Formalization and Verification of AUTOSAR OS Standard's Memory Protection

LE Khanh Trinh*, Yuki Chiba†, and Toshiaki Aoki*
*Japan Advanced Institute of Science and Technology (JAIST), Nomi, Ishikawa, Japan
{trinh.le, toshiaki}@jaist.ac.jp
†DENSO CORPORATION, Chuo-ku, Tokyo, Japan
yuki_chiba@denso.co.jp

***Abstract*—AUTOSAR OS is a standard for automotive operating systems, which provides a specification that consists of functionalities such as scheduling services, timing services, and memory protection. In this paper, we focus on memory protection features among them. As the AUTOSAR OS specification is described in natural language, its ambiguity may confuse developers as well as cause the contradiction of the specification, then eventually lead to serious problems of automotive systems such as bugs and errors. These problems in automotive systems relate directly to the safety of human being. Thus, it is very important to ensure the unambiguity and consistency of the specification. Our solution for the problems is formalizing the AUTOSAR OS specification using Event-B specification language which allows us to formally specify the functionalities of AUTOSAR OS and reduce the ambiguity of natural language. We developed a formal specification of the memory protection of AUTOSAR OS and verified its consistency. In this verification, we found the inconsistency of the specification during discharging proof obligations generated by RODIN which is a tool for Event-B. This inconsistency comes from the ambiguity of the original specification, and finding it by reviewing based on natural language description is very hard. In this paper, we explain how we found the inconsistency existed in the AUTOSAR OS standard after showing our approach to formalize and verify it with Event-B.**

***Index Terms*—AUTOSAR OS, memory protection, formalization, verification**

I. Introduction

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry [1]; AUTOSAR OS (AUTOSAR Operating System) is a standard proposed by AUTOSAR. AUTOSAR OS specification is extended from OSEK/VDX OS specification [2] with a numerous of protection facilities such as timing, services, and memory protection. Protecting memory is important because the main purpose of the memory protection is to minimize the potential risks that could harm the memory of the system. Thus, in this paper, we focus on the AUTOSAR OS memory protection facilities. In the rest of this paper, the terms AUTOSAR OS specification [3], original specification, and informal specification are used equivalently to the specification of AUTOSAR OS memory protection.

AUTOSAR OS specification defines memory protection features to **protect** data and stacks of an OS-Application from the possible corruption of Non-Trusted OS Applications. The OS-Applications are introduced as basic entities of AUTOSAR OS. There are two types of OS-Application. The first type is *Trusted OS-Applications*, which are allowed to run in privileged mode at runtime. The second one is *Non-Trusted OS-Applications*, which have restricted access to memory. Each OS-Application contains OS-Objects such as Tasks and ISRs[1]. OS-Applications and OS-Objects have their own stack and may have data section. The code of an OS-Application is saved in the code section.

The problem is that these features are written in natural language, which may cause ambiguity. For example, the description "OS-Applications **can have** private data sections and Tasks/ISRs **can have** private data sections" has two ambiguous words including "can have" and "private". In detail, the word "can have" of this description has several kinds of implementation such as (1) OS-Application and Tasks/ISRs do not have private data section; (2) OS-Application and Tasks/ISRs have data section; (3) OS-Application has the data section but Tasks/ISRs do not have; and (4) Tasks/ISRs have the data section but OS-Application does not have. In addition, developers also are confused about the phrase "private data section". The meaning can be understood that the OS-Application owns the data section or all the data sections of the OS-Application cannot be shared.

In addition, the original specification may contain implications. For example, a requirement of the original specification said: "*OS-Application is permitted to read and write its own data section*", so the implicit understanding can be found is "OS-Objects which belong to an OS-Application are permitted to read and write the OS-Application's data section". From these problems of the AUTOSAR OS specification, there is a concern that the specification contains some contradiction.

Our solution for the problems is writing a specification using Event-B formal specification language [4]. Event-B uses set theory as modeling notation. Thus, it can remove the ambiguity of the descriptions in AUTOSAR OS specification. In this research, the structure of AUTOSAR OS is described in the *context* part of the Event-B description and the requirements are written as events in the *machine* part of Event-B. This formal specification strictly describes the structure of AUTOSAR

[1]Interrupt Service Routine

OS using set theory. Thus, it reduces the ambiguity caused by natural language. This formal specification is then used to verify the consistency of the AUTOSAR OS specification. There are two purposes for the verification. The first one is to evaluate that the formal specification faithfully formalizes the AUTOSAR OS specification. The other is to verify the consistency of the AUTOSAR OS specification. In order to do that, invariants are defined as constraints. Each invariant is written corresponding to each event and the implication in the form of "*Conditions* $\Rightarrow$ *Permission*", where the left-hand-side (LHS) of the invariant describes the guards of the event and the right-hand-side (RHS) describes the action of the event.

In this research, there are three obtained results of the formalization and verification of the AUTOSAR OS standard's memory protection. Firstly, the formalization removed the ambiguity of the AUTOSAR OS specification, which is written in natural language. Secondly, the formalization described the implications come from the AUTOSAR OS structure and the memory protection requirements. Thirdly, the verification found the inconsistency of the original specification. Then, we proposed an improvement to make the AUTOSAR OS specification consistent.

The remainder of this paper is organized as follows: section II shows problems and ideas of our approach. Section III presents formalization of AUTOSAR OS. The verification is illustrated in the section IV. Section V presents how to find the inconsistency. Section VI makes some discussions based on the result of the previous section. Some related works are presented in the section VII to compare with our research. Finally, section VIII presents some conclusions and future works.

II. Approach

In this section, we explain our approach to formalizing the AUTOSAR OS specification, which can be verified the consistency of the specification. The figure 1 shows an overview of our approach.

There are three main steps of the formalization. The first step is analyzing the original specification to find components constructing the architecture of AUTOSAR OS. Most of the information about AUTOSAR OS's structure is written in natural language and scattered distributed in the original specification. The original specification is arranged to figure out the components of AUTOSAR OS and is analyzed to show the evidence that this specification contains the implicit understanding. In order to present AUTOSAR OS architecture and the implicit understanding clearly, we present such components using sets and relations. The requirements are then described on top of the structure's components.

The second step is writing the formal specification of AUTOSAR OS, which is used to verify the consistency of the AUTOSAR OS specification. The formal specification is written to reduce the ambiguity of the original specification. For example, the formal specification should cover all meaning of the "can have" word in the AUTOSAR OS descriptions.

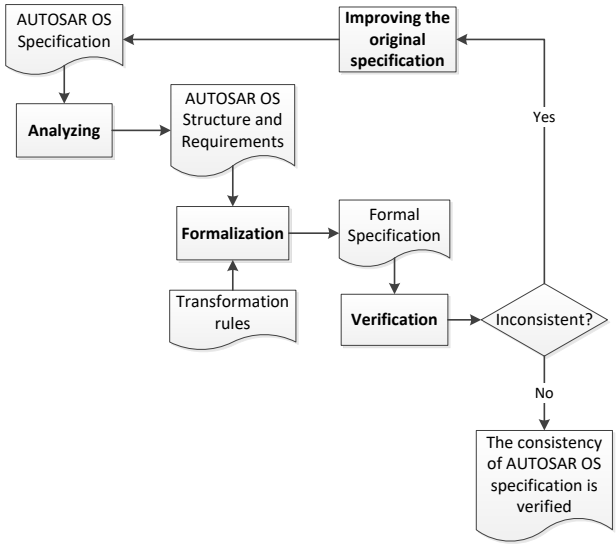

Fig. 1. Overview of formalizing the memory protection of AUTOSAR OS.

A list of transformation rules is defined to map the analyzed components as formal notations in Even-B language. These transformation rules aim at bi-directional traceability between the formal specification and informal specification to ensure the integrity of AUTOSAR OS specification. They also support to establish trace links between elements in the original specification and the formal specification. We choose Event-B to write the formal specification. Event-B is a formal specification language based on mathematics and it supports set theory, which is suitable to describe the analyzed components of AUTOSAR OS in the previous step. However, the protection is described as if-then-else event in the original specification. For example, "read and write access from a non-trusted application to the data section of another OS-Application is prevented". These events of the original specification are not real events because it does not contain behavior. Thus, the notion of the events is used to represent such if-then-else in Event-B. By applying transformation rules, each component of AUTOSAR OS is formalized in Event-B notations. An Event-B specification is made of several components of two kinds: *Contexts* and *Machines*. Machines contain the variables, invariants, theorems, and events of an Event-B specification, whereas contexts contain carrier sets, constants, axioms, and theorems. In the AUTOSAR OS formal specification, the context is used to formalize the architecture of AUTOSAR OS and the machine formalizes requirements.

The third step is verification. The purpose of this step is to verify the consistency of the original specification. In order to do that, the invariants are written corresponding to the memory protection's requirements to ensure the correctness of our formalization. The implicit understanding is also formalized as invariants using events and facts defined in the context. Then, proof obligations are generated by the Rodin

platform [4], which is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. These proof obligations are then discharged to verify the consistency of the AUTOSAR OS specification. If all proof obligations are discharged, the original specification is consistent. Otherwise, there is some inconsistent description of it. If there is any inconsistency, the original specification should be improved. According to this approach, the inconsistency of the AUTOSAR OS specification was found in the description "OS-Applications private data sections are **shared by all** Tasks/ISRs belonging to that OS-Application".

III. Formalization of AUTOSAR OS standard's memory protection

In this section, we present the formalization of AUTOSAR OS standard's memory protection. The main point is to introduce the structure of AUTOSAR OS. After analyzing the informal specification to find components of the structure, a list of transformation rules is used to write the formal specification. Then, AUTOSAR OS's requirements are described on top of the structure.

A. Analyzing the AUTOSAR OS specification

In this step, the input is AUTOSAR OS specification document. From this document, our formalization focuses on three representation kinds of operating system's components in Event-B: entity, relation, and constraint.

Entities: From the AUTOSAR OS specification document, entities relating to the memory protection are identified and extracted the description of their attribute. There are four kinds of entities are shown in table I.

TABLE I
Entities of AUTOSAR OS specification

Entities	Description
OS_Applications	The basic realms of spatial isolation in AUTOSAR OS [5]
OS_Objects	The set of elements inside an OS_Applications like *Tasks* and *ISRs*
Memory	The object of protection, which contains *Data sections* and *Stack*
Code Sections	This part contains source codes of an AUTOSAR OS

Relations: To formalize the memory protection specification, the structure of the system is visualized using entities and relationships between them. However, descriptions of the relations are scattered in the original specification. Thus, analyzing AUTOSAR OS specification is necessary to collect the relations. For example, The relation *"each object has their own stack"* is found by analyzing the description *"The stack for these objects, by definition, belongs only to the owner object and there is, therefore, no need to share stack data between objects, even if those objects belong to the same OS-Application"*. Moreover, there is a constraint from this description is that "*the stack data of an object cannot be shared with others*"

Constraints: They are limitations which the system is expected to accomplish. The description of the constraints is scattered throughout the original specification as shown in the above example. Therefore, it takes the effort to identify each of them.

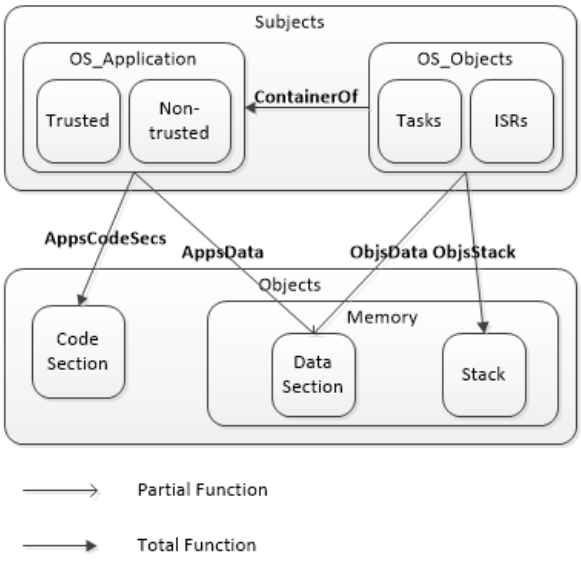

Fig. 2. The structure of AUTOSAR OS

The structure of AUTOSAR OS is visualized as figure 2. In the aspect of accession, the AUTOSAR OS's structure is shown that the source of the accession is *Subjects* and the destination is *Objects*. The set *Subjects* has two subsets are *OS_Applications* and *OS_Objects*. Two kinds of *OS_Applications* are *Trusted_OS* and *Non-Trusted_OS*. The *OS_Objects* also has two kinds are *Tasks* and *ISRs* (Interrupt Service Routine). The set *Objects* has two subsets are *CodeSections* and *Memory*. The *Memory* has two parts are *Data Section* and *Stack*. Two types of function are used to present the relationship between system's elements are: *partial function* and *total function*, where the partial function and total function presents "*at-most-one*" ownership ($\nrightarrow$) and "*always have*" ownership ($\rightarrow$) respectively.

B. Formalizing the AUTOSAR OS structure

After analyzing the AUTOSAR OS specification, the AUTOSAR OS components are collected. The idea of the formalization is formalizing AUTOSAR OS's memory protection features based on accession's description. For more detail, the *entities* are classified into two sub-classes: *Subjects* describes the source of the accession, including *OS_Application* and *OS_Objects*. *Objects* describes the destination of the accession including *Memory* and *Code Section*. The formalization is performed by writing each sub-class, relation, and constraint by a proper Event-B notation. Transformation rules are defined in the table II to write a formal specification in Event-B based on the analysis. The detail of transformation rules are explained as follows:

Table III show the applications for the rules 1,3, and 7. *Subjects* and *Objects* are defined in the Event-B as carrier

TABLE II
TRANSFORMATION RULES

Rule	Components	Event-B notations
rule 1	Entities classes	Carrier sets
rule 2	Attributes	Variables
rule 3	System's objects	Constants
rule 4	Requirements	Events, Invariants
rule 5	Accession Conditions	Guards
rule 6	Permissions	Actions
rule 7	Relationships and Global constraints	Axioms

sets. The relations *ObjData*, *ObjStack* and the objects *OS_Obj*, *OS_App* are defined as constants. Then, the related constraints are defined in the axioms.

TABLE III
FORMALIZING AUTOSAR OS'S ELEMENTS

Type	Event-B notations	Explanation
Carrier Sets	Subjects, Objects	The source and destination of an accession
Constant and axioms of objects	$OS\_Obj \subseteq Subjects$ $OS\_App \subseteq Subjects \setminus OS\_Obj$	Two subsets of set *Subjects*
	$CodeSection \subseteq Objects$ $Memory \subseteq Objects \setminus CodeSection$	This *Object* contains the code of an OS_Application This element presents the AUTOSAR OS's memory
Constant and axioms of relations	$ObjData \in OS\_Obj \nrightarrow DataSecs$	An OS_Object has at most one data sections
	$ObjStack \in OS\_Obj \rightarrow Stacks$	An OS_Object always have stack
	$ContainerOf \in OS\_Obj \rightarrow OS\_App$	An OS_Object belongs to an OS_Application

In the original specification, the relations *ObjData* and *AppData* describe that *OS_Objects* and *OS_Applications* **can have** data section. The word "can have" can be understood in two ways: they have no data section, and they have data section. Both of the understanding are satisfied the specification. Thus, the formal specification should cover all the meaning of the original description. In the formalization, the partial function is used to formalize the "can have" word.

The remaining rules are applied to formalize requirements of AUTOSAR OS specification. Each AUTOSAR OS standard's memory protection requirement is described in the same name event. Conditions of the requirements are described as guards and the permission is described as the action of the event. The permission is not a real action but we use the notion of events to describe the requirement. Thus, the permission is considered as an action. For instance, the requirement SWS_OS_086 "The Operating System module shall permit an OS-Application read and write access to that OS-Application's own private data sections" is written in Event-B follows:

event_SWS_Os_00086 **ANY** *app*
GUARDS
grd1: $action = read \lor action = write$
grd2: $app \in dom(AppData)$
grd3: $src = app$
grd4: $dst = AppData(app)$
grd5: $status = initStt$
ACTION $status := shall\_permit$

By following the transformation rules, the structure of AUTOSAR OS is faithfully formalized. We ensure that no feature or element is missed or added redundantly during the formalization because the traceability is ensured using these rules.

IV. VERIFICATION OF AUTOSAR OS MEMORY PROTECTION SPECIFICATION

In this section, we first define the consistency of the memory protection. Then, we explain how to ensure the consistency by discharging proof obligations in Event-B. In the verification, invariants are defined as AUTOSAR OS properties which are maintained to be always true all reachable states in the Event-B state machine. Based on the invariants and events, the Rodin platform generates proof obligations and discharges them to prove the consistency of the AUTOSAR OS memory protection.

A. Purpose of the verification

The first thing needs to be clarified is the purpose of the verification. The verification aims to:

1) Clarifying implications of the original specification, and
2) Verify the consistency of the original specification.

In order to deal with the first one, the original specification is analyzed carefully to discover all possible implications and define them as invariants. Basically, the implications are defined based on the invariants expression of the corresponding event but changed the conditions of the invariant. In particular, the events which have "*src*" is an OS-Application imply that all Tasks/ISRs belonging to that OS-Application can be "*src*". These implications are then formalized to verify the consistency of the original specification.

As described in chapter III, the formal specification is faithfully described the original specifications. Hence, the traceability has been ensured and it is possible to verify the consistency of the original specifications through the formal specifications. The consistency is defined that "**Each access has an unique permission**".

Definition 1. Access
Let S be the set of subjects and O be the set of the objects. *a(s,o)* presents an access from s to o where $s \in S$ and $o \in O$.

Definition 1 defines accesses in AUTOSAR OS memory protections, where Subjects consists of OS-Application and OS-Objects and Objects consists of Memory and Code sections.

Definition 2. Consistency
$\forall s, s' \in S, o, o' \in O.\ s = s' \land o = o' \Rightarrow a(s,o) = a'(s', o')$

The consistency is defined based on the access as the definition 2. For all access a from subject s to object o and access a' from subject s' to object o'. If s is s' and o is o', then the access a and a' is the same. In the other words, the access from a subject to an object is unique.

B. Defining invariants to generate proof obligations

The transformation rule 4 describes that each requirement is described not only as an event but also as an invariant as "$C \wedge status \neq initStt \Rightarrow P$" where C is the set of the condition's expressions and P is the permission expression. The LHS of the invariant describes the guards of the event and the RHS describes the action of the event. Expression "$status \neq initStt$" is used to ensure that each event is executed once for an invariant where *initStt* is the initial value of the system's status.

There are many kinds of proof obligations such as *well-definedness* and *invariant preservation*. In this research, we focus on the ***Invariant Preservation Proof Obligations***. For each event and invariant, the Rodin platform [6] generates a corresponding invariant preservation proof obligation. This proof obligation is then discharged to verify the consistency of the two requirements (once from the event and another one from the invariant).

After executing an event, the value of variables appearing in the event is updated. The consistency is then verified by checking the following condition which ensures that the invariant is preserved for each of events.

$(C \wedge status \neq initStt \Rightarrow P) \Rightarrow (C' \wedge G \wedge status' \neq initStt \Rightarrow P')$

where *C'*, *status'* and, *P'* are the updated value of *C*, *status* and, *P* respectively; and G is the conjunction of an executed event's guards. In fact, for each executed event, *status* is updated but the variables appearing in *C* are not updated. Furthermore, "$status \neq initStt$" is used to ensure that each event is executed once for an invariant. Thus, the condition becomes

$(C \Rightarrow P) \Rightarrow (C \wedge G \Rightarrow P')$

If the $C \wedge G$ is satisfiable and $P = P'$, that condition becomes true, that is, the two requirements are consistent. If the $C \wedge G$ is satisfiable but $P \neq P'$, that condition becomes false, that is, the two requirements are inconsistent.

C. Discharging the proof obligations

After writing the formal specification of AUTOSAR OS memory protection, proof obligations will be generated by Rodin platform to verify their consistency. There are two kinds of the proof obligations including:

- The invariant and the event are the same requirement: This proof obligation is used to prove that the approach of the formalization is true.
- The invariant and the event are the different requirements: If the proof obligation generating from them cannot be discharged, there is a contradiction between the requirement of the event and the requirement of the invariant. Otherwise, these requirements are consistent.

Rodin platform provides some provers to discharge these proof obligations. The provers are useful for discharging the *well-definedness* proof obligations. However, it was impossible to discharge all *Invariant Preservation* proof obligations using the provers. 108 of 171 proof obligations are discharged using automated proof including 31 *well-definedness* proof obligations and 77 *Invariant Preservation* proof obligations. Therefore, we apply interactive proof methods to discharge this proof obligation.

Interactive proving

It was impossible to discharge all proof obligations automatically using the provers. Interactive proof methods are used to discharge the remaining proof obligations. There are some important methods to discharging the proof obligation as follows: (1)*Re-ordering hypotheses*: removing unnecessary hypotheses; adding needed hypotheses which have been missing; (2) *Creating a case distinction*: applying case distinction to find the cause of the problem; (3) *Instantiate quantifiers variable*; and (4) *Applying abstract expression* to replace the complicated expression with fresh variables.

For example, table IV describes two parts of the obligation "*SWS_Os_00026/prf_086/INV*" are *hypotheses* in the first column and *goals* in the second column. This proof obligation is undischarged because the goals is not satisfied by the hypotheses. In the first step, the hypotheses $\neg read = write$, $\neg initact = read$, $\neg initact = write$, $\neg may\_permit = shall\_prevent$, $\neg may\_prevent = shall\_prevent$, $\neg init = may\_prevent$, $\neg init = may\_permit$, $\neg may\_permit = may\_prevent$, $\neg init = shall\_prevent$, and $\neg init = shall\_permit$ are removed. These hypotheses are generated from *set_of_action*={*initact*, *read*, *write*} and *set_of_status*={*init*, *shall_permit*, *may_permit*, *may_prevent*, *shall_prevent*} to ensure that elements of the sets are different but it is unnecessary. The hypotheses set is then added some axioms from the Event-B context to satisfy the goals. In this example, the hypothesis $AppData \in OS\_App \nrightarrow DataSecs$ is added to describe the function *AppData*. The goals of the proof obligation $SWS\_Os\_00026/prf\_086/INV$ is satisfied by applying the hypotheses set.

TABLE IV
PROOF OBLIGATION *SWS_Os_00026/prf_086/INV*

Hypotheses	
action = read	$\neg read = write$
$app1 \in NonTrusted\_OS$	$\neg initact = read$
$app2 \in OS\_App$	$\neg initact = write$
$src \in app1$	$\neg may\_permit = shall\_prevent$
$dst = AppData(app2)$	$\neg may\_prevent = shall\_prevent$
$app2 \in dom(AppData)$	$\neg init = may\_prevent$
$status = init$	$\neg init = may\_permit$
set_of_action={*initact*, *read*, *write*}	$\neg shall\_permit = may\_prevent$
set_of_status={*initStt*, *shall_permit*, *may_permit*, *may_prevent*, *shall_prevent*}	$\neg may\_permit = may\_prevent$
$\neg app1 = app2$	$\neg init = shall\_prevent$
$action = read$	$\neg init = shall\_permit$
Goals	
$\forall$ app .$(action = read \vee action = write) \wedge app \in dom(AppData) \wedge src = app \wedge dst = AppData(app) \wedge may\_prevent \neq initStt \wedge \Rightarrow may\_prevent = shall\_permit$	

In our work, the method *Re-ordering hypotheses* is used to discharge proof obligations. 62 of 63 remaining proof obligations were discharged using this method. However, the last poof obligation *SWS_Os_00195/prf_086g/INV* cannot be

discharged by all above methods. We worry that it may present the inconsistency of the AUTOSAR OS specification.

V. The inconsistency of AUTOSAR OS specification

The re-ordering hypotheses method solved almost all proof obligations. However, the proof obligation *SWS_Os_-00195/prf_086g/INV* cannot be discharged by any proof method. This proof obligation is generated to prove that the invariant prf_086g is preserved by event SWS_Os_00195. These requirements are described as follows:

- *SWS_Os_00195*: "*The Operating System module may prevent write access to the private data sections of a Task/Category 2 ISR of a non-trusted application from all other Tasks/ISRs in the same OS-Application.*", and the corresponding formalized description is:
  $\forall app1, obj1, obj2 \ . \ ((action = write) \wedge app1 \in NonTrusted\_OS \wedge obj1 \in OS\_Obj \wedge obj2 \in dom(ObjData) \wedge obj2 \in (Tasks \cup Category\_2\_ISRs) \wedge obj1 \neq obj2 \wedge app1 = ContainerOf(obj1) \wedge app1 = ContainerOf(obj2) \wedge src = obj1 \wedge dst = ObjData(obj2) \wedge status \neq initStt \Rightarrow status = may\_prevent)$

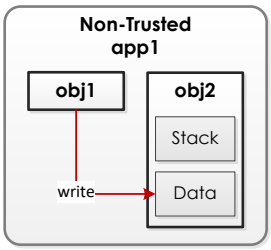

Fig. 3. Requirement SWS_Os_00195.

The requirement SWS_Os_00195 is visualized as Fig. 3.

- *SWS_Os_00086g*: "*OS-Objects contained in an OS-Application shall be permitted to read and write this OS-Application's private data sections*", and the corresponding formalized description is:
  $\forall app1, obj1 \ . \ ((action = read \vee action = write) \wedge app1 \in dom(AppData) \wedge obj1 \in OS\_Obj \wedge app1 = ContainerOf(obj1) \wedge src = obj1 \wedge dst = AppData(app1) \wedge status \neq initStt \Rightarrow status = shall\_permit)$

Figure 5 visualizes the goal for discharging the proof obligation between SWS_Os_00195 and SWS_Os_00086g. If the goal is true, the two requirements are consistent. Otherwise, they are inconsistent. The goal of this proof obligation is: $\neg app2 \in dom(AppData) \vee \neg obj3 \in OS\_Obj \vee \neg app2 = ContainerOf(obj3) \vee \neg obj1 = obj3 \vee \neg ObjData(obj2) = AppData(app2)$.

If one of these expressions is true, we can conclude that the goal is held. From the description of the requirements, three expressions $\neg app2 \in dom(AppData)$, $\neg obj3 \in OS\_Obj$, and

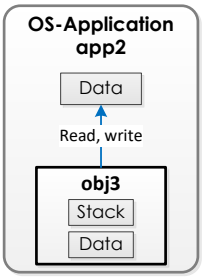

Fig. 4. Requirement SWS_Os_00086g.

$\neg app2 = ContainerOf(obj3)$ are always false. Therefore, we just considered the correctness of expressions $\neg obj1 = obj3$, and $\neg ObjData(obj2) = AppData(app2)$.

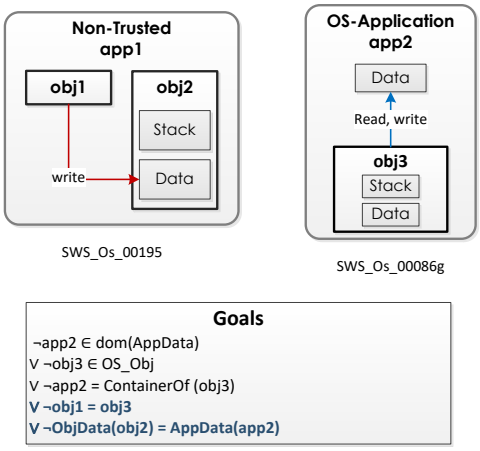

Fig. 5. The goal for discharging the proof obligation SWS_Os_00195/prf_-086g/INV.

The case distinction method is used to discharge the proof obligation as follows:

- $app1 \neq app2$: The $app1$ and $app2$ are different implies $obj1 \neq obj3$ is true. Then, the proof obligation *SWS_-Os_00195/prf_086g/INV* is hold.
- $app1 = app2$: When $app1$ = $app2$, we cannot ensure whether expression $obj1 \neq obj3$ and $\neg ObjData(obj2) = AppData(app2)$ are true.

The correctness of the expression $\neg ObjData(obj2) = AppData(app2)$ is not identified because the description "OS-Applications private data sections are **shared by all** Tasks/ISRs belonging to that OS-Application", which is a global constraint could be understood in two different meanings. The first meanings is that an OS-Application is initiated a memory partition, then each task or ISRs is distributed appropriate memory partition from the OS-Application's memory for running. This approach is presented in the figure 6 and the formal description as follows: $\forall obj, app \ . \ (app \in dom(AppData) \wedge obj \in OS\_Obj \wedge obj \in dom(ObjData) \Rightarrow ObjData(obj) \neq AppData(app)$. In this case, the expression

$\neg ObjData(obj2) = AppData(app2)$ is true. Hence, the goal is satisfied.

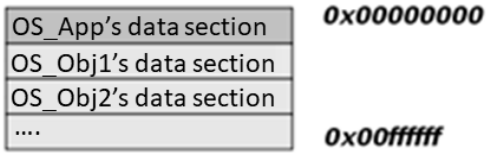

Fig. 6. Os-Application distributes memory partitions to its OS-Objects.

The other considers the memory partition of an OS-Application as a resource and each OS-Object like Task, ISR can access to it. Therefore, the case that the OS-Application and OS-Objects have the same resource can be occurred as the presentation of figure 7. Then, the formal description of this global constrain is: $\forall obj, app \,.\, (app \in dom(AppData) \wedge obj \in OS\_Obj \wedge obj \in dom(ObjData) \wedge \ app \neq ContainerOf(obj)) \Rightarrow ObjData(obj) \neq AppData(app)$. The figure 7 shows that

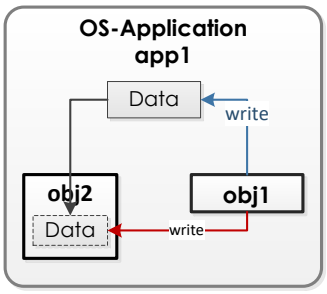

Fig. 7. OS-Application and its OS-Object are considered as subjects

the OS-Application *app1* which contains two OS-Object obj1 and obj2 has a data section. app1 shares this data section to obj1. Regarding the requirement SWS_Os_00195, the write access from obj1 to obj2's data section is *prevented*. However, requirement SWS_Os_00086g describes that the write access from obj1 to app1's data section is *permitted*. This is the counterexample of the definition 2.

As the result of the verification, the ambiguous description of the original specification is discovered is that "OS-Applications private data sections are **shared by all** Tasks/ISRs belonging to that OS-Application". The original specification needs to be improved to ensure the consistency. From the above analysis, we propose the new description is: "The data section of an OS-Application **should not be shared** to the Tasks/ISRs belonging to it".

VI. Discussion

In our formalization, the traceability between two specifications is achieved by following the specification policy. In detail, the elements of the formal specification formalize exactly the components of the AUTOSAR OS specification. If there is any component of the original specification cannot be found in the corresponding element in the formal specification, it means that the component lost during formalization. Then, the formal specification must be added the corresponding elements. On the other side, the components can be traced back from elements of the formal specification. If a formalized model has not corresponded to any component in the original specification, it must be removed because it has been added redundantly into the formal specification. It was proved that the approach of this thesis is appropriate for preventing not only the loss but also the redundancy of the formalization.

In the formalization, there may be many ways to understand and formalize from a description of the original specification. We carefully analyzed and picked the appropriate Event-B notations to cover all possible understanding of a description. Then, the consistency of the specification is verified by discharging proof obligations, which are generated automatically by Rodin platform. From 9 events and 18 invariants, there are 171 proof obligations are generated including 140 *invariant preservation proof obligations* and 31 *well-definedness proof obligations*. By applying automatic proof, 108 of 171 proof obligations including all 31 *well-definedness proof obligations* are discharged accounting for 63.2% of the total proof obligations. 62 other proof obligations are discharged using interactive proof accounting for 36.3%. The last obligation cannot be discharged is then analyzed and the inconsistency is discovered from this analyzing.

As the result of the verification, the inconsistency of the AUTOSAR OS memory protection's specification has been found. This inconsistency comes from the ambiguous description of the original specification is that "*the data section of an OS-Application is shared by all its own OS-Objects*". Our approach is effective for verifying the consistency of the memory protection specification because it is hard to verify the consistency by reviewing the formal specification. Finally, we proposed an improvement based on the ambiguous descriptions in the original specification making the inconsistency.

VII. Related work

Our formalization methodology includes three main steps: *Analyzing*, *Formalization* and *Proving* corresponding to three steps of a general formalization process mentioned in [7]. Tools or frameworks are proposed to formalize informal specification such as A. Fantechi et al. [8] develops a tool for translating informal specification into formulae of the action-based temporal logic ACTL. Huang et al. [9] employ process algebra CSP to describe and reason about a real code-level OSEK/VDX operating system (***O**ffene **S**ysteme und deren Schnittstellen fr die **E**lektronik in **K**raftfahrzeugen*). The expected properties are described and expressed in terms of the first-order logic. They propose a framework to establish and verify the properties to check the deadlock-free of the system and the soundness of the scheduling scheme. In contrast, our work uses sets and relations to represent the structure of AUTOSAR OS. The requirements are then described on top of this structure.

Some other works propose the integrating between formal and informal methods in order to provide an evolutionary path

to the use of more formal approaches to software development [10] or increase the complement formal and informal specification [11]. In our research, the equivalent between formal and informal specification is evaluated based on the bi-directional traceability instead of proving the completeness of such specifications. Craig used Z and Object-Z as formal specification languages to formally specify a conventional paradigm of operating system [12]. However, this work makes the formal specification from scratch but the input of our formalization is a given informal specification.

Huong et al. proposed a method which can faithfully formalize the OSEK/VDX operating system using Event-B [13]. Comparing with our work, the features of OSEK/VDX operating system has already contained system behavior, so events presenting these features are created directly. On the other side, AUTOSAR OS memory protection features are not a real event. Thus, such features are formalized using the notion of the events. D. Bertrand et al. are interested in timing protection, which is another facility of AUTOSAR OS [14]. The purpose of this work is evaluating the timing protection mechanism proposed in the AUTOSAR OS standard. This evaluation shows that the present version of the mechanism is not fully adapted to multi-critical systems because it does not handle soft/non real-time applications.

VIII. Conclusion

In conclusion, we proposed a formal specification of AUTOSAR OS memory protection. The requirements of the original specification are static constraints which do not contain the behaviors of the system. Thus, the notion of events is used to formalize such requirements. Although proving a complete equivalence between the original and formal specifications is impossible, the confidence of the formalization is ensured by applying transformation rules. Hence, the formalization reaches two goals: (1) the formal specification can formalize exactly every element and requirement of the AUTOSAR OS memory protection and (2) the formal specification does not contain redundant elements. The formal specification is the input of the verification to verify the consistency of the memory protection. The result of the verification is the ambiguous description in the original specification. Thus, we improved this description to remove the inconsistency. This improvement will be sent to the related organization to confirm our works.

There are some advantages of our research. Firstly, this approach is able to find the inconsistency, which is hard to be found in natural language. We introduced a formal specification to formalize faithfully the AUTOSAR OS specification. This formal specification can cover all possible implementations from a description. Secondly, our research is practical enough to deals with the AUTOSAR OS, which is a real standard. Thirdly, our approach is able to be supported by platforms. In fact, using set theory in the formalization makes it easy to be supported by proof mechanisms such as Rodin. Last but not least, our method is able to verify the consistency of other standards such as AUTOSAR OS timing protection and services protection. These standards are described based on the AUTOSAR OS structure. Thus, it is able to formalize them using set and relation in Event-B. Furthermore, the requirements of these standards contain system's behaviors. Hence, they can be directly formalized in Event-B instead of using the notions of the events like the memory protection requirements.

The limitation of our research is that the formalization is not fully automated. It requires human actions like analyzing and writing the formal specification and discharging proof obligations. In addition, verifying the specification is not sufficient to validate AUTOSAR OS memory protection. The implementation should be verified as well. Therefore, testing is needed to ensure that the implementation is protected against memory attacks. In future, we will propose a method to generate test cases for the implementation. These test cases are considered as attacks on the memory because the main purpose of memory protection is controlling access rights.

Acknowledgments

This work is supported by the project no. 102.03–2015.25 granted by Vietnam National Foundation for Science and Technology Development (Nafosted).

References

[1] AUTOSAR. (2017) About autosar. [Online]. Available: https://www.autosar.org/

[2] O. Group. (2005) Osek/vdx operating system specication 2.2.3.

[3] AUTOSAR. (2016) Specification of operating system.

[4] R. platform. (2012) Rodin user's handbook v.2.8. [Online]. Available: https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf

[5] M. Stilkerich, "Memory protection at option: application-tailored memory safety in safety-critical embedded systems (speicherschutz nach wahl)," Ph.D. dissertation, University of Erlangen-Nuremberg, 2012. [Online]. Available: http://www.opus.ub.uni-erlangen.de/opus/volltexte/2012/3969/

[6] J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin, *An Open Extensible Tool Environment for Event-B*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 588–605. [Online]. Available: https://doi.org/10.1007/11901433_32

[7] M. Bidoit, C. Chevenier, and C. Pellen, "An algebraic specification of the steam-boiler control system," 1996.

[8] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini, "Assisting requirement formalization by means of natural language translation," *Formal Methods in System Design*, vol. 4, no. 3, pp. 243–263, May 1994. [Online]. Available: https://doi.org/10.1007/BF01384048

[9] Y. Huang, Y. Zhao, L. Zhu, Q. Li, H. Zhu, and J. Shi, "Modeling and verifying the code-level osek/vdx operating system with csp," in *2011 Fifth International Conference on Theoretical Aspects of Software Engineering*, Aug 2011, pp. 142–149.

[10] J. michel Bruel, P. Chair, and S. E. Nasa, "Integrating formal and informal specification techniques. why? how?" 1998.

[11] R. B. France and M. M. Larrondo-Petrie, *A two-dimensional view of integrated formal and informal specification techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 434–448. [Online]. Available: https://doi.org/10.1007/3-540-60271-2_135

[12] I. D. Craig, *Formal Models of Operating System Kernels*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[13] D.-H. Vu and T. Aoki, "Faithfully formalizing osek/vdx operating system specification," in *Proceedings of the Third Symposium on Information and Communication Technology*, ser. SoICT '12. New York, NY, USA: ACM, 2012, pp. 13–20. [Online]. Available: http://doi.acm.org/10.1145/2350716.2350721

[14] D. Bertrand, S. Faucou, and Y. Trinquet, "An analysis of the autosar os timing protection mechanism," in *2009 IEEE Conference on Emerging Technologies Factory Automation*, Sept 2009, pp. 1–8.