific languages (DSL).

This paper proposes an automated and model-based DSE approach for resource allocation in SDVs. Our method is compatible with different architecture scenarios, requirements, and objectives so that it can be seamlessly applied to design SDVs. We introduce a model-independent transformation to encode model information (meta-models, instance models, constraints, and objectives) as mathematical expressions. Instead of a custom DSL, we rely on well-

known definitions from the software and systems engineering domain. Via state-of-the-art solvers, users can explore different design alternatives. To demonstrate the feasibility of our approach, we provide examples of SDV meta-models, requirements, and optimization goals. We further show how the proposed approach can be applied to a SDV scenario with multiple applications on a platform with many cores.

This paper is structured as follows: Section II introduces related work about automotive DSE and general MBSE methods. Section III shows fundamentals of model-based system description and provides examples of SDV-related models. In Section IV, the proposed DSE approach is presented, including model transformation. To show the applicability of our method, a concrete experiment is detailed in Section V.

II. Related Work

There are various research works discussing the design space exploration for resource allocation in automotive systems. Pohlmann et al. [5] presented the MechatronicUML approach for resource allocation in distributed vehicular systems. They created a platform-specific description language to model system components (e.g., cores, memories) and a custom DSL to specify five types of allocation constraints. Each constraint type was combined with a transformation template so that an integer linear programming (ILP) problem could be generated for the DSL automatically. ILP solvers were employed to process the generated formulations and to find feasible resource allocation configurations. Eder et al. [6] developed AutoFocus3 to map software into hardware. They utilized customized meta-models and a self-defined DSL for the formal description of system components, allocation constraints, and optimization objectives. Constraints and objectives following particular patterns were categorized and specified in the DSL. These DSL patterns can be translated to satisfiability modulo theories (SMT) expressions through pre-defined templates and further be optimized. Both MechatronicUML and AutoFocus3 target distributed automotive E/E architecture and offer limited support for the SDV architecture and requirements.

Al-Azzoni et al. [7] presented an adaptable framework to solve resource allocation problems, in which users can develop individual meta- and instance models to describe their systems. This framework differs from the above-mentioned works, whose meta-models were designed for particular systems; it provides a customized meta-model for software component allocation problems containing classes for target components and constraints. Models conforming to this meta-model can be translated to a mixed integer linear programming MILP problem and solved accordingly by MILP solvers. However, before the model can be applied to the automotive systems, a model transformation has to be defined by the user to convert into problem description models that conform to the framework's meta-model. Additionally, any undefined constraints in the meta-model cannot be resolved.

Compared to individual implementations, standard-based technologies can provide more comprehensive functionalities and broader industry acceptance. In the industry, the Eclipse Modeling Framework (EMF) is the de-facto modeling framework. Object Constraint Language (OCL) [8] is a widely acknowledged standard for constraint description. Thus, our research also draws on standard-based tools for validating and optimizing EMF- and OCL-based models. EMF2CSP [9] provides an automated translation from OCL expressions to constraint satisfaction problems to prove model consistency. It verifies if a meta-model can be instantiated according to the meta-model multiplicities and the OCL constraints. A valid instance model that conforms to the verified meta-model and follows the specified OCL constraints can be created. However, this tool cannot obtain information from the existing instance model, which typically contains hardware/software information and partial resource allocation decisions in an automotive DSE approach. EFinder [10] utilizes the USE Model Validator [11] to transform meta-models and OCL constraints into boolean satisfiability (SAT) problems and solves them with an SAT solver. In addition to maintaining consistency within the model, it can also identify and fill in missing information in terms of instance objects, references, and attributions for incomplete instance models. However, for the automotive resource allocation, instance objects of hardware resources should not be added during DSE. Besides, optimization problems can neither be defined nor executed.

To our knowledge, no existing DSE tool fulfills the following requirements simultaneously: (1) can be seamlessly applied for a centralized SDV architecture, (2) is based on existing systems and software engineering standards, (3) offers the freedom for specifying different kinds of requirements, (4) allows the specification of different optimization goals.

III. Model-based System Description

In MBSE, models that represent system information serve as the basis for analysis. Our work builds upon the existing standards for model description. Thus, it can easily combine with other tools conforming to the same standard, e.g., for model verification. The model-based description in this work contains several parts: meta-model, instance model, formal expressed requirements, and optimization objectives.

A. Meta-model and instance model

A meta-model is an abstraction of a target system and defines rules, meta-types, and properties needed for creating semantic models. The meta-models of systems have no distinctive design since they can be based on different abstraction levels or notions. An instance model instantiates a specific meta-model to describe a particular system. Due to industry-wide acceptance, we employ the Object Management Group (OMG) standards for the specification meta- and instance model. The OMG Meta Object Facility MOF defines a meta-data architecture in which every model element on every layer (instance) is strictly in correspondence with a model element of the layer above (classifier) [12]. It is supported by state-of-the-art modeling languages, including Unified Modeling Language (UML) and Systems Modeling

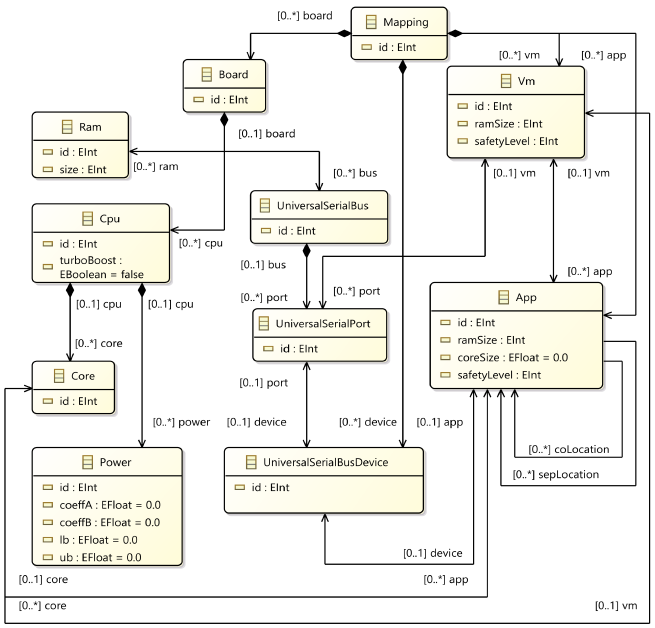

Fig. 2: An example meta-model for SDVs with classes for hardware and software components and their mapping.

Language (SysML). In the Eclipse Platform, the EMF is designed to realize the MOF standard [13]. EMF supports the specification of individual meta- and instance models using diagrammatic notations.

Fig. 2 presents an example of the EMF meta-model illustrated using the Eclipse Ecore Editor and describes the central-computer-based SDV architecture presented in Fig. 1. In the meta-model, we use classes (blocks) and attributes to specify hardware/software components and their properties. Various references (lines with arrows) are defined to express relationships among components, e.g., resource allocation or containment. We set the *Mapping* class as the starting point of the system modeling, which includes *Board*, *Vm*, *App*, and *UniversalSerialBusDevice*. Hardware components of the central computer are defined as separate classes (e.g., *Cpu*) contained by *Board*. The properties of each board component are modeled as attributes in each class. For example, we define a boolean attribute *turboBoost* in *Cpu* for the overclocking feature. Here, we assume that overclocking might jeopardize the stability of applications. Thus, we consider it safety-related. Furthermore, a separate class *Power* is created to express single core's power consumption as linear functions. With proper approximation, complex power functions can be described via piecewise linear functions instantiated by multiple instances of *Power*. In SDVs, *Vm* instances are isolated partitions to host applications. System configurations and resource allocations of *Vm* can be specified via attributes (e.g., *safetyLevel*) and references (e.g., *core*). The Class *UniversalSerialBusDevice* can be used to model sensors and heterogeneous hardware, e.g., cameras and AI accelerators. Application requirements are modeled

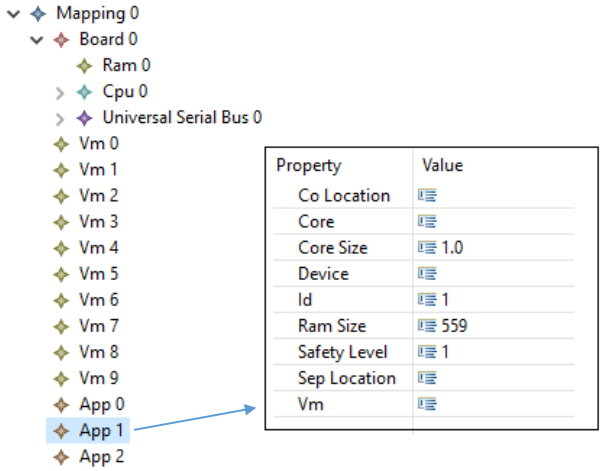

Fig. 3: Instance model example (partial) for a SDV system.

as attributes of *App* (e.g., *ramSize*). We consider each *App* instance a service segment that requires a maximum of one *Core* instance. Resource mappings of *App* are defined via references (e.g., vm). In addition, we define references *coLocation* and *sepLocation* to specify applications that need to be mapped to the same or different VMs. Apart from component descriptions, we can also specify basic allocation constraints via reference boundaries. For instance, the reference *vm* of *Core* is bounded from 0 to 1. It indicates that one core can be allocated to a maximum of one VM. Fig. 3 presents the instantiation of a SDV system based on the meta-model defined in Fig. 2. It describes part of the SDV system specified in Section V, where multiple hardware, VMs, and applications should be allocated. The properties of each instances are specified in a separate property view.

While system components and basic allocation constraints can be specified formally in EMF models, advanced constraints (e.g., freedom of interference) can hardly be expressed. Therefore, complementary constraint models need to be introduced for formal requirement specification.

B. Requirements

Formal constraint languages are designed as additions to the meta-model for constraint specification. In MBSE, they are typically employed for model verification. OMG introduces a declarative formal language, the OCL to support MOF-based models (EMF, UML, SysML, etc.) for describing constraints that cannot otherwise be expressed [8]. An OCL constraint contains a context (*context*) and an invariant (*inv*). In the OCL syntax, the context specifies the target of constraints. The rule of each constraint is defined as the OCL expression in each invariant. Due to its first-order-logic-like nature, OCL can be used to describe different automotive requirements. In this section, we will present some SDV-related requirement examples expressed via OCL.

1) Resource availability: The availability of resources is the fundamental requirement of resource allocation. In SDVs, the physical hardware is partitioned into VMs. In

order to host applications, sufficient sources (e.g., cores, memories, devices) must be assigned to each individual VM. The following OCL invariant *VmRamSize* shows an example of VMs' RAM allocation.

```
context Vm
 inv VmRamSize:
  app->collect(ramSize)->sum() <= ramSize
```

2) Application relationships: Automotive applications are interactive components that are dependent on each other. Thus, constraints are defined for the specification of relationships among different applications. The OCL invariant *CoLocation* specifies that co-locational applications should be assigned to the same VM. Similarly, the OCL invariant *SeperateLocation* describes that applications requiring separate locations should be executed on different VMs.

```
context App
 inv CoLocation:
  coLocation->forAll(vm = self.vm)
 inv SeparateLocation:
  sepLocation->forAll(vm <> self.vm)
```

3) Safety requirements: Freedom from interference (FFI) is an essential safety requirement introduced by ISO 26262 [14]. It defines the absence of cascading failures between two or more elements that could lead to the violation of safety goals. Automotive applications are categorized into different safety levels based on the risk classification scheme presented in ISO 26262. The Automotive Safety Integrity Levels (ASIL) A, B, C, and D are used to identify safety-critical applications, such as pedestrian detection. In turn, QM identifies non-safety-critical applications, such as infotainment. Based on the ASIL classification, we define that applications of different ASIL levels cannot influence each other. An OCL interpretation of FFI is presented in the constraint *AppAndVmSafety*. It specifies that applications can be only hosted by VMs with the same safety levels (e.g., an ASIL D application can only be executed on a VM which satisfies ASIL D).

```
context App
 inv AppAndVmSafety:
  safetyLevel = vm.safetyLevel
```

4) Safety-related platform constraints: Apart from system requirements, platform restrictions should also be addressed by the design of automotive systems. In SDVs, hardware should be appropriately configured to meet individual safety requests of VMs and applications. For example, a CPU overclocking feature (e.g., *turboBoost*) might jeopardize the stability of safety-critical applications. As a result, we define the constraint *VmSafetyConfig* as the core hosting safety applications should have *turboBoost* feature disabled.

```
context App
 inv VmSafetyConfig:
  safetyLevel > 0 implies
   core.cpu.turboBoost = false
```

OCL is capable of expressing various kinds of requirements. When using formal OCL constraint descriptions, instance models can be easily verified using pre-existing OCL verification tools without additional effort in implementation. In this paper, we aim for automated solving and optimization of the resource allocation problem. Thus, the OCL constraints will be transformed into mathematical formulations for further processing (Section IV).

C. Optimization objectives

In automotive system design, optimization improves system performance by adjusting system configurations without violating pre-defined constraints. Varying optimization goals can lead to different design solutions. OCL is a formal language initially developed for constraint specification. However, it may be utilized for querying model information as well. In this work, we combine OCL query expressions and optimization directions (minimize/maximize) to formally state different optimization objectives. We present three examples of SDV-related optimization objectives to show the feasibility of this approach.

1) Minimize VM numbers: In SDVs, VMs provide different execution environments to meet the requirements of applications. However, unnecessary VMs might influence the overall system performance. By minimizing the number of VMs that are instantiated, system overheads caused by VMs can be optimized. The optimization objective of minimizing the number of VMs (*minVm*) can be formally expressed as:

```
minimize:
 Vm.allInstances()->select(app->size() > 0)
  ->size()
```

2) Minimize number of utilized cores: CPU cores are the most crucial components of a modern vehicle, and host the majority of the calculations. However, due to the resource limitations of automotive platforms, the number of utilized cores has to be considered during the design of the E/E architecture. An SDV should be a dynamic system where new VMs and applications can be added to the system with restricted influence on existing configurations. Therefore, we set the minimal core number as the optimization goal. In a system utilizing a minimal number of cores, new VMs and applications can easily be allocated with free cores without interfering with the previous core configuration. The optimization objective of minimizing the number of cores (*minCore*) can be formally expressed as:

```
minimize:
 Core.allInstances()->select(app->size() > 0)
  ->size()
```

3) Minimize power consumption: Energy efficiency is a vital property for automotive platforms. Reduced power consumption benefits not only the environment but also the customer's experience and expense. For example, vehicles can travel with less fuel or electricity for longer distances with efficient power usage. The power consumption of a CPU core is typically a non-linear function, which can be approximated with piecewise linear functions. In Fig. 2, the class *Power* is created to define linear power consumption. Therefore, the OCL optimization goal for minimizing cores' power consumption (*minPower*) can be expressed as:

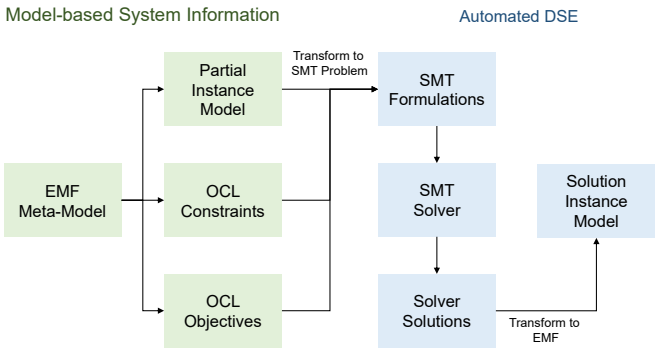

Fig. 4: Overview of the presented DSE approach.

```
minimize:
 Power.allInstances()->collect(
   p | Core.allInstances()->collect(
    app.coreSize->sum()
   )->collect(
     u | if (u > p.lb and u <= p.ub)
      then u * p.coeffA + p.coeffB
      else 0 endif
   )
 )->sum()
```

Given OCL's broad expressiveness, we consider it a suitable formal language to describe various optimization goals. In this work, the OCL objectives will be encoded together with OCL constraints to mathematical formulations for the DSE.

IV. Design Space Exploration

DSE is the procedure of searching for design solutions from a space of design possibilities that fulfill the desired criteria [15]. For the resource allocation in SDVs, the design space includes the configuration of hardware (e.g., safety-related CPU settings), the resource assignment to applications and VMs, and the mapping of applications to VMs. In this work, we propose an automated and model-based DSE approach (Fig. 4).

The starting point for our method is different models containing system information and requirements. The input models (including meta-model, partial instance model, constraints, and optimization objectives) should conform to existing modeling standards and can be defined by users according to individual project needs. A partial instance model can contain primitive system configurations depending on individual use cases. These configurations are preserved in the DSE solution. In Section III, we presented exemplary models for SDVs based on EMF and OCL. However, the proposed approach is not limited to EMF models. Other standardized modeling languages, e.g., UML and SysML, can also be applied. The information that is contained in the input models will be transformed automatically into a solver-understandable format. We utilize the solver-independent format, SMT-lib [16], to express optimization problems. The SMT-lib format is a standard that is used to describe SMT problems that determine whether a set of mathematical formulas can be satisfied. This format uses first-order logic as its underlying logic and can be processed by state-of-the-art SMT solvers, e.g., Z3 [17] and OptiMathSAT [18]. The description of optimization objectives is not contained in the current SMT-lib specification. Thus, we employ the extended definition of Z3 and OptiMathSAT to express optimization objectives [19]. Multiple optimization objectives can be defined and considered by a lexicographic priority. The objectives that are declared first have higher priorities during optimization. Apart from the proposed SMT-based DSE approach, the use of ILP or MILP solvers could be conceivable, but the automatic encoding from model information to solver-specific formats would be more complicated [20]. Once the solver completes the calculation, the solver solutions are transformed back to the EMF model for further inspection.

Algorithm 1 EMF2SMT to transform EMF models along with OCL expressions to SMT expressions

Require: OCL expressions, EMF models
for *class* **in** classes of the meta-model **do**
  for *ins* **in** instances of *class* in instance model **do**
    for *property* **in** properties of *class* **do**
      if *property* refers to a reference **then**
        Create binary decision variables for referred instances of *ins*
      else
        Create decision variable for *property* of *ins* according to the attribute type
      end if
      if *property* of *ins* is set **then**
        Assign values to decision variable
      end if
    end for
  end for
end for
for *expression* **in** OCL expressions **do**
  for *ins* **in** instances of context class **do**
    Set ins as context instance for *expression*
    res ← visitExpression(*expression*)
    Add *res* to the list of SMT expressions
  end for
end for
return list of SMT expressions

The key enabler of the proposed DSE approach is the automatic transformation from meta-/instance models, constraints, and objectives to a solver-understandable format. It is an exogenous and vertical model transformation since its source/target models are expressed in different languages and reside at different abstraction levels [21]. We propose the algorithm EMF2SMT (Alg. 1) to transform EMF models (meta- and instance models) and OCL expressions (constraints and objectives) to SMT expressions. The transformation can be categorized into two steps.

In the first step, EMF models will be parsed and analyzed. The information on classes and their properties is obtained

Algorithm 2 visitExpression() to visit OCL expressions

Require: OCL expression
switch type of OCL expression **do**
  case OperationCallExp
    $srcRes \leftarrow$ visitExpression(source expression)
    $srcRes \leftarrow$ visitExpression(argument expression)
    $opt \leftarrow$ get expression operator
    Combine $srcRes$, $argExp$, opt
    Formulate expression as SMT
  case PropertyCallExp
    $srcRes \leftarrow$ visitExpression(source expression)
    $property \leftarrow$ get referred property
    Query decision variables of $property$ for $srcRes$
    and represented instances (if applicable).
    Combine $srcRes$ and decision variables
    Formulate expression as SMT
  case IteratorExp
    $bodyExp \leftarrow$ get body expression
    $instances \leftarrow$ visitExpression(source expression)
    for ins **in** $instances$ **do**
      Set ins as context instance for $bodyExp$
      visitExpression($bodyExp$)
    end for
  case ...
return SMT expression, represented instances, etc.

from the meta-model. In EMF, a property can be either an attribute, which is a value of different types, or a reference, which links to other instances. In the instance model, all instances and their properties will be iterated and checked for existence. For each attribute property, individual decision variables of specific types will be created. For instance, we can create an integer variable $app1\_ram$ to represent the memory size of application 1. Since we focus on the automotive resource allocation problem, we mainly consider numerical representative attributes. For each reference property of a single instance, we propose to create binary decision variables for all possible referred instances to achieve a generic model transformation, e.g., to describe application 1's core assignment, we can create binary variables $app1\_core1$, etc., to link all possible cores. If a property already has defined values in the instance model, these values will be assigned to related decision variables as constants.

After reading the model information and creating decision variables, the OCL expressions will be analyzed and reformulated into SMT expressions. Both expressions of constraints and objectives follow the same OCL syntax. Each OCL expression has a recursive structure, which means an OCL expression can have other OCL expressions as subcomponents. OCL expressions can be categorized into different types: OperationCallExp, PropertyCallExp, IteratorExp, etc. As presented in Section III-B, the automotive requirements are typically expressed as OperationCallExps with different operators, e.g., plus/minus, multiplication/division, size, etc. Each expression type and each operator must be handled separately during the transformation. Alg. 2 presents examples of handling different OCL expression types. The algorithm takes a single OCL expression as input, queries instance model information according to the OCL expression, inserts decision variables, and produces SMT expressions that solvers can process. Examples of encoding OCL expressions to SMT expressions are presented in Section V.

V. Experiment

In this section, we present a demonstration of exploring the resource allocation design alternatives in SDVs to show the feasibility of the proposed method. The experiment was conducted with Eclipse IDE 2022-06 on a Windows 10 platform with Intel i7-8568U CPU (four cores) and 40 GB RAM. The meta-model was created using Eclipse Modeling Tools 4.24.0.20220609-1200. The OCL constraints and objectives were defined as text-based files via OCL All-In-One SDK 6.17.1.v20220309-0840. Moreover, we utilized the OCL visitor API and Java SE Development Kit 16.0.1 for the implementation and execution of transformation algorithms. The state-of-the-art SMT solver, Z3 (version 4.8.17), was employed to solve the transformed SMT problem.

The SDV scenario is defined in an instance model based on the meta-model in Fig. 2. Part of the instance model is illustrated in Fig. 3. We specified a central computer equipping one memory in size 40 GB, one CPU with 16 cores, one bus with two ports, and two devices. The power consumption p of each core was defined as a piecewise linear function with the core usage u as the input variable.

$$p = \begin{cases} 5u + 0.2\,, & 0 < u \leq 0.6 \\ 15u - 5.8\,, & 0.6 < u \leq 1 \end{cases}$$

In addition, we modeled 10 VMs and 40 applications with randomly generated requirements that need to be deployed onto the central computer. Part of the application requirements is listed in Table I. In the instance model, properties related to resource allocation (e.g., *turboBoost* in *Cpu0*, *vm* in *App0*) remained unset, since the DSE process should search for their solutions. To solve the resource allocation problem in SDVs, different requirements (Section III) should be considered at the same time. Thus, we prepared eleven OCL expressions to describe those requirements:

- Each application should be assigned to one core. (Section III-B.1)
- Each application should be assigned to one VM. (Section III-B.1)
- The core performance should not be exceeded. (Section III-B.1)
- Application should be mapped to VMs with the respective safety levels. (Section III-B.3, *AppAndVmSafety*)
- Co-location applications should be mapped on the same VMs. (Section III-B.2, *CoLocation*)
- Separate-location applications should be mapped on different VMs. (Section III-B.2, *SeparateLocation*)
- Cores assigned to safety-critical VMs should disable the turbo boost feature. (Section III-B.4, *VmSafetyConfig*)

TABLE I: Partial application requirements. This table shows the requirements of 8 out of 40 applications in the example SDV system.

id	ramSize	coreSize	safetyLevel	colLocation	sepLocation
0	120	0.1	QM (0)	5	-
1	559	1.0	ASIL A (1)	-	-
2	75	0.2	ASIL B (2)	-	-
3	250	0.6	ASIL C (3)	-	-
4	109	0.4	ASIL D (4)	-	-
5	222	0.4	QM (0)	0	6
6	328	0.5	QM (0)	-	5
⋮	⋮	⋮	⋮	⋮	⋮
39	66	0.3	QM (0)	-	-

- VM should provide sufficient cores for applications. (Section III-B.1)
- VM should provide sufficient memory for applications. (Section III-B.1, *VmRamSize*)
- VM should provide sufficient ports for necessary devices. (Section III-B.1)
- VM's memory assignment should not exceed the limitation of the hardware platform. (Section III-B.1)

Furthermore, we performed single-objective optimization for each optimization goals presented in Section III to discover different system configuration alternatives.

A. Automated model transformation

During DSE, the model-based information is converted into SMT formulations. In the transformed SMT expressions, each assertion (*assert*) is an instantiation of an abstract OCL constraint against a specific model instance. It is a boolean-valued function expressed as a logical proposition using the variables. In the demonstration, 3632 assertions are generated to describe the demonstrated scenario as an SMT problem. In order to increase the readability of the SMT formulation, we defined the name of each variable based on the corresponding class/reference/attribute name. The prefix *ref* is used for the specification of boolean variables representing the reference relationship in the models. For instance, *ref_app0_core0* is a boolean variable representing the mapping relationship between *app0* and *core0*. *vm0_ram* defines the RAM size of *vm0*. As all resources (e.g., resource amount, hardware containment relationship, and VM numbers) are predefined in the instance model, these values are presented as known values in expressions. Most OCL operators can be translated to corresponding SMT operators. In addition, we use the if-then-else (*ite*) operator in SMT to convert boolean variables to the numerical value 0 or 1. The following example shows transformed assertions of the constraint *VmRamSize* (introduced in Section III-B). The OCL constraint is expressed in the context of VM. Thus, ten SMT assertions are created for ten VM instances. In the assertions, the memory consumption of applications is given as values obtained from the instance model.

```
(assert (<= (+
  (* (ite ref_app0_vm0 1 0) 120) ...
  (* (ite ref_app39_vm0 1 0) 66))
  (* (ite (and true) 1 0) ramSize_vm0)))
...
(assert (<= (+
  (* (ite ref_app0_vm9 1 0) 120) ...
  (* (ite ref_app39_vm9 1 0) 66))
  (* (ite (and true) 1 0) ramSize_vm9)))
```

Optimization objectives in SMT problems are formalized via extended definition, where keywords *minimize* and *maximize* are specified. Each OCL objective will be transformed into a single SMT objective. The following SMT expression represents the objective of minimizing power consumption (introduced in Section III-C). To better show the transformed SMT expression, we use additional variables to replace part of the expression, e.g., util0 represents the SMT formula calculating the utilization of *core0*: *(+ (* (ite ref_app0_core0 1 0) 0.1) ... (* (ite ref_app39_core0 1 0) 0.3))*, in which 0.1 and 0.3 represent known core usages of *app0* and *app39*.

```
(minimize (+
  (ite (and (> util0 0.0) (<= util0 0.6))
  (+ (* util0 5.0) 0.2) 0) ...
  (ite (and (> util15 0.0) (<= util15 0.6))
  (+ (* util15 5.0) 0.2) 0)
  (ite (and (> util0 0.6) (<= util0 1.0))
  (+ (* util0 15.0) -5.8) 0) ...
  (ite (and (> util15 0.6) (<= util15 1.0))
  (+ (* util15 15.0) -5.8) 0)))
```

B. DSE solutions

We utilized the Z3 solver to explore the design possibilities of the SDV. The processing time varied based on the optimization engines used and based on the complexity of the problem. In our demonstration, we set the maximum processing time of the Z3 solver to 60 min. By enabling the WMax engine [19], the Z3 could return sub-optimal solutions, if no optimal solution can be determined within the time limit. After each DSE process, a solution instance model containing a complete resource allocation decision (including VM resource assignments, VM safety features, application mapping) was generated automatically. It can be further inspected by system engineers and used for deployment.

Fig. 5 shows system configurations optimized for different objectives. The axes represent the processing time and three system properties to be optimized: the power consumption, the number of VMs, and the number of cores. Each area illustrates a feasible resource allocation solution. Solution *minPower* (blue) and *minCore* (cyan) are resource allocation decisions optimized against power consumption and core number. Compared with other results, solution *minPower* has the least power consumption; solution *minCore* consumes the least cores. However, they are considered sub-optimal solutions since Z3 reaches the predefined calculation time limit. In solution *minVm* (orange), the optimal solution with the least VMs is presented. As the demonstration scenario has fewer VM components than cores, the optimization against objective *minVM* requires less processing time than *minPower* and *minCore*. Solution default (green) shows one of the feasible system configurations. Without any optimiza-

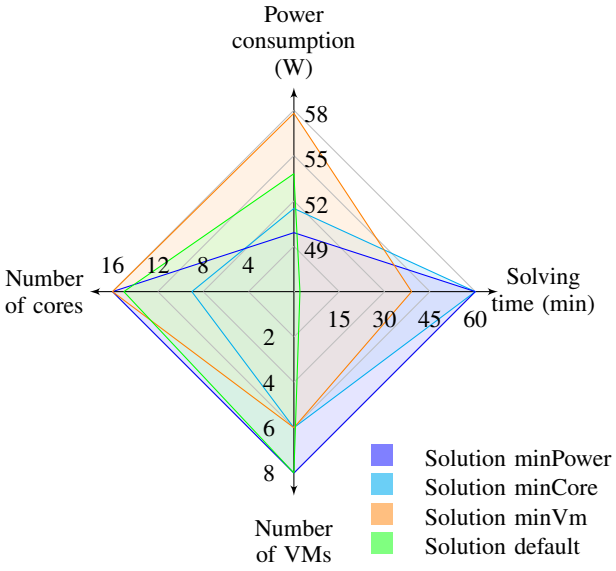

Fig. 5: SDV design alternatives applying different optimization objectives. Solution minPower, minCore, minVm are system configurations optimized for power consumption, core number, VM number respectively. Solution default is a feasible solution without any optimization.

tion objective, Z3 was able to solve the resource allocation problem within 2 min.

VI. Conclusion and Future Work

In this paper, we present an automated and model-based DSE approach for resource allocation in SDVs. We aim to accelerate the system design in SDVs through formal system descriptions. The presented method provides the flexibility of defining different systems, requirements, and objectives. Thus, it can be seamlessly applied to different scenarios derived from the development of SDVs. By introducing the model-independent EMF2SMT interface, model-based system information, including constraints and optimization goals, can be transformed into optimization problems in the SMT-lib format. Design solutions are generated automatically via state-of-the-art solvers and reflected in the models. The feasibility of this approach is illustrated through a complex SDV scenario.

This work builds upon widely-accepted MBSE standards MOF and OCL. Via the formal constraint language OCL, users are able to describe resource allocation problems without any expertise in the specific problem formulation. It can be easily extended with additional engineering steps (e.g., model validation via Eclipse OCL plugin). Furthermore, the presented DSE process is fully automated; therefore, it can be placed on the cloud or the vehicle itself for user-triggered software installations and updates.

As a future work, we plan to further simplify this process by combining natural language processing (NLP) techniques such that OCL constraints can be generated from requirements in natural language. In addition, constraint programming (CP) or ILP could also be utilized to express resource allocation problems. We plan to investigate and compare the performance of CP and ILP vs. SMT.

VII. Acknowledgement

This work was conducted as part of the Huawei-TUM Innovation Lab, sponsored by Huawei Technologies Düsseldorf GmbH. We would like to thank Sandro Nüesch, Raphael Fonte Boa Trindade, Christoph Ainhauser, and Marek Neumann for discussions related to safety-critical automotive functions.

References

[1] V. Bandur, G. Selim, V. Pantelic, and M. Lawford, "Making the case for centralized automotive E/E architectures," *IEEE Transactions on Vehicular Technology*, vol. PP, pp. 1–1, 01 2021.

[2] F. Pan, J. Lin, M. Rickert, and A. Knoll, "Resource allocation in software-defined vehicles: ILP model formulation and solver evaluation," in *Proceedings of the International Conference on Intelligent Transportation Systems (ITSC)*, 2022, pp. 2577–2584.

[3] P. Struss and C. Price, "Model-based systems in the automotive industry," *AI Magazine*, 2004.

[4] J. D'Ambrosio and G. Soremekun, "Systems engineering challenges and MBSE opportunities for automotive system design," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2017, pp. 2075–2080.

[5] U. Pohlmann and M. Hüwe, "Model-driven allocation engineering: specifying and solving constraints based on the example of automotive systems," *Automated Software Engineering*, vol. 26, pp. 315–378, 2019.

[6] J. Eder, S. Voss, A. Bayha, A. Ipatiov, and M. Khalil, "Hardware architecture exploration: automatic exploration of distributed automotive hardware architectures," *Software and Systems Modeling*, vol. 19, no. 911-934, 2020.

[7] I. Al-Azzoni, J. Blank, and N. Petrović, "A model-driven approach for solving the software component allocation problem," *Algorithms*, vol. 14, no. 12, 2021.

[8] OMG, *Object Constraint Language Version 2.4*, Feb. 2014.

[9] C. A. González, F. Büttner, R. Clarisó, and J. Cabot, "EMFtoCSP: A tool for the lightweight verification of EMF models," in *International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, 2012.

[10] J. S. Cuadrado and M. Gogolla, "Model finding in the EMF ecosystem," *Journal of Object Technology*, 2020.

[11] M. Gogolla, F. Hilken, and K.-H. Doan, "Achieving model quality through model validation, verification and exploration," *Computer Languages, Systems & Structures*, 2018.

[12] OMG, *OMG Meta Object Facility (MOF) Core Specification Version 2.5.1*, Oct. 2019.

[13] R. F. Paige, D. S. Kolovos, and F. A.C.Polack, "A tutorial on metamodelling for grammar researchers," *Science of Computer Programming*, vol. 96, pp. 396–416, 2014.

[14] "Road vehicles — functional safety," Standard ISO 26262:2018, 2018.

[15] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 8 - additional topics," in *Embedded Computing for High Performance*. Boston: Morgan Kaufmann, 2017, pp. 255–280.

[16] C. Barrett, A. Stump, C. Tinelli *et al.*, "The SMT-LIB standard: Version 2.0," in *Proceedings of the 8th international workshop on satisfiability modulo theories*, vol. 13, 2010, p. 14.

[17] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 4963, Apr. 2008, pp. 337–340.

[18] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," *Journal of Automated Reasoning*, Dec 2018.

[19] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "vZ - an optimizing SMT solver," in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*. Berlin, Heidelberg: Springer-Verlag, 2015, p. 194–199.

[20] S. Kugele, P. Obergfell, and E. Sax, "Model-based resource analysis and synthesis of service-oriented automotive software architectures," *Software and Systems Modeling*, vol. 20, pp. 1945–1975, 2021.

[21] M. Brambilla, J. Cabot, M. Wimmer, and L. Baresi, *Model-Driven Software Engineering in Practice*, 2nd ed. Springer Cham, 2017.