An Ada-Based Object-Oriented Modeling Language*

DAI Gui-lan, XU Bao-wen

(*Department of Computer Science and Technology, Southeast University, Nanjing* 210096, *China*);
(*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072, *China*)
E-mail: bwxu@seu.edu.cn
http://www.seu.edu.cn
Received February 1, 2000; accepted December 19, 2000

Abstract: This paper describes an Ada-based object-oriented modeling language AML which takes a unique and innovative approach to import the fundament and philosophies of Ada95, and extends it with some facilities for the support of object-oriented modeling so that AML is suitable to model large software systems. AML adopts the package concept from Ada95 and makes it become the core construct of AML. At the same time, AML introduces the facilities such as the task unit and the protect unit to describe concisely positive control components and resource protect components. Absorbing the idea of multiple viewpoint models, AML disjoints the information describing different characteristics of the given entity. AML uses the new concurrency model and the restriction facility to address modeling concurrency and nonfunctional characteristics of systems. Also AML has good extensibility and can be applied to all kinds of domains. In short, AML is user-oriented, developer-oriented, and system-oriented modeling language, and overcomes some limitations such as insufficient expressive capability and limited application domain of some other modeling languages.

Key words: modeling language; programming language; software modeling; modeling method; object-orientation; concurrent processing

Model-based approaches, one of the main software development methods, have become the hot research realms of software engineering. Modeling languages are key to the model-based method. Because of the importance of software models and modeling languages, for years people have done many successful modeling researches, and developed a set of important modeling languages, such as VDM[1], UML[2] and ROOM[3,4] etc., that are of great theoretical and practical value. Modeling languages can be divided into formal and semiformal languages according to levels of formality. Since object oriented technique can be used to deal with the software system complexity efficiently, the commonly used semiformal modeling languages are graphic object-oriented ones.

Although modeling languages and methods based on formal and graphic object-oriented technique have been progressing rapidly, there are still some shortcomings. Formal languages require profound mathematical background, which prevents the users from participating in the development and impedes the wider acceptance by

* Supported by the National Natural Science Foundation of China under Grant No. 60073012 (国家自然科学基金); the Foundation of Visiting Scholar of Key Laboratory in University of China (高等学校重点实验室访问学者基金); the Foundation of Key Teacher in University of China (高等学校骨干教授资助计划)

DAI Gui-lan was born in 1972. She got her Ph. D. degree at the Department of Computer Science and Technology, Southeast University. Her research interests are modeling language, software engineering and object technique. **XU Bao-wen** was born in 1961. He is a professor and doctoral supervisor of the Department of Computer Science and Technology, Southeast University. His current research areas include programming language, software engineering, concurrent and network software.

users. Though extended by using object-oriented technology, some formal languages, such as Z and VDM, lack the ability of expression for software structure and are not very suitable to the large software system development. Graphic object-oriented languages exceed the limitation of the formal language in readability, architecture representation and efficiency, but they easily arouse discontinuity and are impractical for capturing detail and automatic support. Furthermore most modeling languages are only suitable for some specific domains. Also, there are some questions in modeling the system of nondeterminism and concurrency, which seriously affects the expressive ability of the languages.

To research into some questions of modeling languages in expressive capability, expressive way, and application domain, we have designed an Ada-based object-oriented modeling language AML, studied its modeling method and formal technique, and discussed the implementation technique of AML model[5~7]. This paper presents the AML modeling language. It is organized as follows. Section 1 analyses the main philosophies of AML. Section 2 discusses its basic constructs and their inherent relationships. Section 3 gives an AML model to illustrate the use of AML. Section 4 compares AML with several modeling languages used commonly and draws conclusions.

1 The Design Philosophies of AML

In general, graphic languages are more intuitive and easier to depict the framework of systems. However, in describing the details of software systems, text-based programming language approach is more powerful than graphic approach. Moreover, programs written in modern programming languages, such as Ada, are normally readable and thus understandable. Particularly, Ada can be both programming-in-the-large language that specifies module interface and programming-in-the-small language that specifies module details. Ada95 not only has rich data structures and control structures, but also provides some facilities such as real time handling, concurrency handling and exception handling. Ada95 has strong functions and elegantly shows the ideas of software engineering[8]. Therefore, AML imports Ada95's fundament and philosophies, adopts its basic facilities, and extends it with the facilities for the support of modeling large software systems.

Software models are abstract descriptions for software systems to be built, which emphasize main aspects and ignore minor ones, so as to help software developers and end users to understand the features of the systems well before being built. In order to make software models intuitive and compact, AML absorbs the idea of multiple viewpoint models, disjoints the information describing different characteristics of the given entity, and places them in corresponding program unit specification, program unit body or program unit description respectively. These models complement each other. This helps the developer to concentrate on one aspect at a time, to detect early errors or inconsistency in the modeling process to improve consistency and completeness, to solve effectively inheritance anomaly, and improve model reusability and the provided facilities to keep with the refining process of the software development.

Because concurrency is considered to be an inherent feature in many application domains, it must be explicitly included in the model. Not only in order to simplify the modeling process, but also to picture the systems properly and visually, AML uses the new concurrency model, which absorbs the benefits from the explicit concurrency model and the implicit concurrency model. We consider concurrency in the early stage of the software development, and use the inherent properties of the program units to ensure that the entities are consistent and complete[5].

2 Basic Syntactic Elements

Modeling languages should describe software systems at different abstraction levels. Types or classes are basic

abstractions in object-oriented languages. AML adopts the ways of type definition, and abstracts entities in the real world as types. Since package has better encapsulation and the result model by using it corresponds well to the real world, AML adopts the package concept from Ada95 and makes it become the core construct of AML. There are some positive control components and resources protecting components in software systems. Since they have no attributes, they are difficult to be abstracted as types. Moreover, these components should be accessed exclusively, while packages cannot provide the corresponding functions. As a result AML introduces the task unit and the protected unit concepts from Ada95 and absorbs some of their features. At the same time, AML provides subprogram unit (procedure or function) which is used to model sequential activities, and generic unit which is used to avoid writing similar models simply because of different entity types.

From this point, an AML model consists of a series of interacting program units. Since AML and Ada95 are used in different development stages and abstraction levels, the program unit is modified and extended in terms of the design idea of AML, and separates its static components from its dynamic behavior ones. Thus, a program unit (i.e., package, task and protected unit) consists of a program unit specification, an optional program unit body and an optional program unit description in AML, in which the program unit specification provides an interface for users, the body part gives the concrete implementation to perform the program unit's functions, and the description part describes the program unit's dynamic behavior and constraint. In AML, the program units and parts of their facilities are similar to those in Ada95 and thus are not discussed in detail. Rather, taking package as an example, we discuss briefly special components needed in software modeling.

2.1 Package specification

Packages are program units that allow the specification of groups of logically related entities. A package consists of a package specification, a package body and a package description, in which the package specification is used to describe the part of package that is visible to other package units. In fact, package specification is a kind of protocols stipulating the available entities and operations, the accomplishing functions and the satisfying needs of the implementers of a package. Package specification itself can be divided into two parts, public part and private part. The public part of a package contains all the information that another program unit is able to know about the package. The private part, following the public part with the reserved word private, consists of a group of specifications unavailable to other program units.

According to the principle of information hiding, the definition of a type is invisible to the outside world, and thus is placed in the private part of a package. The operations corresponding to the transitions triggered by external events have two meanings, one indicates these operations that can be directly used by users and the other denotes the received messages from surrounding environment. The operations corresponding to entry actions, exit actions and self-transitions of the states are declared in the private part of a package.

To model containment relationship between the entities, and to embody adequately the refining process of the software development, a package can nest another package, task, protected unit, etc. If the nested program units are visible to users, then they are declared in the public part of the nesting package, otherwise in the private part or the body part.

In order to model the architecture of the software systems appropriately and intuitionally, to improve the tractability of the software development, AML provides association facility in a package specification to describe direct communication relationship between the nested program units. The basic syntax of a package specification is given in BNF as follows.

```
Package_Specification::=
  package Defining_Program_Unit_Name is
    {Basic_Declarative_Item}
  private
    {Basic_Declarative_Item}]
  end [[Parent_Unit_Name.]Identifier]
  Basic_Declarative_Item::=
      Type_Declaration|Object_Declaration|Subprogram_Declaration|Structure_Description_Facility
    |Package_Declaration|Task_Declaration|Protected_Unit_Declaration
```

2.1.1 Structure description facility

Architecture is one of the mechanisms dealing with complicated software systems[9]. So it must be explicitly given in the model for the convenience of the software development and maintenance. Graphic object-oriented modeling languages and modeling methods, such UML, OMT[4,10], OOA[11] etc., provide the facilities to describe the static relationship between classes, that is, association, containment relationship and inheritance relationship.

AML provides the derived type and the program units that can be nested to model intuitionally and concisely containment relationship and inheritance relationship. At the same time, AML provides the association facility to model the possible direct communication relationship (i.e., unidirectional and bi-directional) and the execution mode between the program units (i.e., sequential execution and concurrent execution). It is given in BNF as follows.

```
Structure_Description_Facility::=
    relation Association_Description
Association_Description::=
    Association_Program_Unit Association_Operator[Execution_Mode] Association_Program_Unit
Association_Program_Unit::=
    Program_Unit_Name (Multiple_Constraint)
Association_Operator::=<-|<->
Execution_Mode::=concur|sequence
Multiple_Constraint::=Integer|Integer..Integer
```

where "->" denotes the left sending one-way messages to the right, and "<->" denotes the two-side sending bi-directional messages. Multiple constraint designates the cardinality of the association numerically specified. The reserved word concur denotes concurrent execution, and sequence denotes sequential execution. If the execution mode between the program units can not be determined for the moment, then it is assigned a null.

2.2 Package body

The implementations of the operations declared in a package specification are invisible to users, and thus are encapsulated in corresponding package body. Since the operations may use some public variable or data type in the process of implementation, AML also provides object declaration and data type declaration in a package body. The bodies of the nested program units are given in that of the nesting package. The syntax of a package body is given in BNF as follows.

```
Package_Body::=
    package body Defining_Program_Unit_Name is
      {Package_Body_Declarative_Item}
    end [[Parent_Unit_Name.]Identifier];
Package_Body_Declarative_Item::=
      Type_Declaration|Object_Declaration|Subprogram_Declaration|Subprogram_Body|Package_Declaration
    |Package_Body|Task_Declaration|Task_Body|Protected_Unit_Declaration|Protected_Unit_Body
```

2.3 Package description

Package description is an extension to package in Ada95, and is used to describe the dynamic behavior and constraint of the encapsulated entities from inter-entity viewpoint and intra-entity viewpoint. In a package description, AML provides the state model to describe the process of an entity receiving messages, transforming from one state to another, and sending messages, and provides the synchronization facility to model the sequence of messages between the nested package units and their synchronization points. At the same time, AML provides the restriction facility and the exception handler for the support of the characteristics of non-functional and exception. Like package body, package description has invisibility to users, and is only involved in the implementation of a package. The grammar is partly given in BNF as follows.

```
Package_Description::=
    package description Defining_Program_Unit_Name is
        {Package_Description_Declarative_Item}
    end [[Parent_Unit_Name.]Identifier];
Package_Description_Declarative_Item::=
      Package_Description|Task_Description|Protected_Unit_Description|Exception_Handler_Facility
     |State_Machine_Facility|Synchronization_Facility|Restriction_Facility
State_Machine_Facility::=
        statemachine State_Model_Description
Synchronization_Facility::=
        synchronization Synchronization_Relationship_Description
Restriction_Facility::=
        restriction Restriction_Description
Exception_Handler_Facility::=
        exception Exception_Handler{Exception_Handler}
```

These facilities are briefly discussed as follows. Since the state model and the synchronization facility are based on certain communication mechanism, the communication mechanism should be discussed in advance.

2.3.1 Communication mechanism

In order to increase reusability and maintainability, the coupling between program units should be made as weak as possible. In AML, communication between program units is based on a message-passing model. The only thing that a sender and a receiver must share is the general semantics and format of the message.

AML provides two kinds of communication mechanisms, the synchronous communication and the asynchronous communication. Generally the former is used to synchronize or start other program units, while the latter notifies other program units about the events. To simplify modeling, the synchronous communication and the asynchronous communication use similar syntax structure and the same keywords, and add the message sign before the message name to identify them. In early stage of the software development, we may take no account of the destination or source of a message. This can reduce the coupling between program units and improve AML flexibility. Briefly it is shown in BNF as follows.

```
Message_Receiving_Statements::=
        Synchronous_Message_Receiving_Statements|Asynchronous_Message_Receiving_Statements
Synchronous_Message_Receiving_Statements::=
        accept syn Message_Name[(Parameter_Profile)][do
          Sequence_Of_Statements
        end [Message_Name]]
Asynchronous_Message_Receiving_Statements::=
```

accept asyn Message_Name[(Parameter_Profile)]

Message_Sending_Statements::=

Synchronous Message Sending Statements|Asynchronous Message Sending Statements

Synchronous Message_Sending_Statements::=

send syn Message_Name[(Actual_Parameter Part)][**to** Program_Unit_Name]

Asynchronous_Message_Sending_Statements::=

send asyn Message_Name[(Actual_Parameter_Part)][**to** Program_Unit_Name]

where **"syn"** denotes synchronous message sign and **"asyn"** denotes asynchronous message sign.

2.3.2 State model

State-Machine technology is the most direct and common way for modeling behavior[2]. A state model consists of a set of states named and direct transition between states, and is used to describe the behavior of an entity in response to external stimuli. To eliminate state explosion that attributes to flat or unstructured state model, AML provides explicit representation for hierarchical state, 'and' state and 'or' state. The hierarchical state modeling capabilities of AML can be used to express this abstraction by nesting substates inside a higher-level state. If the system is in some substate, then it is also in the corresponding superstate. 'And' state means that all the members have values at the same time. 'Or' state means that one of the members has value at a time. Each state component may have its own sub-components. The basic syntax of state type definition is given in BNF as follows.

State_Type_Definition::=State_Component_List

State_Component_List::='Or'_State_Component_List|'And'_State_Component_List

'Or' State Component List::=

(State_Value[State_Component List]{; State_Value [State_Component_List]})

'And' State_Component_List::=

(State_Value[State_Component List]{, State_Value [State_Component_List]})

AML modifies the select statement referenced to Ada95 to model all kinds of possible structures in state model. A series of actions is performed when an entity transforms from a state to another. Though these actions (such as sending messages and modifying the value of state) are given in the corresponding subprogram body, in order to improve state model readability and expressive capability, the message sending statements and the object state values are explicitly shown by comments. Moreover, because of the transition triggered by any reason, the exit action and entry action of the same state are invariable. An entry action, a transition and an exit action of each state are thus modeled respectively as an operation. This helps to improve model reusability and simplify design.

2.3.3 Synchronization facility

In the process of modeling software systems, it is quite important for the description of sequence of handling messages, which not only reflects the process of direct interaction between the entities and their synchronization points, but also is the foundation for refining the behavior of an entity further. Most graphic modeling languages provide corresponding facilities for the support of synchronization mechanism, such as sequence diagram in UML, scenario in ROOM. There are some common questions: lack of expressive capability of the cases of nondeterminate interaction and an entity sending or receiving many messages simultaneously. Based on the description idea of sequence diagram and scenario, AML provides synchronization facility, which can not only describe the process of determinate or nondeterminate interaction between the program units, but also model the cases of a program unit sending or receiving many messages simultaneously. The syntax of synchronization facility is briefly described as follows.

Synchronization_Relationship Description::=

when Cond=>

Synchronization_Description_Statement {;Synchronization_Description_Statement};

Synchronization_Description Statement::=

([Message_Label.] Program Unit_Name, The_Received_Message_List, The_Sent_Message_List)

The_Received_Message_List::=

Message_Name [The_Relationship_Between_Messages] Message_Name

{[The Relationship_Between_Messages] Message_Name}

The_Sent_Message_List::=

Message_Name [The_Relationship_Between_Messages] Message_Name

{[The_Relationship_Between_Messages] Message_Name}

The_Relationship_Between_Message::=>|||

Message_Label::=Integer

where "message_label" identifies the sequence of interaction between the program units. If the arrival of a message is at random, then the "message label" may not be needed. ">" denotes the order relationship between the messages and "|" denotes that the two are optional. Further more, since it is possible that the activities of an entity are a circular process, we can see the statements as a loop body.

2.3.4 Restriction facility

In order to facilitate the model proof, and to ensure system correctness, the restriction section of an AML program unit contains an extended RTL (Real Time Logic) predicate which constrains the behavior of an entity to describe some properties such as sequence of messages handling, safety and liveliness. The restriction facility complements the behavior that state model and synchronization facility can not describe. On the other hand, it describes the non-functional characteristic of an entity. In AML, integrating restriction facilities with inherent characteristics of the program units may efficiently deal with nondeterminism of the systems. For example: A program unit may receive a set of messages: M_1, M_2, M_3, M_4 and M_5. Suppose the operation $M_1()$ is triggered by M_1, the operation $M_2()$ is triggered by M_2, and so on. If $M_3()$ is delayed until $M_1()$ has commenced execution, $M_5()$ is delayed until $M_3()$ has commenced execution, but $M_2()$ and $M_4()$ are at random, then it is described as follows.

$$\forall i \in N: !(M_1(),i) \textbf{ before } \hat{}(M3(),i) \textbf{ and } !(M_3(),i) \textbf{ before } \hat{}(M_5(),i)$$

where "!" denotes the time of stopping execution, "^" denotes the time of beginning execution, and "i" denotes the i-th time. Restriction clauses also include the connectives "**and**", "**or**", negation "**not**" and so on.

3 An Example

In order to illustrate the use of AML, we give an example of AML model, which schedules and controls four elevators in a building with 40 floors. The elevators will be used to carry people from one floor to another in the conventional way. The system can be divided into three subsystems. These are a problem domain subsystem (PDM), a hardware interface subsystem (HIM), and a task management subsystem (TMM). We only model the PDM partly owing to the limitation of space.

The PDM subsystem receives messages from the HIM subsystem such as Elevator_Summoned, performs a series of activities, and then sends appropriate messages to the HIM subsystem and the TMM subsystem. The PDM encapsulates several entities: (1) an Arrival_Event provicing all the services that an elevator arriving at a floor must perform; (2) a Destination_Event encapsulating how to recognize destination request; (3) an Elevator providing the data and services needed for the elevator control and management; (4) an Elevator_Motor containing all kinds of the elevator control services; (5) an Overweight_Sensor encapsulating the know_how of the sensor; (6) a Destination_Panel being a part of interface and pointing out which elevator should get to the destination; (7) a Floor implementing the functions of elevator scheduling; (8) a Summons_Event encapsulating how to recognize summon request; and (9) a Summon_Panel being similar to the Destination_Panel[11]. These entities and their

relationship are partly given as follows.

```
package PDM is -- The subsystem PDM is encapsulated in the package PDM.
  procedure ElevatorCommand (...);
  task Elevator is -- Since the elevator is a positive control, and thus is modeled as a task unit.
   entry Control_Elevator (Control_Elevator_Command: String; Current_Direction, Current_State: out String);
   entry Recognize_Elevator_Ready (Elevator_Id: Integer; Current_State: String);
   entry Report_Elevator_Status (…);
  end Elevator;
  package ArrivalEvent is
    ...
  end ArrivalEvent;
private
 relation
   Summons_Event(1)->concur Summons_Panel(0..m);
   Floor(1..m)<->concur Elevator(1..n);
     ...
end PDM;

package body PDM is
 task body Elevator is
   package Destination_Panel is
     ...
   end Destination_Panel;…
   procedure Control_Elevator (Control_Elevator_Command: String; Current_Direction, Current_State: out String) is
   begin
   if Control_Elevator_Command=(Up|Down) then
     Current_Direction:=(Up|Down);
     Current_Status:=Busy;
   else
     Current_Status:=Stopped;
     send asyn Control_Elevator_Motor to Elevator_Motor(Elevator_Id);
   end;
     ...
 end Elevator;
end PDM;

package description PDM is
 task description Elevator is
   package description Destination_Panel is
   ...
   end Destination_Panel;
   type Elevator_State is (Busy(Up_Up,Up_No,Dn_Dn,Dn_No), Stop, Idle, Ready);
     ...
   E_S: Elevator_State;
 statemachine
   loop
     when true=>
       accept syn Report_Elevator_Status (…) do
         Report_Elevator_Status(…);
       end;
     when (E_S=Idle and Summon_Floor=Current_Floor)
       accept asyn Elevator_Summoned(…);
         E_S:=Stopped;
           ...
       raise State_Error; -raises exception.
```

```
    when E_S=Stopped=>do --The following is a self transition.
      Recognize_Elevator_Ready(…);
    --send syn Report_Sensor_Status (Elevator_Id) to Overweight_Sensor(Elevator_Id); E_S:=Ready;
    end loop;
    restriction...
    exception
      when State_Error=>do...
  end Elevator;...
  synchronization--Description for synchronization points and the transferred messages between the entities in PDM.
    (Elevator, Control_Elevator, Control_Elevator_Motor);
    (Elevator, Recongnize_Elevator_Ready, Report_Sensor_Status; Report_Elevator_Ready);
    (Floor, Report_Elevator_Ready,...);...
  end PDM.
```

4 Discussion and Conclusion

AML inherits many mechanisms of Ada95. So AML has the characteristics of Ada95. It absorbs the idea of multiple viewpoint models used in some graphical object-oriented modeling languages and modeling methods. Therefore it can early detect errors or inconsistency and make the models more intuitive and clear. This overcomes, to some extent, the shortcoming of modeling large and complex systems in textual languages.

Though OCTOPUS and OMT use the three models to describe the given system, its underlying language lacks rigorous syntax and semantics. UML is a modeling language based on several OOA/OOD methods. Though its semantics is not given in formal way completely, it is still quite robust for it is a modeling language that has defined the complete semantics of its elements. No doubt it has very strong ability of expression. But UML adopts the extended concepts and graphic notations of those OO methods that contribute to the UML. So its complexity is well above those OO methods. It has more than one hundred modeling elements, and needs nine kinds of diagrams to describe a system. Moreover, different kinds of diagrams have redundancy and UML doesn't indicate which diagrams can be vomited. As a matter of fact, we must pay much more to learn and to use in the projects. AML uses the program units to describe the given system, so this language is simple and can reduce the redundancy to large extent. Further more, AML adopts the basic idea of OO graphics structure representations, and provides the structure description facility to describe the architecture of the given system so that it has got rid of the weakness of structure expression of textual language.

VDM++ is a formal modeling language. Though it has rigorous syntax and semantics, it lacks the ability of expressing large software architecture. AML uses the facility similar to the aux_reasoning part of VDM++ to describe the nonfunctional characteristics of systems such as nondeterminism, liveliness, real-time reactiveness etc. This increases the system reliability dramatically.

AML has both formal notations and semiformal notations. By using Z notation to formalize basic concept of AML, we can translate people-oriented informal representation with certain concepts into computer-oriented formal one. Therefore, AML has the merits of formal languages.

Further more, AML not only has more powerful ability of expression for concurrent model, but also has the advantages of easily transforming programming language into implementation and reducing discontinuity. This helps to improve the software automation. Also AML is a general-purpose modeling language which has good extensibility and can be applied to all kinds of domains.

In short, AML is user-oriented, developer-oriented, and system-oriented and overcomes the limitations of existing modeling languages efficiently. At the same time, it also propels forward the study of modeling languages.

Acknowledgements The authors wish to thank CHEN Zhang-qiang, LI Bang-qing, ZHOU Yu-ming, LIU Yuan, LI Sheng-zhi and TENG Cheng for their valuable comments.

References:

[1] Lano, K. Formal Object-Oriented Development. London: Springer-Verlag, 1995.

[2] Sinan, S. A. UML in a Nutshell. New York: O'Reilly & Associates, 1998.

[3] Selic, B. Real-Time Object-Oriented Modeling. US: Katherine Schowdter, 1994.

[4] Dai, Gui-lan, Xu, Bao-wen. A comparison and analysis of real-time object-oriented modeling methods ROOM and OCTOPUS. ACM SIGPLAN Notices, 1999,34(12):67~70.

[5] Dai, Gui-lan. A study on Ada-based modeling language AML and its modeling method [Ph. D. Thesis]. Nanjing: Southeast University, 2000 (in Chinese).

[6] Dai, Gui-lan, Xu, Bao-wen. Design and implementation of modeling language AML. Journal of Wuhan University (Natural Science Edition), 1999,45(5B):654~657 (in Chinese).

[7] Dai, Gui-lan, Xu, Bao-wen. Algebraic semantics of modeling language AML. Journal of Wuhan University (Natural Science Edition), 1999,45(5B):658~660 (in Chinese).

[8] ISO/IEC. Ada Reference Manual—— Language and Standard Libraries. ISO/IEC 9652: 1995(E).

[9] Deniel, L. M. Describing software architecture styles using graph grammars. IEEE Transactions on Software Engineering, 1998.24(7):521~533.

[10] Maher, A., Juha, K., Jurgen, Z. Object-Oriented Technology for Real-Time Systems a Practical Approach Using OMT and Fusion. New York: Prentice-Hall Press, 1996.

[11] Yourdon, E., Argila, C. Case Studies in Object-Oriented Analysis & Design. New York: Prentice-Hall Press, 1996.

附中文参考文献:

[5] 戴桂兰.基于 Ada 的建模语言 AML 及其建模方法研究[博士学位论文].南京:东南大学,2000.

[6] 戴桂兰,徐宝文.面向对象建模语言 AML 的设计与实现.武汉大学学报(自然科学版),1999,45(5B):654~657.

[7] 戴桂兰,徐宝文.面向对象建模语言 AML 的代数语义.武汉大学学报(自然科学版),1999,45(5B):658~660.

一个基于 Ada 的面向对象建模语言

戴桂兰, 徐宝文

(东南大学 计算机科学与工程系,江苏 南京 210096);
(武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

摘要: 给出了一个基于 Ada 的建模语言 AML. AML 以 Ada95为基础,吸取了 Ada95的基本原理和思想,利用支持建模的设施对其进行扩充,以便适用于软件建模的各个阶段. AML 沿用了 Ada95中的程序包概念,并将程序包作为它的核心成份,同时,沿用了 Ada95的任务单元和保护单元等设施,以精确描述了软件系统中主动控制成份和资源保护成份的各种特性;AML 吸取了图形化面向对象建模语言与建模方法所采用的多视点模型思想,将实体不同侧面的特征分开描述;AML 利用一种新的并发模型和限制设施,有效地解决了系统的并发特性和不确定性等非功能特性的描述问题;AML 有较强的可扩充性,能应用于各种应用领域. 总之,AML 是一个既面向用户,面向开发者,又面向系统的通用建模语言,有效地克服了现有一些建模语言在表达能力和应用范围等方面存在的诸多不足.

关键词: 建模语言;编程语言;软件建模;建模方法;面向对象;并行处理

中图法分类号: TP312 **文献标识码**: A