Available online at www.ijpe-online.com

International Journal of Performability Engineering

Vol. 13, No. 7, November 2017, pp. 1111-1122
DOI: 10.23940/ijpe.17.07.p14.11111122

Automatic Generation of the AADL ALISA Verification Plan with ATL

Tianyi Wu[a], Zhiqiu Huang[a,*], Zhibin Yang[a], Tiexin Wang[a], Lei Xue[b]

[a]*Nanjing University of Aeronautics and Astronautics, 29 Jiangjun Road, Nanjing 211106, China*
[b]*Shanghai Aerospace Electronic Technology Institute, Shanghai 201109, China*

Abstract

Architecture Led Incremental System Assurance short for ALISA presents a method to check if a system implementation meets its requirements. This method helps find errors in the early phase of system integration. ALISA provides four notations—requirements specifications, architecture models, verification techniques and assurance cases. The verification plan, which is designed by extracting information from requirement specification and architecture model, can be executed on the system and the result is significant metrics to judge the system quality. There are problems when generating a verification plan. As for the hierarchical architecture model with increasing complexity, the system may be divided into several parts and it is difficult to accomplish the assurance manually for each tier of the architecture. The approach also needs to respond to the ever-changing demands rapidly. New faults may be introduced artificially when designing requirement specifications and verification plans. In the paper, we propose an approach which uses ATL, whose full name is ATLAS transformation language, to help automatically generate verification plans. The meta-model of the verification plan and of requirement specification are given. Thus, designing the transformation rules from the verification part to the requirement part is easy. A lightweight template described in ATL is used to generate the verification plan for critical requirement and quality property.

Keywords: ALISA; requirement specification; verification; ATL; model transformation template; AADL

(Submitted on July 25, 2017; Revised on August 30, 2017; Accepted on September 15, 2017)

(This paper was presented at the Third International Symposium on System and Software Reliability.)

1. Introduction

In the process of model driven development, requirement modeling and software architecture modeling are two key layers. These two critical layers are designed carefully and rigorously. Stakeholders define their goals while architects describe the corresponding system architecture with AADL, named Architecture and Analysis Design Language. The system is broken into parts as goals, and system requirements are designed by stakeholders and engineers respectively. There exists problems in the system integration process.

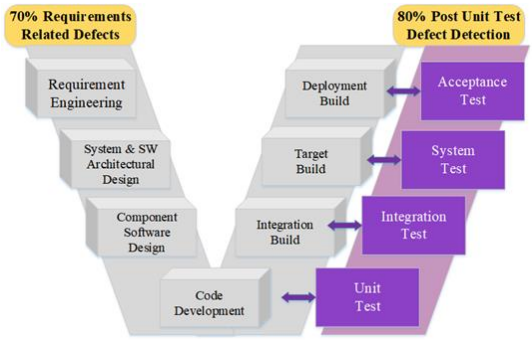

Figure 1. The Double-V model

* Corresponding author.
E-mail address: zqhuang@nuaa.edu.cn .

Obtained from the double-V model in Figure 1, studies have shown that 80 percent of implementation error could be detected when system designers integrate the system; nevertheless, 70 percent of them could be found earlier. If the relation between natural textual requirement specification and AADL architecture can be improved, the errors or faults will be fixed and handled earlier. To solve this problem, RF [4] introduces an approach called Architecture Led Incremental System Assurance, ALISA. It describes an incremental life-cycle assurance for high-reliant systems. ALSIA utilizes the architecture abstractions in the model to manage requirements across multiple layer of a system architecture and gives assurance result of a system implementation against its requirement.

ALISA presents four key pillars to assure requirements which are: requirement specifications, architecture model, verification method, and assurance cases. The requirement specifications is a document-based and architecture-led textual notation. The specification covers system interaction, design quality attributes and fault impact. The architecture model is mainly managed by architecture languages such as AADL. It presents a new medium to get access to the analysis on system designs through the whole system life cycle. The verification techniques are used to analyze system artifacts to check requirements satisfaction. Assurance cases show how systems can be taken into considerations and information of system quality can be provided. The target of assurance is multi-valued verification result measures and weighed requirement claims, verification activity results which reflect the importance.

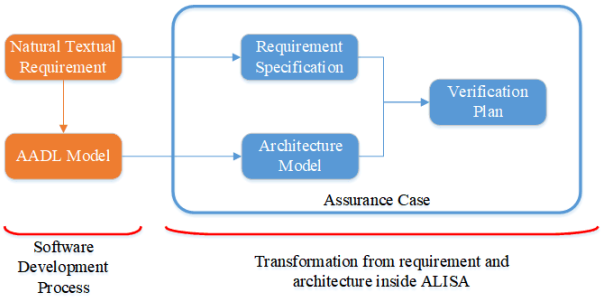

Figure 2. The paper framework

The framework presented in Figure 2 shows how ALISA works in a development process to execute a verification plan on a system. Once a natural informal requirement is given, it can be transformed to a formal requirement specification in the ALISA framework. AADL model can be directly used in the architecture model of ALISA. Considering the component type and implementation in the architecture or property set, information will be used to design verification plans executed on the system.

However, some problems in the transformation need to be fixed. With ALISA's one of the flavors which is working with one architecture layer at a time, a multi-tier system model can be divided into many layers or parts and AADL component are usually designed layer by layer. Otherwise, requirement specification and verification plan need a refactor in order to handle the ever-changing demands. Although the simplest way is to rewrite all the specifications, all the rewriting work requires a lot of effort, considering the size and complexity of a software system.

The assurance case includes all the artifacts in the development process. The correctness of assurance execution and assurance result is of much importance. The degree of requirements coverage and verification results are important metrics for assurance case. Requirement, model, and code are all involved to results.

In this work, we propose an approach that could automatically generate verification plans with the reference of requirement specifications and architecture model. This method applies ATL transformation language to meta-model level as ATL provides ways to produce a set of target models from a set of source models. We define a lightweight template for generating some regular verification plans. From the elements in these specifications, our method makes the transformation reusable over different requirement specifications. This method also helps automation of assurance for the multi-tier system.

2. Related Work

There has been much work that have tried to build a bridge between requirements by using the KAOS requirement model in Ponsard et al. [16]. By comparing and evaluating the existing method, Galster et al. [10] presents a new way to achieve the transition between requirement and architecture. An approach that transforms UML requirement model to AADL model is described in Zeng et al. [22]. The SysML in Albinet et al. [1] and model to model in Drivalos et al. [6] present traceability techniques.

The approach described in Goknil et al. [11] acts on AADL software architecture and is based on verifying functional requirements and uses Maude model checker. Blouin et al. [2] defines a way to link requirement and AADL element and ALISA also has the similar meta-model with this tool. Feiler [8] gives the process in which stakeholder goals are turned into verifiable system requirement specifications by annotating an AADL model. Singhoff et al. [19] presents an Ada tool used for resource requirements analysis of AADL. Whalen et al. [21] and Schätz et al. [17] are the most similar approach with ALISA.

The possibility of XMI-based generic transformations of UML models is examined in Jiménez et al. [13]. Cuadrado et al. [3] gives an approach to present the generic programming transformation template for reusable. Kovse et al. [14] provides a method to make the model-driven development of RubyTL model transformations easier. Lara et al. [15] also designs transformation by using template binding concept to meta-model.

3. ALISA Language

3.1. Introduction of AADL

The Architecture Analysis & Design Language (AADL) was approved and published as SAE Standard AS-5506 in November 2004. AADL is an architecture language for the specification, analysis, automated integration and code generation when describing real-time performance-critical computer systems. In AADL model, components are designed through type and implementation declarations as indicated by Feiler et al. in [9]. An AADL system model has the capacity to describe the architecture and runtime environment of an application system. Its component software and execution platforms usually for hardware components are described by component or connection. AADL support mode change to express the evolution of system. AADL provides annex for core language of AADL such as error annex or behavior annex to extend the description capacity of AADL. In AADL tool set environment, textual specification, xml specification and graphical notation are all supported.

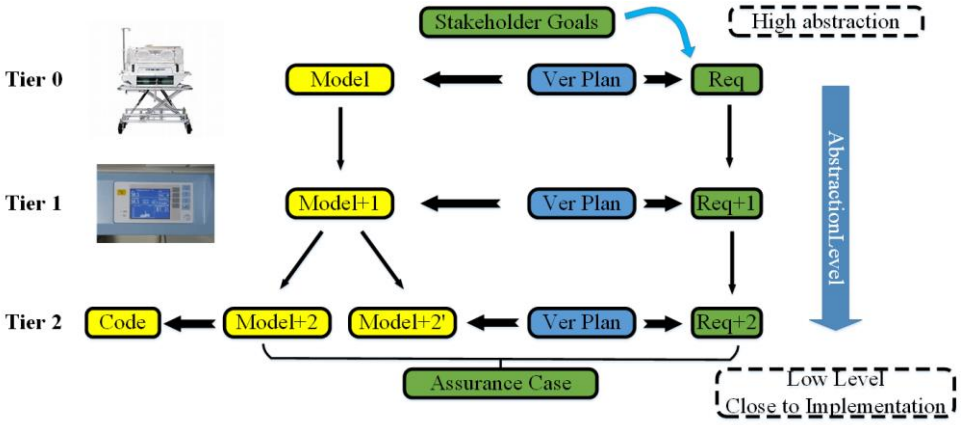

Figure 3. ALISA framework with example of isolette

3.2. Introduction of ALISA Language

The whole ALISA framework is shown in Figure 3. The ALISA method can work on a multi-tier system such as the example in Figure 3. From top level to the bottom, the lower tier is closer to implementation. Tier 0 indicates the whole isolette system. Tier 1 describes the operational interface as a subsystem in isolette. And tier 2 indicates a more specific implementation level. Every tier can be refined towards a more detailed level. The ALISA framework is mainly described by the assurance case.

ReqSpec are described from stakeholder goals in Figure 2. ReqSpec is described in Delange et al. [5], which introduces a requirement specification Language, short for ReqSpec. The purpose of ReqSpec is to support the requirements elicitation, definition, and modeling for real-time embedded systems.

In Figure 2, a verification plan is described in notations ‘Ver Plan’, which acts as a connection between the AADL model and requirements. After the verification plan is well designed, a system requirement can be verifiable by annotating an AADL model of the system of interest in its operational environment and, as appropriate, elements of the system architecture.

A verification plan specifies how every requirement of a system set is verified. Verification activities are performed on an artifact representing the system implementation. This artifact is an AADL model representing the architecture specification of the system, or accessible from the AADL model, such as detailed design models. This file declares verification methods that are contained in the verification activities. A verification method registry allows users to specify the type and signature of a verification method as well as reference to its implementation in an implementation language neutral way. In other word, a verification method registry file is the link between the verification method declaration and its method implementation by a variety of forms like Osate analysis, Resolute, Junit4 and Java/Xtend.

The ALISA notation lets users specify assurance cases and assurance plans, which means that assurance instance is based on requirement specification, verification plan and architecture model. An assurance case configures how a system is to be assured. An assurance plan allows users to focus on a subset of the requirements and verification activities at a time by specifying a set of filter criteria in terms of category labels.

4. ATL Transformation Rules

4.1. Assumption of the transformation

In this paper's case, suppose that only the system requirements have been taken into consideration, owing to in the existed requirement template that is designed by engineers. Due to the feature of ALISA framework, only the critical requirement or quality property will be considered to implement the automatic analysis. Also, only the first three parts in ALISA framework is considered and they are requirements specifications, architecture models and verification techniques. So, we can focus on the automatic generation problems. More than that, suppose that the AADL model has been built to support the invoking of the AADL property or some AADL components.

4.2. ATL Transformation Framework

ATL, the Atlas Transformation Language, is a model transformation language specified both as a meta-model and as a textual concrete syntax in RF [12]. It is a hybrid programming way both of declarative and imperative.

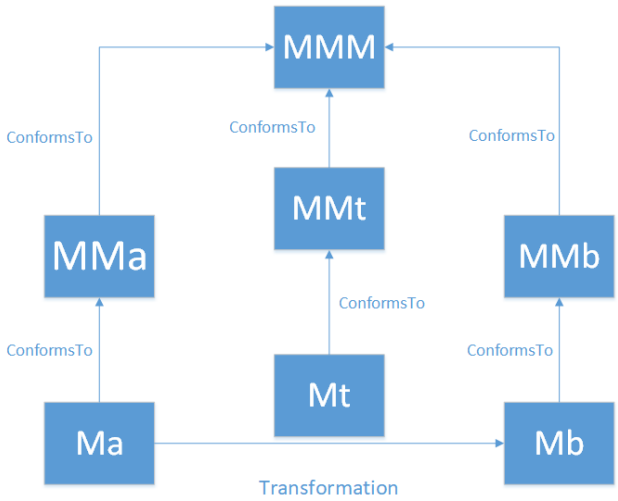

Figure 4. ATL transformation from Ma to Mb

The ATL transformation framework is presented in Figure 4. ATL mainly concentrates on the model-to-model transformation. This diagram summarizes the complete transformation process. ATL makes it possible to specify the process that one or more target model can be produced from a group of source models.

An ATL program includes rules, queries and libraries. Rules define how elements in source model are mapped and lead to elements of the target models. The aim of a query is to compute a primitive value, such as a string value or a numerical value from source models. The way that independent ATL libraries can be imported from different types of ATL units provides a convenient way to factorize ATL code.

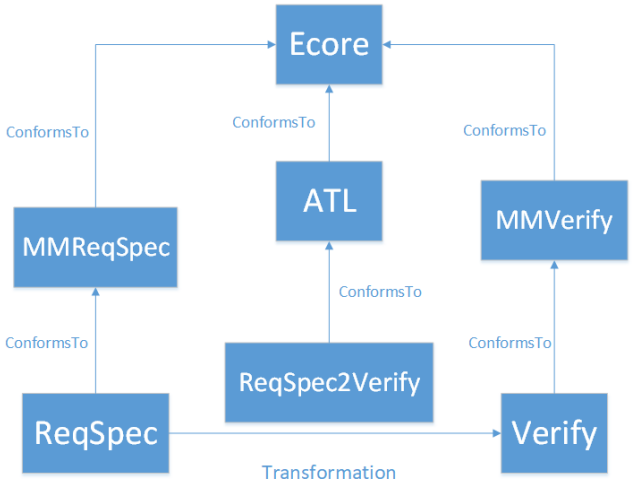

Figure 5. ATL transformation from ReqSpec to Verify

The ReqSpec meta-model that we provide to perform the transformation is a subset of the ReqSpec defined in Delange et al. [5]. The transformation shown in Figure 5 describes the whole transformation is similar with the transformation in Figure 4. The ReqSpec can be transformed to Verify automatically by designing the ATL module named 'ReqSpec2Verify' and execute the ATL module in ATL engine. All these operations are based on the meta-model.

4.3. ATL Transformation Framework

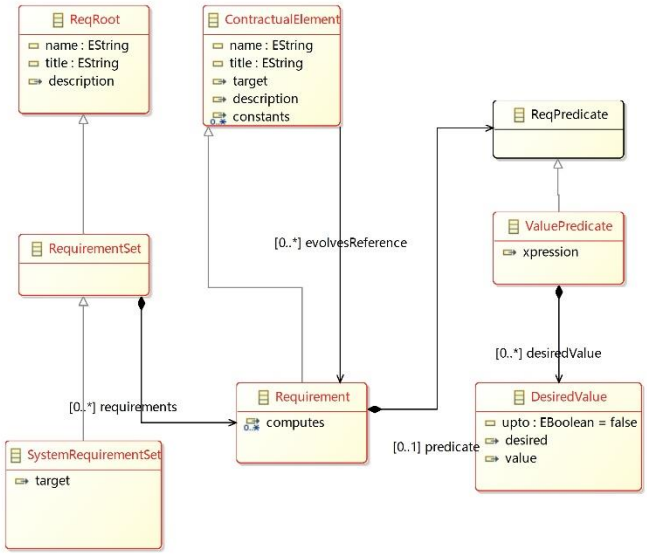

Figure 6. Subset of ReqSpec meta-model

The subset meta-model of ReqSpec is described in Figure 6. The ReqRoot class contains the basic information of ReqSpec. The SystemRequirementSet class inherits from the RequirementSet as well as RequirementSet class inherits from ReqRoot. ReqSpec totally consists of a requirement set. A requirement set contains multiple requirements. Each requirement may be specified to a subcomponent and predicates or variables defined in requirement block.

The subset meta-model of ReqSpec is explained as following:

- The name attribute is the identifier of the RequirementSet.
- The title is the short description.
- The description attribute of ReqRoot class is a textual description of the system requirements for a specific system.

- The target attribute is a component in the AADL architecture. This element specifies the system requirements for the particular component of the whole system.
- The Requirement class which inherits from the ContractualElement class is specified as a single requirement in a RequirementSet. The attributes like name, title, target and description defined in the ContractualElement is detail for each requirement.
- The constants attribute acts as parameters and then can be referenced by Predicate. Their value could be value expression that may be ranges, string, Boolean as well as numeric values with a unit defined in the AADL property set.
- The computes attribute declaration allow user to introduce the name of such variables explicitly. And they can be referenced in the predicate declarations.
- The predicate class is used to compare the constants defined by the user and the computed variable.

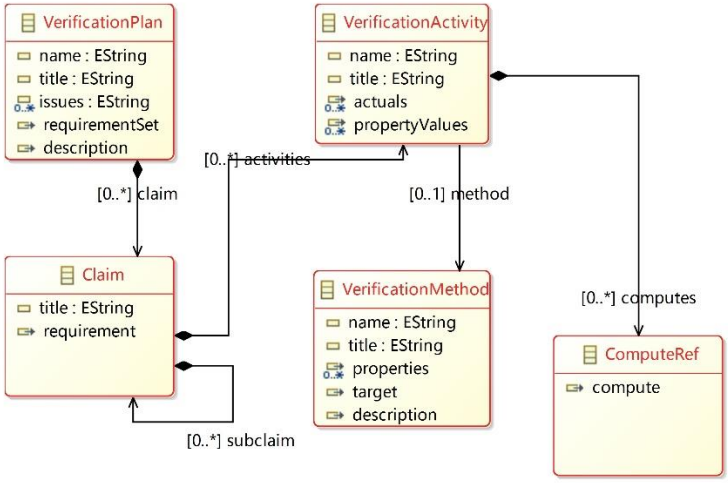

Figure 7. Subset of Verify meta-model

The subset meta-model of Verify part is described in Figure 7. The Verify notation allows the architects to design the way that describes how requirements are verified. Each Verify file contains a VerificationPlan that is specified by users according to the requirement specification and architecture model. And each VerificationPlan contains Claims which consists of VerificationActivity declaration in which architects can specify VerificationMethod declaration.

The subset meta-model of Verify is explained as following:

- The VerificationPlan class is specified for a SystemRequirementSet. The name attribute is the reference of the VerificationPlan. And the title is a short description for the declaration. The requirementSet attribute is the reference of the corresponding requirement.
- The Claim class is the representation of the set of verification activities that should be executed successfully and this means the corresponding requirement is met.
- The VerificationActivity class specifies the verification method which will be used to verify the requirement in the next assurance. The actual attribute is the parameters that can be called in the assurance phase. The propertyValues attribute is reference which comes from the AADL property set defined by users.

4.4. ATL Transformation Rules

According to the two meta-models described in Figure 8, the name attribute of the requirement set is specified by users. This name label is also used in the VerificationPlan declaration to get the reference of the corresponding requirement set. Then binding these two attributes can be defined in ATL rule. The name attribute of SystemRequirmentSet class will be automatically transformed to VerificationPlan name.

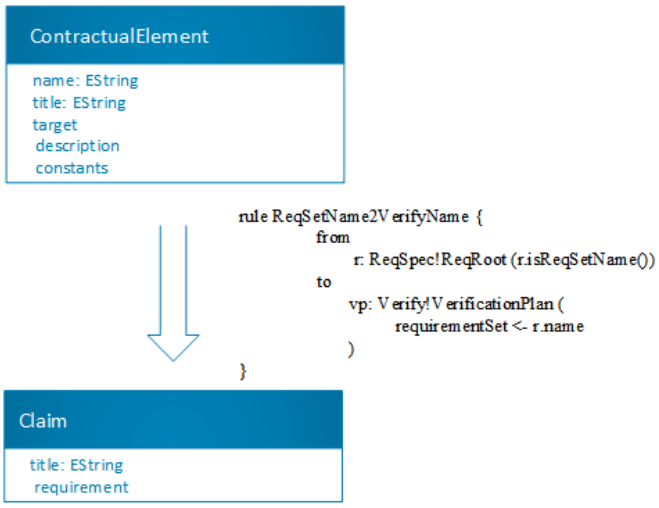

Figure 8. ATL rule from ReqSpec name to VerificationPlan name

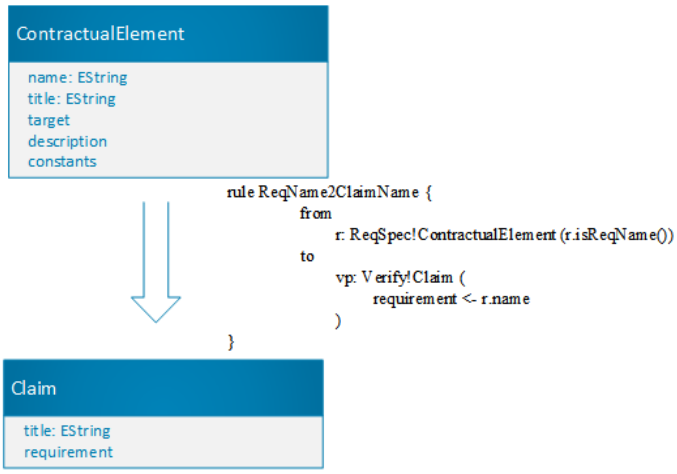

Figure 9. ATL rule from ContractualElement name to Claim name

The transformation in Figure 9 is from the name of each requirement in the whole requirement set construct to the claim name attribute of the VerificationPlan. The ContractualElement class is the super class of the requirement class. The Claim class and the VerificationPlan class are association and composition in meta-model. As a result, the reference of each single requirement can be transformed to the attribute in VerificationPlan.

This paper proposes a lightweight template based method for transformation from ReqSpec to Verify. Due to the feature of the ALISA framework, the critical property should be considered as a high priority. For a safety-critical system, there must be some important quality that takes precedence when these properties are specified in requirement part. The ReqSpec is to introduce these critical properties into ALISA framework. So, a template is designed to achieve the process of automatically generating Verify from ReqSpec and architecture model.

In Figure 10, Requirement class, which contains the compute attribute, is transformed to VerificationActivity class that is associated to the VerificationMethod. The VerificationMethod class is a type in the VerificationActivity.

The ‘using’ section in ATL rule means that they are local variables for using just in this rule declaration. Those variables can be used in the ‘to’ and ‘do’ section of ATL rule. In the using section, verification activity name declaration and the verification method name declaration are defined in advance. The method name called ‘hasproperty’ can be implemented by the method reference ‘Alisa_Verification.GetProperty’ set in the predefined project named ‘Alisa_Verification’. This method aims to assign the name to computed variable that defined by user in the ReqSpec section. If the architecture model does not provide a computed variable with a property set with a value or range, the variable will be assigned with an initial default value. Totally, this method helps to get the critical property in the architecture model.

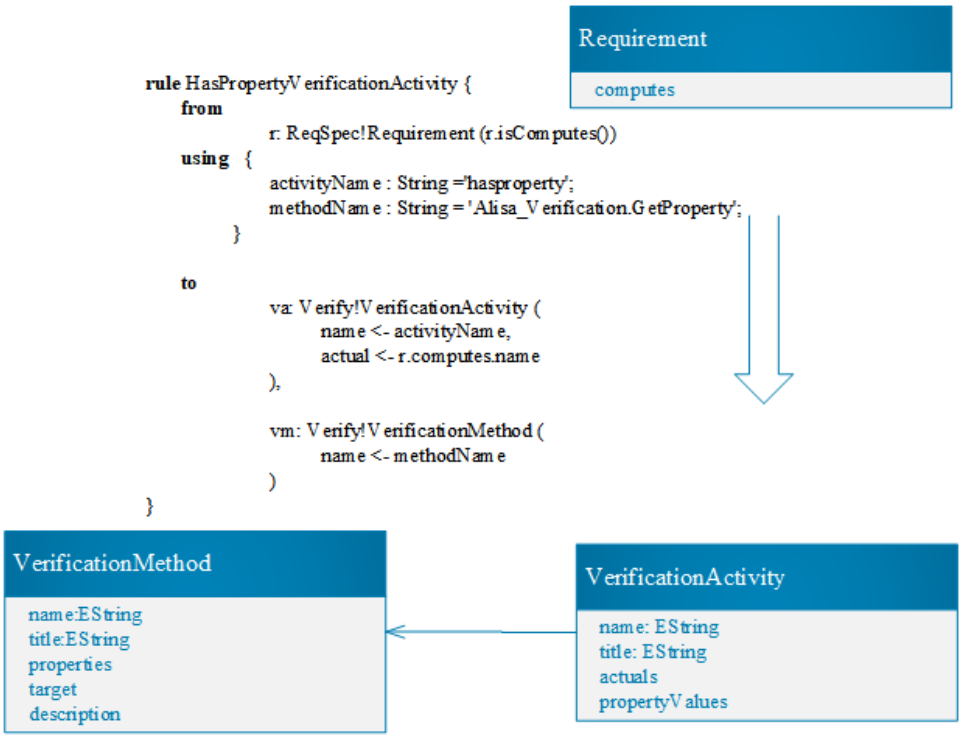

Figure 10. ATL rule to generate VerificationActivity named 'hasproperty'

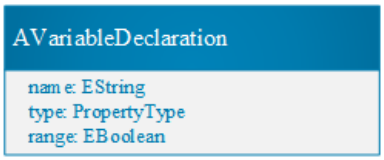

Figure 11. AVaribaleDeclaration meta-model

The parameter of the verification method is from compute declaration. AVariableDeclaration class is shown in Figure 11 which is the type of compute attribute. So, actual parameter of the VerificationMethod class can be obtained from the name attribute in the AVariableDeclaration class.

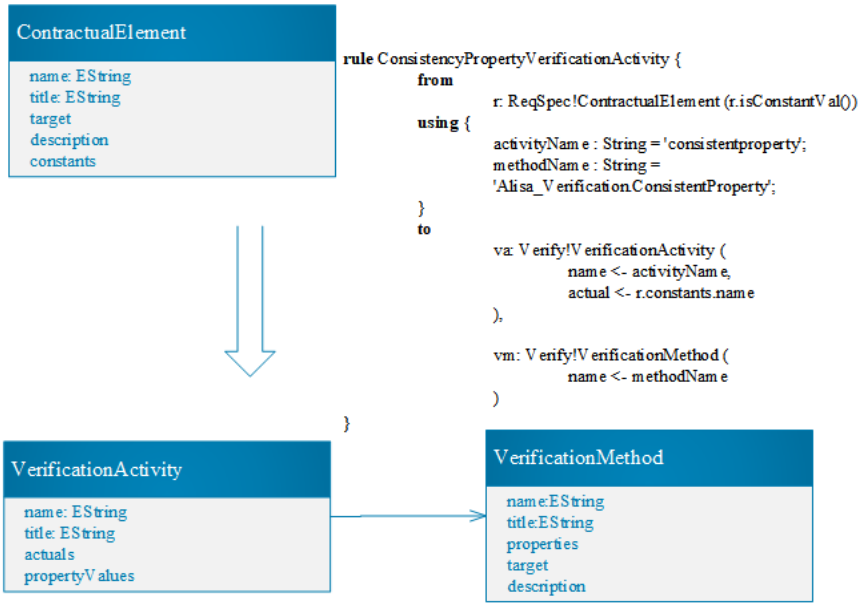

Figure 12. ATL rule to generate VerificationActivity named 'consistentproperty'

The ATL rule describes the transformation from ContractualElement class to VerificationActivity class and VerificationMethod class shown in Figure 12. The transformation is similar to the last transformation. In this transformation, another verification method is defined which is declared as the name 'consistentproperty'. This verification activity is to define a method to help check if the property is consistent with the property of the feature component. In the implementation phase, the method will return a Boolean type to report success or failure.

The ATL rule in Figure 12 is from ContractualElement class to VerificationActivity and VerificationMethod. The activity name and method name is the predefined section. The parameter in this verification method comes from constants defined in the requirement specification part. The constant type also has the same meta-model like compute type has. Its type is AvariableDeclaration class too. And the name attribute is the constant reference and can be obtained from the meta-model diagram.

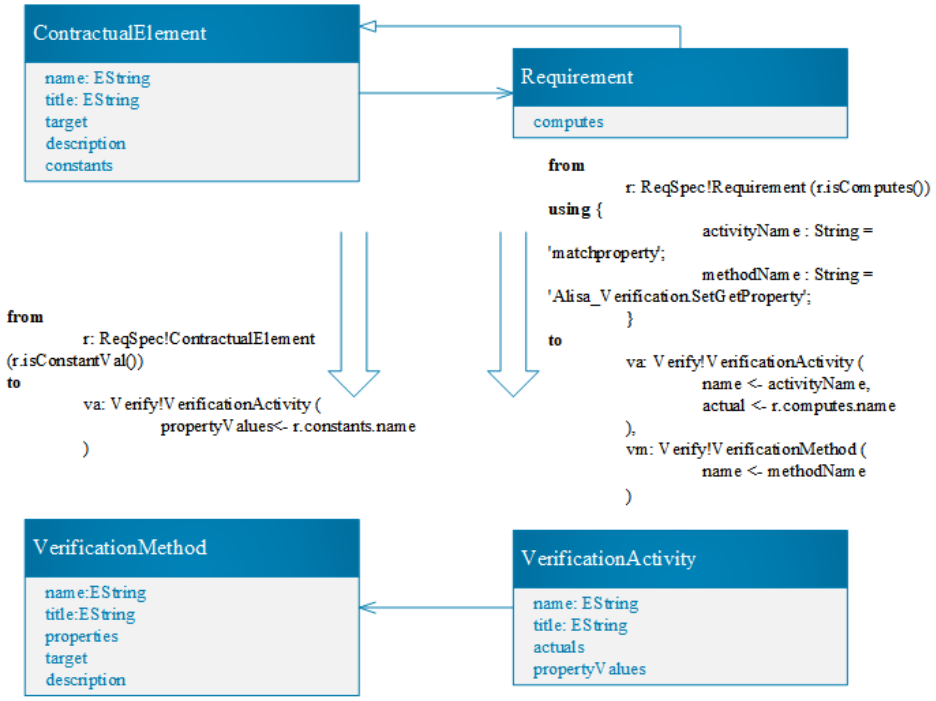

Figure 13. ATL rule to generate VerificationActivity named matchproperty

In Figure 13, the VerificationActivity and VerificationMethod are transformed from ContractualElement and requirement respectively. That arrow between VerificationActivity and VerificationMethod means these two classes are association which indicates that each VerificationActivity has one VerificationMethod. Similarly, ContractualElement and requirement have the same relationship, and requirement inherits from ContractualElement.

In some cases, the verification method expects the values it operates on to be available as property values in the model. We can specify this fact as part of the verification method registration. So, we register a new method 'SetGetProperty' that will set the property value that is specified as part of a call. For this situation, we define an activity named 'matchproperty' for the method.

After the activity name and method name are defined, the right part provides the transformation from ContractualElement class to VerificationMethod class to get the computed variable reference. The meta-model in Figure 11 is the meta-model of AVariableDeclaration which is the concrete type of compute attribute. According to the meta-model, the computed variable reference can be obtained from the name attribute.

The left part of Figure 13 is also a part of the same transformation. The propertyValues is an attribute that is an AADL property indeed. Currently this attribute is limited to reference to constant variable. The propertyValues declaration is transformed from constants attribute. Due to the constants' parent type, there is also a AVariableDeclaration class so that the transformation is easy to know.

5. Case Study

In order to illustrate how the Verify file can be generated automatically. This section will provide an example to introduce process while the ReqSpec file is set up manually and the Verify file is generated automatically.

Here is an example from the appendix A of the RF [20]. The annex is requirement specification for Isolette Thermostat which is an incubator for an Infant. This device can provide controlled temperature, humidity, and oxygen (if necessary). The operational context of the Isolette Thermostat is shown in Figure 14.

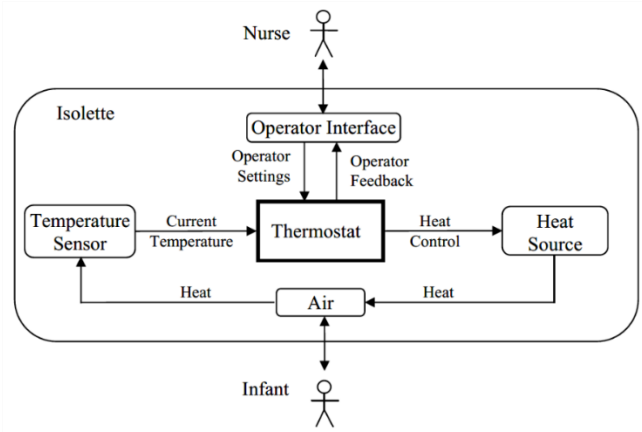

Figure 14. Context Diagram for the Isolette Thermostat

This figure totally describes a thermostat system of an isolette. The temperature sensor will sense the current air temperature and the thermostat will control the heat source based on the information from senor. Nevertheless, if the current temperature is beyond the desired range, it will give out an alarm to alert the nurse. The nurse could set the desired temperature range and turn up or turn down the temperature.

This annex provides the detailed natural language requirement for each subsystem of the Isolette Thermostat. Here is one of the requirements for the Operator Interface subsystem:

- EA-OI-3: The Lower Alarm Temperature will always be ⩾93° F.

This is an environmental assumption that the lower bound of alarm temperature must always be larger than 93 Fahrenheit.

The next step is to set up the ReqSpec file manually and the imperative part is to specify EA-OI-3 by using ALISA requirement specification language. The ReqSpec textual model is showed as follow:

Table 1. ReqSpec specification for EA-OI-3

```
system requirements reqs for isolette::thermostat_th
[
    requirement R1 : "lower alarm temperature" for lower_alarm_temperature
    [
        val tem = 93 Fahrenheit
        compute actualtem: iso_variables::lower_alarm_temperature
        value predicate actualtem >= tem
    ]
]
```

The code in Table 1 shows how the EA-OI-3 is described in the ReqSpec. Firstly, a system requirement set construct is named ‘reqs’. And the name of the requirement set will be a reference when designing verification plan. The ‘isolette::thermostat_th’ actually is a thread component in AADL model. It identifies the target of the requirement on a thread. The ‘R1’ is the name reference for a single requirement in the requirement set. And this ‘R1’ requirement is for ‘lower_alarm_temperature’ which is a data type feature in Isolette architecture model which can be seen as the target classifier of requirement R1. The ‘tem’ is the constant defined by user. The ‘actualtem’ is introduced from the AADL property set. The property type is described in Table 2 from Senn et al. [18]. The predicate section of the requirement gives

the capacity to support the requirement formalization. The predicate should be satisfied as part of a verification activity in a verification plan. The result will show if the requirement is met. It is worth mentioning that the actual temperature will be compared against the expected temperature. The operators like, and, or, not, and others are in detail provided in Expression Notation in Delange et al. [5].

Table 2. AADL property set for lower alarm temperature

```
data lower_alarm_temperature
properties
        Data_Model::Data_Representation => Struct;
        Data_Model::Element_Names => ("t");
        Data_Model::Base_Type => (classifier (iso_variables::lower_alarm_range));
        BLESS::Typed => "record (t:lower_alarm_range)";
end lower_alarm_temperature;
```

Table 3. Verify specification for EA-OI-3

```
verification plan vplan for reqs
[
claim R1  [
    activities
      hasproperty:          actualtem          =
Alisa_Verification.GetProperty()
      consistentproperty:
Alisa_Verification.ConsistentProperty(tem)
    ]
]
```

Based on the maps defined in ATL module, the ReqSpec can be transformed to Verify as show in Table 3. The verification plan named ‘vplan’ refers to the requirement sets named ‘reqs’ which is the particular requirement set defined above so that the verification plan will work if the corresponding requirement specification is in the same project with it. The claim ‘R1’ refers to the requirement ‘R1’ likewise. After the ATL transformation, there are two verification activities, including two verification methods, respectively, for verifying. The ‘hasproperty’ activity is bound to a verification method ‘GetProperty’ which returns a real value that gets bound to actual temperature value and sets the initial value in case it fails. The ‘consistentproperty’ activity imports a parameter defined in the previous phase to check the consistency with the corresponding property in architecture model. And the method bound to this activity will return a Boolean value to report the verification result.

The ATL transformation module has been developed into a plug-in tool deployed in the OSATE tool set environment described in Feiler [7]. By using the tool to parse ReqSpec file, which specifies a critical property for a feature or a component in AADL, the corresponding Verify file can be generated automatically. This tool is based on the ATL technique, though the subset of ALISA is supported, the method is extendable for users to add full ALISA meta-model based ATL rule.

6. Conclusions

In the paper, we proposed an approach to generate verification plan from requirement specification automatically based on the ALISA method. This approach generates particular verification activity and verification plan for the critical requirement or quality property by defining a lightweight template. The method can help the automation of assurance execution and improve the correctness of assurance case when dealing with the multi-tier system and ever-changing demands.

Our method is implemented by ATL technique. Though only a part of the meta-model is taken into consideration, the ATL module could be extended to fit more of the ALISA meta-model for verification plan generation.

Acknowledgements

This work was supported by the graduate student innovation base (laboratory) open fund project of Nanjing University of Aeronautics and Astronautics, kfjj20171606 and the National High Technology Research, the National Natural Science Foundation of China (61502231). Natural Science Foundation of Jiangsu Province (BK20150753). The Project of the State Key Laboratory of Software Development Environment of China (SKLSDE-2015KF-04). The Avionics Science Foundation of China (2015ZC52027).

References

1. A. Albinet, J. L. Boulanger, H. Dubois, M. A. Peraldi-Frati, Y. Sorel, and Q. D. Van, "Model-Based Methodology for Requirements Traceability in Embedded Systems," 2007.
2. D. Blouin, E. Senn, and S. Turki, "Defining an Annex Language to the Architecture Analysis and Design Language for Requirements Engineering Activities Support," in *Model-Driven Requirements Engineering Workshop*, 2011, pp. 11-20.
3. J. S. Cuadrado, E. Guerra, and J. D. Lara, "Flexible Model-to-Model Transformation Templates: An Application to ATL," in *International Conference on Prostate Cancer Imaging: Image Analysis and Image-Guided Interventions*, 2011, pp. 70-79.
4. J. Delange, P. Feiler, and N. Ernst, "Incremental Life Cycle Assurance of Safety-Critical Systems," in *Ertss*, 2016.
5. J. Delange, P. Feiler, and L. Wrage, "A Requirement Specification Language for AADL," 2016.
6. N. Drivalos, R. F. Paige, K. J. Fernandes, and D. S. Kolovos, "Towards Rigorously Defined Model-to-Model Traceability," in *Ecmda Traceability Workshop Sintef Technical Report*, 2008
7. P. Feiler, "Open Source AADL Tool Environment (OSATE)," *Aadl Workshop*, 2004.
8. P. Feiler, "Requirements and Architecture Specification of the Joint Multi-Role (JMR) Joint Common Architecture (JCA) Demonstration System," 2015.
9. P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The Architecture Analysis & Design Language (AADL): An Introduction," 2006.
10. M. Galster, A. Eberlein, and M. Moussavi, "Transition from Requirements to Architecture: A Review and Future Perspective," in *Acis International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/distributed Computing*, 2006, pp. 9-16.
11. A. Goknil, I. Kurtev, and V. D. B. Klaas, "A Rule-Based Approach for Evolution of AADL Models Based on Changes in Functional Requirements," in *Proccedings of the European Conference on Software Architecture Workshops*, 2016, p. 10.
12. A. Group, "ATL: Atlas Transformation Language," 2006.
13. Á. Jiménez, D. Granada, V. Bollati, and J. M. Vara, "Using ATL to support Model-Driven Development of RubyTL Model Transformations," in *Proceedings of the third International Workshop on Model Transformation with ATL*, pp 35-48, 2014.
14. J. Kovse and T. Härder, "Generic XMI-Based UML Model Transformations," *Lecture Notes in Computer Science*, vol. 2425, pp. 192-198, 2002.
15. J. D. Lara and E. Guerra, "Towards the flexible reuse of model transformations: A formal approach based on graph transformation," *Journal of Logical & Algebraic Methods in Programming*, vol. 83, no. 5-6, pp. 427-458, 2014.
16. C. Ponsard and M. Delehaye, "Towards a Model-Driven Approach for Mapping Requirements on AADL Architectures," in *IEEE International Conference on Engineering of Complex Computer Systems*, 2009, pp. 353-358.
17. B. Schätz, V. Aravantinos, S. Voss, S. Teufl, and F. Hözl, "AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems," in *International Workshop on Model-Based Architecting of Cyber-Physical and Embedded Systems*, 2015.
18. E. Senn, D. Blouin, and S. Turki, "AADL Requirements Annex Explored With FAA Handbook Example," 2012.
19. F. Singhoff, J. Legrand, and L. Nana, "Scheduling and memory requirements analysis with AADL," *Acm Sigada Ada Letters*, vol. XXV, no. 4, pp. 1-10, 2005.
20. Springfield, "Requirements Engineering Management Handbook."
21. M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. E. Heimdahl, and S. Rayadurgam, "Your "What" Is My "How": Iteration and Hierarchy in System Design," *IEEE Software*, vol. 30, no. 2, pp. 54-60, 2013.
22. Y. W. Zeng and M. L. Zhang, "An Approach about Translating from Requirement Model to AADL Software Architecture," in *International Conference on Information NETWORKING and Automation*, 2010, pp. V1-275-V1-279.