Composition of AADL Components by Transformation to Interface Automata

Jiangwei Li[1,2], Jizhou Zhao[1,2], Qingqing Sun[1,2], Xiaopu Huang[1,2], Yan Zhang[3], and Tian Zhang[1,2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University
[2] Department of Computer Science and Technology, Nanjing University, Nanjing, P.R. China
{ljw,zjz,sunqq,hxp}@seg.nju.edu.cn, ztluck@nju.edu.cn
[3] Department of Computer Science and Technology, Beijing Electronic Science and Technology Institute, Beijing, P.R. China
zhangyan@besti.edu.cn

Abstract. AADL, an industrial standard in embedded field, is a component-based semi-formal modeling language. Incompatibility of behaviors is a problem that we must face up with when the AADL components composite, because the sequence of some interactive activities may not match with each other. Shielding the incompatible behavior and reusing the compatibly behavior maximally are main problems to increase the reusability of AADL components. This paper proposes an MDE based method to implement the transformation from AADL to IA using the heterogeneous model transformation framework. Then we can use the IA model to derive available behavior all out from incompatible component compositions through construct the environment, and now the environment maps back to AADL component to solve the AADL components composition problems we proposed.

Keywords: AADL, Interface Automata, components composition, model transformation.

1 Introduction

With the development of embedded system, the complexity of embedded software continuously increases. The traditional development method cannot adapt to the requirement. MDE [1], proposed by OMG, is a software development framework, which highlights the usage of models. MDE technology have been introduced to the embedded software development, so the developers have to consider the correctness of the software model, and then lots of problems will be found and solved at the early stage of software development. Then the development cycle will be shortened and development cost will be reduced.

SAE (Society of Automotive Engineers) presented the real time embedded system model language---AADL (Architecture Analysis and Design Language) [4] at 2004

Y. Yuan, X. Wu, and Y. Lu (Eds.): ISCTCS 2012, CCIS 320, pp. 522–529, 2013.
© Springer-Verlag Berlin Heidelberg 2013

and it was released as SAE AS5506 Standard. AADL, a component-based semi-formal modeling language, supports software model, hardware model and NFP (non-function property) analysis. Because of simple grammar and extensible annex, AADL has been supported by many organizations. At 2006 and 2011, SAE released AS5506/1[5] and AS5506/2[6] AADL annex to complement AADL specification. The AADL components behavior description was presented in the AS5506/2.

As a components-based model language, AADL has to face the components composition problem which decides whether the components can be composited or not. Many ideas to solve this problem are to give up the incompatible components or to construct the interface wrapper for them; however, these ideas will bring the development cost increasing and development cycle extending problems. IA (Interface Automata)[7] is proposed to solve these problems. As a formal modeling language, IA uses an optimistic approach to solve components composition problems. In our early study [2], we had given an IA-based method to utmost reuse the available behavior of two incompatible components by constructing an environment for them.

In this paper, components are described by AADL, and we transform AADL models to IA models using ATL (Atlanmod Transformation Language) [8, 9]; IA is used to verify the components composition problems, and then we use the method given in[2] to construct an environment for two incompatible IA. Finally, the environment is mapped back to AADL models, and then the AADL components composition problems and components behavior compatibility problems are solved.

The paper is structured as follows: Section 2 introduces the AADL and Interface Automata simply. Section 3 describes the approach of transforming AADL components models to IA models. In Section 4, we present a case study on the approach. The concluding remarks are shown in Section 5.

2 Background

2.1 AADL

Architecture Analysis and Design Language (AADL) is a kind of architecture design language based on MDA. AADL can be applied in the field of embedded software system.

There are three kinds of components in AADL: software components, execution platform components and system components. Components are defined through type and implementation declarations. A component type declaration defined a component's interface elements and externally observable attributes. A component implementation declaration defines a component's internal structure in terms of subcomponents, subcomponent connections, subprogram call sequences, modes, flow implementations and properties.

In the AS5506/2 annex, the behavior specification of components is presented for the first time. The behavior specifications can be attached to any AADL components types and components implementations using an annex subclause. When defined within component type specifications, it represents behavior common to all the associated implementations. If a component type or implementation is extended, behavior

annex subclause defined in the ancestor are applied to the descendent except if the later defines its own behavior annex subclause.

The detailed description and examples of the AADL behavior have been provided in the AS5506/2 annex, so we will not provide in this paper.

2.2 Interface Automata

[7] presents the Interface Automata which was a new theory to describe interface at 2001. It is different from other theories. There are two main features, one is optimistic approach and the other is game thinking. The former feature is used to solve the problem of interface compatibility, the latter feature to describe the semantics of this problem. The theorems and definition of IA will not be shown in this paper because IA has been explained in detail in [7].

3 AADL2IA Transformation

In the practical application, components composition problems are ubiquitous. If we can find out and solve the components composition problems in the modeling stage of software development, the development time and cost will be reduced. AADL is components-based modeling language; however, it is not a formal modeling language. To solve the components composition problems of AADL, we transform AADL models to the IA models which are easy to verify the components compatibility.

3.1 Transformation Framework

The transformation framework is described in Fig.1 [10]. It shows the general model transformation process [3]: from the source model Ma, conform to the MMa (meta model of the Ma), then conform to the MMM (meta-meta model of the Ma). In the M3 level, we use the ecore as the meta-meta modeling language to describe the AADL meta-model and IA meta-model, and then we use the mapping rules to complete the transformation.

3.2 AADL and IA Meta-model

We adopt the AADL meta-model given by AS5506/1 annex, since it has contained all the AADL components. The meta model of IA in Fig.2, is designed by ourselves using EMF (Eclipse Modeling Framework).

3.3 Transformation Mapping Rules

To complete transformation from AADL models to IA models, Table 1 has given the main mapping rules from AADL to IA.

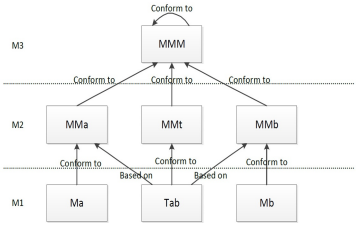

Fig. 1. Model Transformation Framework

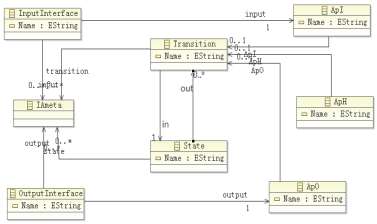

Fig. 2. IA meta-model

4 Case Study

4.1 Scenarios Description

We illustrate the feasible of the proposed rules with an example which describes the preparation work of docking of spaceship and space station simply.

Table 1. Transformation Mapping Rules

AADL components info	IA info
single component	an interface automaton
component features	IA ports
component states	IA state set V_P
component transitions	IA transition setT_P
component transitions Guard	IA input action set A_P^I
component transitions Action	IA output action set A_P^O
……	……

4.2 Model with AADL

To model this system easily, we consider the space station as *Space* thread, spaceship as *Ship* thread components, the *Space* thread and the *Ship* thread composite the process A. In the model, some components declarations have been removed due to space limitations.

The Model of The System.

```
thread Ship
  features
    msg: in event data port ;
    ack: in event port;
    nack: in event port;
    send: out event data port;
    ok: out event port;
```

```
    fail: out event port;
  --Snip
end Ship;

thread implementation Ship.impl
  annex behavior_specification {**
   states
  0:initial complete final state;
  2,3:complete state;
  1,4: state;
   transitions
  0-[on dispatch,msg]->1;
  1-[]->2{send!("send")};
  2-[on dispatch,ack]->3;
  2-[on dispatch,nack]->4;
  3-[]->0{ok!("ok")};
  4-[]->0{fail!("fail")};
  **};
  --Snip
end Ship.impl;

thread implementation Space.impl
  annex behavior_specification {**
   states
  v0:initial final state;
  v1:complete state;
   transitions{**
  v0-[]->v1{msg!("msg")};
  v1-[on dispatch,ok]->v0;
  **};
  --Snip
end Space.impl;

process implementation A.impl
  subcomponents
    SpaceA: thread Space.impl;
    ShipA: thread Ship.impl;
  connections
    A1: event port        ShipA.ok -> SpaceA.ok;
    A2: event port        ShipA.fail -> SpaceA.fail;
    A3: event data port   SpaceA.msg -> ShipA.msg;
  --Snip
end A.impl;
```

4.3 Model Transformation to IA

The AADL models we have created can transform to the IA models according to the transformation mapping rules. We get the IA models of *Ship* and *Space* in Fig.3 finally.

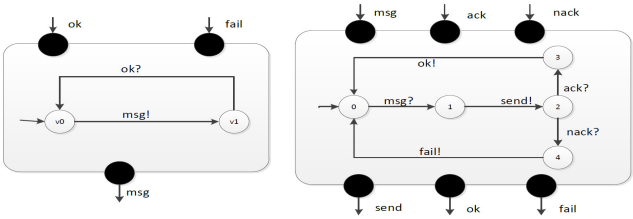

Fig. 3. Space station and spaceship IA models

4.4 Verification and Environment Construction

According to the interface automata composition definition [7], we can give the interface automaton C (Fig.4) which is the composition of the *Ship* and *Space* interface automata. In the Fig.4, there is an illegal state $(v_1, 4)$ can be reached, according to the interface automata compatibility definition [2], the interface automata *Ship* and *Space* are incompatible. This problem has been solved in our early research [2], and then we can use this method directly.

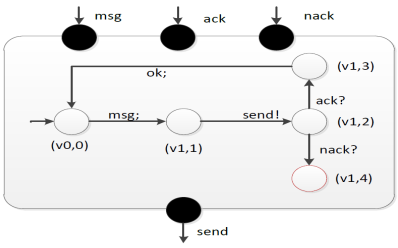

Fig. 4. Maximum legal environment user

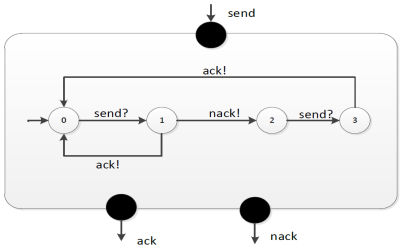

Fig. 5. Interface Automaton C

According to the construction algorithm of maximum legal environment shown in [2], we can construct maximum legal environment of the interface automata *Ship* and *Space*, named *User* in Fig.5.

The interface automaton space $\otimes$ ship $\otimes$ user is closed, definite and nonblocking in Fig.6. In other words, the behavior incompatible components spaceship and space station can work compatibly in the environment user.

Finally, the environment user maps back to AADL thread component which make the behavior incompatible AADL components spaceship and space station can work compatibly. This process is implemented according to some mapping strategies. Parts of the rules are listed as follows.

Rule 1.
An Interface Automaton → AADL Component..

Rule 2.
The Interface Automaton Port Set → The Feature of AADL Component.

Rule 3.
The Interface Automaton State Set → The States of AADL Component.

Rule 4.
The Interface Automaton Input Action → The Guard of AADL Component Transition.

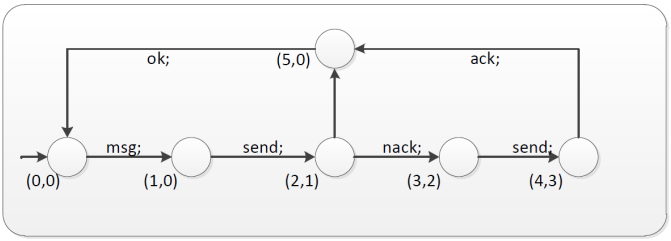

Fig. 6. Space ⊗ ship ⊗ user

5 Related Work

There have been many works about AADL behavior and IA component composition. Reference [11] proposed a formal semantics for the AADL behavior annex using Timed Abstract State Machine (TASM), and used UPPAAL [12] by mapping TASM to timed automata to verify the AADL behavior models. Bernard Berthomieu [13] mapped AADL models into the Fiacre language, which contains assignments, conditionals, while loops and sequential composition constructs. In [14], R.Passerone has developed a game-theoretical approach to find out whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements.

In summary, there are few works on the problems of AADL components composition which are important, and interface automata have been widely used to solve components composition problem. Therefore, it is a new attempt to use IA to solve the AADL components composition problems.

6 Conclusion

Although AADL is a widely used component-based model language, it cannot solve the component composition problems expediently. On the other hand, IA is a formal model for describing software components behavior and it uses the optimistic approach to solve the components composition problems effectively. We study the

transformation from AADL model to IA through ATL heterogeneous model transformation framework, present the meta-model of interface automata and propose a series of transformation rules. IA can be used to verify the behavior compatible of components and construct the compatible working environment. The environment can be mapped back to AADL components which make the behavior incompatible components can work together. In the future, we plan to analyze the semantics of AADL behavior in detail, try to verify the AADL components behavior compatibly and construct the environment which makes the behavior incompatible components can work together on the AADL directly.

Acknowledgment. This work is supported by the National Natural Science Foundation of China (No.61003025, 91118002), and by the Jiangsu Province Research Foundation (BK2010170).

References

1. OMG, Inc. Model Driven Architecture (MDA), http://www.omg.org/gov
2. Zhang, Y., Hu, J., Yu, X.F., Zhang, T., Li, X.D., Zheng, G.L.: Deriving available behavior all out from incompatible component compositions. In: Liu, Z., Barbosa, L. (eds.) Proc.of the 2nd Int'l Workshop on Formal Aspects of Component Software (FACS 2005). ENTCS, vol. 160, pp. 349–361. Elsevier, Netherlands (2006)
3. Zhang, T., Jouault, F., Attiogb, C., Li, X.D.: MDE-based model transformation: from MARTE model to FIACRE model. Journal of Software 20(2), 214–233 (2009)
4. SAE Aerospace. SAE AS5506: Architecture Analysis and Design Language (AADL), Version 1.0 (2004)
5. SAE Aerospace. SAE AS5506/1: Architecture Analysis and Design Language (AADL) Annex vol.1 (2006)
6. SAE Aerospace. SAE AS5506/2: Architecture Analysis and Design Language (AADL) Annex vol. 2 (2011)
7. de Alfaro, L., Henzinger, T.A.: Interface Automata. ACM Sigsoft Software Engineering Notes 26(5), 109–120 (2001)
8. The ATL Model Transformation Language, http://www.emn.fr/z-info/atlanmod/index.php/Model_Transformation
9. ATLAS group LINA and INRIA.: ATL: Atlas Transformation Language (2006)
10. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1/2), 31–39 (2008)
11. Yang, Z., Hu, K., Ma, D., Pi, L.: Towards a formal semantics for the AADL behavior annex. In: Proc. DATE 2009. IEEE, Los Alamitos (2009)
12. UPPAAL, http://www.uppaal.org/
13. Berthomieu, B., Bodeveix, J.-P., Chaudet, C., Dal Zilio, S., Filali, M., Vernadat, F.: Formal Verification of AADL Specifications in the Topcased Environment. In: Kordon, F., Kermarrec, Y. (eds.) Ada-Europe 2009. LNCS, vol. 5570, pp. 207–221. Springer, Heidelberg (2009)
14. Passerone, R., de Alfaro, L., Henzinger, T., Sangiovanni-Vincentelli, A.L.: Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin. In: Proceedings of the International Conference on Computer Aided Design, ICCAD 2002 (2002)