J. Software Engineering & Applications, 2010, 3: 245-254
doi:10.4236/jsea.2010.33030 Published Online March 2010 (http://www.SciRP.org/journal/jsea)

Lightweight Behavior-Based Language for Requirements Modeling

Zhengping Liang[1,2], Guoqing Wu[3], Li Wan[3]

[1]College of Computer and Software Engineering, Shenzhen University, Shenzhen, China; [2]State Key Laboratory of Software Engineering, Wuhan, China; [3]School of Computer, Wuhan University, Wuhan, China.
Email: liangzp@szu.edu.cn

Received December 22nd, 2009; revised January 15th, 2010; accepted January 19th, 2010.

ABSTRACT

Whether or not a software system satisfies the anticipated user requirements is ultimately determined by the behaviors of the software. So it is necessary and valuable to research requirements modeling language and technique from the perspective of behavior. This paper presents a lightweight behavior based requirements modeling language BDL with formal syntax and semantics, and a general-purpose requirements description model BRM synthesizing the concepts of viewpoint and scenario. BRM is good for modeling large and complex system due to its structure is very clear. In addition, the modeling process is demonstrated through the case study On-Line Campus Management System. By lightweight formal style, BDL & BRM can effectively bridge the gap between practicability and rigorousness of formal requirements modeling language and technique.

Keywords: *Behavior Description Language* (*BDL*), *Scenario*, *Viewpoint*, *Behavior Requirements Model* (*BRM*)

1. Introduction

Software requirements modeling is an important phase of software development process. To obtain high quality requirements model, an effective and well-defined requirements modeling language and technique, which both has formal semantic and can be easily understood and used by all kinds of stakeholders, is needed.

The existing requirements modeling languages and techniques can be roughly divided into two categories. One is the semi-formal style based on graph symbol, the most famous representative of which is UML [1]. The other is the formal style based on mathematics symbol, such as Automata [2], Z [3], E-LOTOS [4], Petri net [5], Pi-calculus [6], etc. The former has the advantage of strong intuition, of being easy to be understood and used, but it usually lacks rigorous semantics and easily leads to an inconsistent and incomplete requirements model. On the contrary, the latter has rigorous semantics basis and is convenient to deduce and verify some properties, but it has poor practicability, and requires the user and analyzer with advanced skills.

How to deal with the gap between practicability and rigorousness of formal requirements modeling language and technique is a big challenge [7]. Some researches suggest to designating formal semantic for semi-formal language [8], and others believe the combination of graph symbol and formal language are more positiveness [9]. Although all of those approaches have some effect to bridge the gap, there are still inconvenient to put them into practice. At the same time, whether or not a software system satisfies the anticipated user requirements is ultimately determined by the behaviors of the software. That is to say, the requirements modeling language and technique need to support the description and validation of behavior. So it is necessary and valuable to research software requirements modeling language and technique both has practicability and rigorousness from the perspective of software behavior.

Due to lightweight formal style can help to bridge the gap between practicability and rigorousness [10], we established a lightweight formal language BDL (Behavior Description Language) to modeling user's requirements, which is based on the identifiable behaviors of software system. What should be emphasized is that the behaviors not only include the observable behaviors from the system external interface but also consist of the behaviors resided in the internal of the system. In addition, in order to support the requirements modeling of large and complex software, a general-purpose requirements description model BRM (Behavior Requirements Model) is proposed, which partly synthesizes some ideas of viewpoint-oriented requirements engineering [11] and scenario-oriented requirements engineering [12].

The structure of this paper is organized as follows: Section 2 introduces the formal syntax of BDL and its structural operational semantics. Section 3 introduces the requirements description model BRM and Section 4 demonstrates the modeling process through the case study On-line Campus Management System. Finally, the related works are discussed in Section 5 and the conclusions and future works are discussed in Section 6.

2. Behavior Based Requirements Modeling Language

A behavior is a certain interaction among two or more entities. For easy discussion, this paper presumes a behavior is an interaction only between two entities. We define a software behavior as a process during which a subject implements an operation, service, or action to an object. The subject and the object which may be physical or logistic, can be a person, a software or hardware component of system, or certain element of environment.

The structure of each behavior consists of a subject, an object, some properties, some inputs, some outputs, and an operation, service, or action. If a behavior can't be divided into two or more sub-behaviors, it is an atomic behavior. An atomic behavior is a simple behavior. Two or more simple behaviors form a composite behavior. In addition, according with the interact mode of software behaviors, the combine pattern of simple behaviors can be divided into five categories: sequence, certainty choice, uncertainty choice, parallel and shielding.

Based on the above consideration about software behavior, the followings are the syntax and structural operational semantics of behavior based requirements modeling language BDL.

2.1 Syntax of BDL

Suppose $ABehID, ABehID_i (i \in N)$ are atomic behavior identifier, $BehID, BehID_i (i \in N)$ are behavior identifier.

2.1.1 Atomic Behavior Expression

$$\begin{aligned} ABehID : f(&sub, obj\,[\&\ obj's\ additional\ remarks]) \\ &[\text{When } prepositive\ conditions] \\ &[\text{INFrom}(ID)(u_1,...,u_n)]^* \\ &[\text{OUTTo}(ID)(v_1,...,v_m)]^* \end{aligned}$$

where,

f is an operation or an action;

sub and obj are the behavior's subject and object respectively;

When clause denotes the *prepositive conditions* according to which the behavior can execute;

INFrom and OUTTo clause denote the behavior's input data and output data respectively;

ID denotes a certain atomic behavior identifier, a external entity or a viewpoint identifier related to INFrom or OUTTo ;

$u_i (i \in \{1...n\})$ and $v_i (i \in \{1...m\})$ are described with the format of $dataname$ or $dataname = value$;

The superscript $*$ denotes there are 0 or multiple items that belong to the same category. Besides, there are two kinds of special atomic behavior:

1) Null action: $ABehID : Idle$;

2) End action of composite behavior:

a) $ABehID : Return(ABehID_i)$

//jump to execute atomic behavior $ABehID_i$;

b) $ABehID : Return()$

//end of execute composite bahavior .

2.1.2 Simple Behavior

$\vdash ABehID$ //atomic behavior act as simple behavior

2.1.3 Composite Behavior

1) Sequence behavior:

a) $\dfrac{\vdash ABehID \,\&\vdash BehID}{\vdash ABehID; BehID}$

b) $\dfrac{\vdash BehID \,\&\vdash ABehID}{\vdash BehID; ABehID}$

c) $\dfrac{\vdash BehID_1 \,\&\vdash BehID_2 \,\&...\&\vdash BehID_n}{\vdash BehID_1; BehID_2;...; BehID_n}$

2) Certainty choice behavior:

$$\frac{\vdash BehID_1 \,\&\vdash BehID_2 \,\&\, b \text{ is a boolean expression}}{\vdash If\ b\ Then\ BehID_1\ Else\ BehID_2\ Fi}$$

3) Uncertainty choice behavior:

$$\frac{\vdash BehID_1 \,\&\vdash BehID_2 \,\&...\&\vdash BehID_n}{\vdash BehID_1 + BehID_2 + ... + BehID_n}$$

4) Parallel behavior:

$$\frac{\vdash BehID_1 \,\&\vdash BehID_2 \,\&...\&\vdash BehID_n}{\vdash BehID_1 \,||\, BehID_2 \,||...||\, BehID_n}$$

5) Shielding behavior:

a) $\dfrac{\vdash BehID \,\&\vdash ABehID}{\vdash BehID / ABehID}$ //shielding atomic behavior

b) $\dfrac{\vdash BehID \,\&\vdash BehID_1}{\vdash BehID / BehID_1}$ //shielding composite behavior

2.2 Structural Operational Semantics of BDL

Definition1: Suppose B is a behavior expression, σ is a state of system, then $< B, \sigma >$ is a configuration. $< B, \sigma >$ denotes the current state is σ and the behavior expression to be executed is B. $<\sigma>$ is also a configuration, which denotes the current state is σ and there are no behavior expression need to be executed.

Definition2: Suppose b is a Boolean expression, σ is a state, $eval < b,\sigma >$ denotes the Boolean value of b at σ.

Suppose $\alpha, \alpha_i (i \in N)$ are atomic behavior, B, B_i $(i \in N)$ are behavior expression. The structural operational semantics of BDL can be defined in this way:

1) Semantic of atomic behavior expression:

$$< \alpha, \sigma > \rightarrow < \sigma' >$$

2) Semantic of Null action:

$$< Idle, \sigma > \rightarrow < \sigma >$$

3) Semantic of End action of composite behavior: Suppose α is the first atomic behavior of B.

$$< Return(\alpha), \sigma > \rightarrow < B, \sigma >$$

$$< Return(), \sigma > \rightarrow < \sigma >$$

4) Semantic of sequence behavior:

$$\frac{< B_1, \sigma > \rightarrow < \sigma' >}{< B_1; B_2, \sigma > \rightarrow < B_2, \sigma' >}$$

$$\frac{< B_1, \sigma > \rightarrow < B_1', \sigma' >}{< B_1; B_2, \sigma > \rightarrow < B_1'; B_2, \sigma' >}$$

5) Semantic of certainty choice behavior:

$$\frac{eval < b, \sigma >= TRUE}{< If\ b\ Then\ B_1\ Else\ B_2\ Fi, \sigma > \rightarrow < B_1, \sigma >}$$

$$\frac{eval < b, \sigma >= FALSE}{< If\ b\ Then\ B_1\ Else\ B_2\ Fi, \sigma > \rightarrow < B_2, \sigma >}$$

6) Semantic of uncertainty choice behavior:

$$\frac{< B_1, \sigma > \rightarrow < B_1', \sigma' >}{< B_1 + B_2, \sigma > \rightarrow < B_1', \sigma' >}$$

$$\frac{< B_2, \sigma > \rightarrow < B_2', \sigma' >}{< B_1 + B_2, \sigma > \rightarrow < B_2', \sigma' >}$$

7) Semantic of parallel behavior:

$$\frac{< B_1, \sigma > \rightarrow < B_1', \sigma' >}{< B_1 \| B_2, \sigma > \rightarrow < B_1' \| B_2, \sigma' >}$$

$$\frac{< B_2, \sigma > \rightarrow < B_2', \sigma' >}{< B_1 \| B_2, \sigma > \rightarrow < B_1 \| B_2', \sigma' >}$$

8) Semantic of shielding behavior:
Suppose $B = \alpha'; B'$.

$$\frac{< B, \sigma > \rightarrow < B', \sigma' >}{< B / \alpha, \sigma > \rightarrow < B' / \alpha, \sigma' >} \quad (\alpha' \neq \alpha)$$

//shielding atomic behavior

Suppose $B = B_i; B'$.

$$\frac{< B, \sigma > \rightarrow < B', \sigma' >}{< B / B_1, \sigma > \rightarrow < B' / B_1, \sigma' >} \quad (B_i \neq B_1)$$

//shielding composite bahavior

3. Behavior Based Requirements Description Model

As to small and simple software system, BDL can be used to describe its requirements model directly due to BDL's syntax is also simple and small. But it is hard to describe requirements model of large and complex software system using BDL directly because on the one side the software scale and structure may be very complicated, and on the other side many kinds of stakeholders who reside in different time zone and space, may be involved.

To deal with large and complex problems, people often employ the strategy of divide-and-rule. Based on this method, we propose a general-purpose requirements description model BRM, which synthesizes the concepts of viewpoint and scenario. The model process of BRM consists of five steps: first, to identify the scope of the whole problem domain of the software system, next, to divide the problem domain into some interrelated sub-domains. After that, to list all potential viewpoints and their sequence or overlap relationships of each sub-domain based on the viewpoint identifying methods of viewpoint-oriented requirements engineering [11]. Later on, to look for different scenarios and their sequence or overlap relationships of each viewpoint. Finally, to adopt the scenario describing way of scenario-oriented requirements engineering [12] to establish each scenario model using BDL.

BRM is composed of three kinds of model. One is the scenario behavior model, another is viewpoint behavior model, and the last is system behavior model. The followings are the formal definition of them.

Definition3 (Scenario behavior model): A scenario's behavior model is a 6-tuple:

$$M_s = (B, ;, If, +, \|, /)$$

where,

B is the set of behaviors within the scenario, and each behavior in B has a corresponding behavior expression;

;, If, +, ||, / respectively denotes the relationship of sequence, certainty choice, uncertainty choice, parallel and shielding between behaviors.

The syntax structure of scenario behavior model is defined as **Figure 1**, where, $ABehID : Atomic\ behavior$ is a certain atomic behavior expression; $BehaviorOperator$ is one of the relation symbol between behaviors, that is ;, If, +, ||, / .

Definition4 (Viewpoint behavior model): A viewpoint's behavior model is a 4-tuple:

$$M_v = (S, \circ, \Diamond, \perp)$$

where,

S is the set of scenarios within the viewpoint, and each scenario in S has a corresponding scenario behavior model;

$\circ$ is a 2-tuple operator, which denotes two scenarios have the sequence relationship in terms of execution;

$\Diamond$ is also a 2-tuple operator, which denotes two scenarios have overlaps in content, that is, they have common behaviors;

$\perp$ denotes two or more scenarios are independent of each other in execution order and in content.

These relation operators can be use to assisting analyze and check requirements model's properties from the aspect of syntax and semantic at the phase of requirements analysis.

The syntax structure of viewpoint behavior model is defined as **Figure 2**, where, *ScenarioOperator* is the scenario's relation symbol $\circ$ or $\Diamond$.

Definition5 (System behavior model): A software system's behavior model is a 4-tuple:

$M=(V, \circ, \Diamond, \perp)$

where,

V is the set of viewpoints related to the system, and each viewpoint in V has a corresponding viewpoint behavior model;

$\circ$ is a 2-tuple operator, which denotes two viewpoints have the sequence relationship in terms of execution;

$\Diamond$ is also a 2-tuple operator, which denotes two viewpoints have overlaps in domain, that is, the sub-domains where they belong to have common elements;

$\perp$ denotes two or more viewpoints are independent of each other in execution order and in domain.

These relation operators can also be use to assisting analyze and check requirements model's properties from the aspect of syntax and semantic at the phase of requirements analysis.

The syntax structure of system behavior model is defined as **Figure 3**, where, *ViewpointOperator* is the viewpoint's relation symbol $\circ$ or $\Diamond$.

Obviously, because the structure and relationship of above models are very clear, people can smoothly transfer the user requirements expressed by natural languages to formal requirements model expressed by BDL based on BRM. Hence, BDL & BRM make a moderate balance between practicability and rigorousness.

```
ScenarioID
SCBEGIN
   [ABEH:    //list of atomic behaviors, it also can be given in BEH directly
      ABehID : atomic behavior
      [, ABehID : atomic behavior]*;;]
   BEH :     //list of behaviors
      BehID = AbehID | atomic behavior | BehID |
         (AbehID | atomic behavior | BehID) BehaviorOperator (AbehID | atomic behavior | BehID)
         [BehaviorOperator (AbehID | atomic behavior | BehID)]*  //at lease one behavior in a scenario
      [, BehID = AbehID | atomic behavior | BehID |
         (AbehID | atomic behavior | BehID) BehaviorOperator (AbehID | atomic behavior | BehID)
         [BehaviorOperator (AbehID | atomic behavior | BehID)]* ]*;;
      SBehID =    //scenario behavior expression
         (BehID | BehID BehaviorOperator BehID [BehaviorOperator BehID]*);;
SCEND
```

Figure 1. Syntax of scenario behavior model

```
ViewpointID
VPBEGIN
   [data storage pool ID];; //used to store data input from other viewpoint and data
                            //shared by different scenarios within the viewpoint
   ScenarioID               //at lease one scenario in a viewpoint
   [, ScenarioID]*;;
   SC_Relationship =        //set of relationship between scenarios
      {[< ScenarioID ScenarioOperator ScenarioID >
       [, < ScenarioID ScenarioOperator ScenarioID >]* ]};;
VPEND
```

Figure 2. Syntax of viewpoint behavior model

4. Case Study

On-line Campus Management System (*OCMS*) consists of several subsystems related each other, which used for the daily management of education administrative unit and schools. Its user requirements have modeled using BDL & BRM. In this section, we demonstrate a partial of function requirements model of *OCMS*.

The following is part of the functions of Student Information Management Subsystem:

1) *Student needs to scan his or her IC card at the door-control reader when he or she enters or leaves schoolyard, at the same time, a correlative short message will be automatically sent to the student's parents' mobile phone*;

2) *Teachers can process students' all kinds of information and send student's information to his or her parents by the way of short message and E-mail*;

3) *Administrator distributes IC cards and manages its authorization. Besides, Administrator sets the students attendance rules and the system automatically creates the students attendance reports*;

4) *Parents can query his or her child's all kind of information at school by the way of short message, automatic voice and webpage.*

The logic structure of above functions as **Figure 4**.

Although the above function requirements look very simple, there are many complicated and redundant details. For example, how long and how to does the attendance report is created, how to manage the input, modification, processing, storage, transmission, respondence, etc. of all kinds of students' information among different domain elements. Due to space limitations, we directly give the analysis result of above requirements and only demonstrate a partial of requirements model using BDL & BRM.

The problem domain boundary of above user requirements is clear. The followings are the five sub-domains of it:

Sub-domain 1: student, IC card, IC reader, mainframe, door and swivel of door;

Sub-domain 2: administrator, attendance rules, terminal, mainframe, IC card, IC reader;

Sub-domain 3: teacher, all kinds of student's information at school, terminal, mainframe;

Sub-domain 4: parents, mobile phone, telephone, terminal, mobile phone networks, telephone networks, Internet, all kinds of student's information at school;

Sub-domain 5: mainframe, IC reader, terminal, mobile phone networks, telephone networks, Internet, attendance report, all kinds of student's information at school, list of IC card information.

```
SystemID
SYBEGIN
    ViewpointID        //at lease one viewpoint in a system
    [,ViewpointID]*;;
    VP_Relationship =     //set of relationship between viewpoints
       {[<ViewpointID ViewpointOperator ViewpointID>
        [,<ViewpointID ViewpointOperator ViewpointID>]*]};;
SYEND
```

Figure 3. Syntax of system behavior model

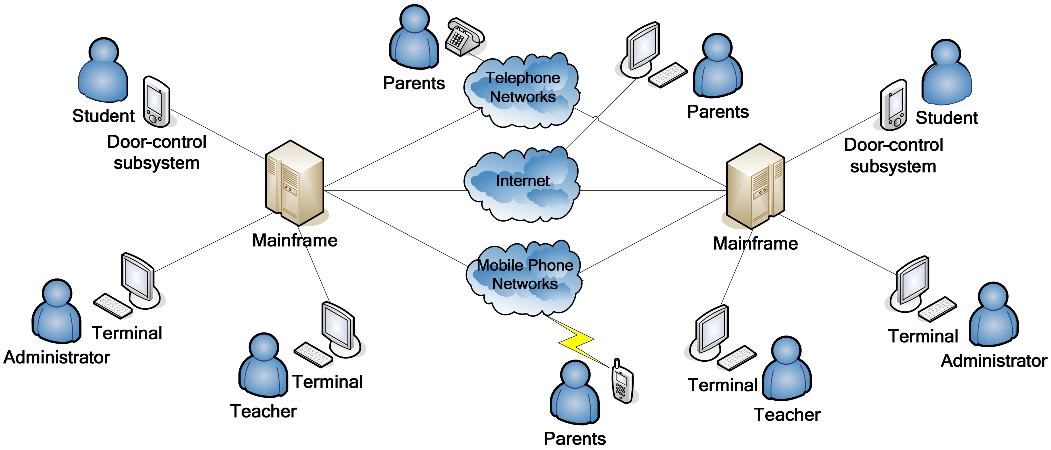

Figure 4. Logic structure of student information management subsystem

Table 1. Relationships of viewpoints belong to Sub-domain 5

Relationships	VP_ICInfo_Manage	VP_AttenRep_Create	VP_Query_Respond	VP_Info_Send	VP_Info_Edit
VP_ICInfo_Manage	\	◊	⊥	⊥	⊥
VP_AttenRep_Create	\	\	∘	⊥	⊥
VP_Query_Respond	\	\	\	◊, ∘	◊, ∘
VP_Info_Send	\	\	\	\	◊, ∘
VP_Info_Edit	\	\	\	\	\

Notes: "\" denotes null.

```
(1) ICreaderDisp1: display(ICreader, screen)
                OUTTo(screen)(Prompt="Please scanning card!")
(2) DoorConWait:idle()          //waiting user to scan card
(3) ScanC:scancard(person, IC)
(4) ReadC:read(ICreader, IC)
          OUTTo(SendCInfo)(ICNo, username),
(5) SendCInfo:send(ICreader, Mainframe)     //send the username to viwepoint VP_ICInfo_Manage
             OUTTo(VP_ICInfo_Manage)(ICNo, username)
(6) RecVerInfo:receive(ICreader, Mainframe)     //receive the verification result of IC
               INFrom(datapool)(result)     //the result is store in the viewpoint's data pool
(7) ICReaderDisp2:display(ICreader, screen)
                OUTTo(screen)(username, Prompt="Coming Please!")
(8) AllowOpen:allow(ICreader, swivel)
              OUTTo(swivel)(signal)
(9) OpenDoor:open(swivel, door)     //the action of open the door
(10) CloseDoor:close(swivel, door)
```

Figure 5. Atomic behavior expressions of the scenario with the right to open the door

These sub-domains related each other through common elements. For example, Sub-domain 1 and Sub-domain 5 has the common element IC reader, which hints some viewpoints of them may have the relationship "◊" defined in Definition 5.

As to Sub-domain 5, we can identify five viewpoints: VP_ICInfo_Manage, VP_AttenRep_Create, VP_Query _Respond, VP_Info_Send, VP_Info_Edit. The relationships of them as **Table 1** using the shape of strictly upper triangular matrix.

As to Sub-domain 1, there is only one viewpoint VP_ScanCard, which have the following relationships with the viewpoints belong to Sub-domain 5:

<VP_ScanCard ◊ VP_ICInfo_Manage>, <VP_Scan Card ∘ VP_Info_Send>, etc.

Now, we give a demonstration of VP_ScanCard's modeling process and its behavior model. The followings are the detailed user requirements of this viewpoint:

When a student wants to enter or leave school, she or he needs to scan her or his IC card at the IC reader firstly. If the IC-holder is authorized to enter or leave the school, the door-control system will display the IC-holder's name on the IC reader's screen and open the door. Otherwise, a warning sound will be played in the IC reader's speaker, and the reason why the person is not permitted to enter or leave will be displayed on the screen.

In this viewpoint, there are two scenarios: one is the IC-holder has the right to enter or leave school SC_ValidScanCard, the other is the opposite SC_InvalidScanCard.

First, we list all atomic behavior expressions belong to SC_ValidScanCard according to above requirements as **Figure 5**.

Then, the scenario behavior model of SC_ValidScanCard can be established as **Figure 6** according to the interrelated relationship of above atomic behavior expressions and Definition 3.

Next, the scenario behavior model of SC_Invalid ScanCard as **Figure 7** can be established similarly.

After that, due to SC_ValidScanCard and SC_ InvalidScanCard have the common elements in domain, the viewpoint behavior model of VP_ScanCard is established as **Figure 8**.

Here, the behavior model of VP_ScanCard is established successfully. Behavior model of other user requirements can be established similarly.

```
SC _ ValidScanCard
SCBEGIN
  ABEH:
    ICreaderDisp1: display(ICreader, screen)
                 OUTTo(screen)(Prompt="Please scanning card!"),
    DoorConWait:idle(),
    ScanC:scancard(person, IC),
    ReadC:read(ICreader, IC)
           OUTTo(SendCInfo)(ICNo, username),
    SendCInfo:send(ICreader, Mainframe)
              OUTTo(VP_ICInfo_Manage)(ICNo, username),
    RecVerInfo:receive(ICreader, Mainframe)
               INFrom(datapool)(result),
    ICReaderDisp2:display(ICreader, screen)
                  OUTTo(screen)(username, Prompt="Coming Please!"),
    AllowOpen:allow(ICreader, swivel)
               OUTTo(swivel)(signal),
    OpenDoor:open(swivel, door),
    CloseDoor:close(swivel, door);;
  BEH:
    BehValidUResp=ICReaderDisp2□AllowOpen,
    BehValidU=
      ICreaderDisp1;
      DoorConWait;
      ScanC;
      ReadC;
      SendCInfo;
      RecVerInfo;
      BehValidUResp;   //if the IC is valid, open the door
      OpenDoor;
      CloseDoor;
      Return(ICreaderDisp1);;
    SBehID=BehValidU;;
SCEND
```

Figure 6. Scenario behavior model with the right to open the door

5. Related Works

The semi-formal and formal requirements modeling language and technique both have achieved prominent outcomes in the past twenty years. As to the behavior based requirements modeling, the importance and validity of it has also recognized by many researchers from academia and industry [13-20].

Ayaz *et al*. propose a behavioral specification language for complex systems—Viewcharts, which extends Statecharts to include behavioral views and their compositions [13]. And they define the syntax of viewcharts as attributed graphs and describe dynamic semantics of viewcharts by object mapping automata [14]. Viewcharts notation allows views to be specified independent of each other, which is similar to BDL. A difference between this work and ours is that Viewcharts does not consider behaviors reside in the internal of system, but only observable behaviors from the external system.

Assem proposes an event-oriented requirements definition approach named Behavioral Pattern Analysis Approach (BPA) [15]. In BPA, Event is the primary object of the world model. And it use the so-called BPA Behavioral Pattern, which is the template that one uses to model and describe an event, takes the place of the use case in the UML. BPA is a more effective alternative to use cases in modeling and understanding the function requirements. However, BPA is special for real-time systems, multi-agent systems and safety-critical systems. Besides, it lacks clear links among Behavioral Patterns and can't be used for modeling complex system and is not convenient for requirements verification. On the contrary, our approach definitely labels the relationships of scenarios, viewpoints, and sub-domains, can effectively

```
SC _ InvalidScanCard
SCBEGIN
  ABEH:
    ICreaderDisp1 : display(ICreader, screen)
                    OUTTo(screen)(Prompt="Please scanning card"),
    DoorConWait:idle(),          //waiting user to scan card
    ScanC:scancard(person, IC),
    ReadC:read(ICreader, IC)
          OUTTo(SendCInfo)(ICNo, username),
    SendCInfo:send(ICreader, Mainframe)
              OUTTo(VP_ICInfo_Manage)(ICNo, username),
    //receive the verification result of IC, and the result is store in the viewpoint's data pool
    RecVerInfo:receive(ICreader, Mainframe)
                INFrom(datapool)(result),
    PlayWarnSound:play(ICreader,speaker)
                     OUTTo(speaker)(soundfile),
    ICReaderDisp2:display(ICreader, screen)
                     OUTTo(screen)(Prompt="Overdue IC card!"),
    ICReaderDisp3:display(ICreader,screen)
                    OUTTo(screen)(Prompt="The IC card has reported be lost!"),
    ICReaderDisp4:display(ICreader,screen)
                     OUTTo(screen)(Prompt="Invalid user, unknown reason!");;
  BEH:
    BehInvalidUResp=
      PlayWarnSound□
      If result="overdue"
      Then  ICReaderDisp2
      Else  If result="lost"
            Then  ICReaderDisp3
            Else  ICReaderDisp4
            Fi
       Fi,
    BehInvalidU=
      ICreaderDisp1;
      DoorConWait;
      ScanC;
      ReadC;
      SendCInfo;
      RecVerInfo;
      BehInvalidUResp;    //if the IC is invalid, don't open the door
      Return(ICreaderDisp1);;
    SBehID=BehInvalidU;;
SCEND
```

Figure 7. Scenario behavior model without the right to open the door

```
VP _ ScanCard
VPBEGIN
  datapool;;      //the data pool of this viewpoint
  SC_ValidScanCard,
  SC_InvalidScanCard;;
  SC_Relationship={<SC_ValidScanCard ◊ SC_InvalidScanCard>};;
VPEND
```

Figure 8. Viewpoint behavior model of the scanning card

support the modeling and verification of large and complex system.

Khairuddin *et al.* propose a requirements notation RNSMA and a behavioral approach to specify interactive multimedia applications [16]. RNSMA is based on Petri Net, but its semantics are extended to support reactive systems. In RNSMA, transitions due to events are subdivided into automatic, user and clock. The transitions due to tasks to be done are subdivided into animate, image, sound, text and video. RNSMA uses an extremely simple syntax, which can be read even by novices as a form of pseudo-code. Compared with RNSMA, our work can support general-purpose requirements modeling, not special for stand-alone multimedia applications.

UML is a general-purpose and most famous modeling language for software engineering, which is standardized by OMG [1]. Requirements modeling manner in UML consists of the use case diagram, sequence diagram, state diagram and activity diagram. UML provides standard notation for modeling software analysis and design. But a common and fair criticism of UML is that it is gratuitously large and complex, imprecise semantics, and a dysfunctional diagram interoperability standard (XMI). As another OMG standard, SysML acts as a general-purpose modeling language for systems engineering applications [17]. SysML is based on UML, and it reduces UML’s size and software bias while extending its semantic to model requirements and parametric constraints. These capabilities are essential to support requirements engineering and performance analysis.

Besides, there are some researches based on UML and SysML. Luigi *et al.* propose combining problem frames and UML to describe software requirements in order to improve the linguistic support for problem frames and the UML development practice by introducing the problem frames approach [18]. Pietro *et al.* propose the integration of SysML and problem frames by presenting how a set of well known problem frames can be represented by means of SysML [19]. Atle *et al.* propose to extend UML sequence diagrams to model trust-dependent behavior with the aim to support risk analysis [20]. All of these researches are good for the enhancement of behavior modeling.

In addition, there are many kinds of formal languages and techniques for requirements modeling, especially for behavior requirements. Most of them are based on state or event. Some are standardized by different international organization, such as Z [3], E-LOTOS [4]. Others may be very famous in industry, such as B [21], VDM [22]. Although formal languages and techniques have many advantages, it is difficult to put into practices totally. On the contrary, our approach can be easily used to transform natural language requirements to formal requirements model because the syntax of BDL and the structure of BRM are very simple and clear.

6. Conclusions and Future Work

Software requirements modeling from the perspective of behavior can not only supports the description and modeling of function requirements but also supports the analysis and deduction of non-function requirements. As a lightweight formal requirements description language and model, BDL & BRM can help to smoothly transfer the user requirements expressed by natural languages to formal requirements model expressed by BDL. And the formal model BRM is also good for subsequent requirements verification and validation. Hence, BDL & BRM can effectively bridge the gap between practicability and rigorousness of formal requirements modeling language and technique. Several completed case studies also testified this kind of feature of BDL & BRM.

Currently, we have realized the prototype requirements modeling tool and experimented some case studies. Future works will mainly focus on to define all kinds of requirements properties based on BDL&BRM, and to design and implement corresponding automatic analyzing and deducing methods.

7. Acknowledgements

This work is supported by the National High Technology Research and Development Program of China under contract 2007AA01Z185, the Open Research Foundation of Chinese State Key Laboratory of Software Engineering under grant SKLSE20080702, and the SZU R/D Fund 200747.

REFERENCES

[1] Object Management Group, “OMG Unified Modeling Language (OMG UML) Version 2.0,” 2005.

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, “Introduction to automata theory, languages, and computation,” 2nd Edition, Pearson Education, 2000.

[3] “Information technology—Z formal specification notation — Syntax, type system and semantics,” ISO/IEC 13568, 2002.

[4] “Information processing systems—Open systems interconnection—Enhancements to LOTOS—A formal description technique based on the temporal ordering of observational behavior,” ISO/IEC 15437, 2001.

[5] J. L. Peterson, “Petri net theory and the modeling of systems,” Prentice Hall, 1981.

[6] R. Milner, “Communicating and mobile systems: The Pi-calculus,” Cambridge University Press, 1999.

[7] J. P. Bowen and M. G. Hinchey, “Ten commandments of formal methods... ten years later,” IEEE Computer, Vol. 39, No. 1, pp. 40–48, January 2006.

[8] J. Kong, K. Zhang, J. Dong, and D. Xu, “Specifying behavioral semantics of UML diagrams through graph transformations,” Journal of Systems and Software, Vol. 82, No. 2, pp. 292–306, April 2009.

[9] C. Attiogbe, P. Poizat, and G. Salaun, "A formal and tool-equipped approach for the integration of state diagrams and formal datatypes," IEEE Transactions on Software Engineering, Vol. 33, No. 3, pp. 157–170, March 2007.

[10] M. Hinchey, M. Jackson, and P. Cousot, J. P. Bowen, and T. Margeria, "Software engineering and formal methods," Communications of the ACM, Vol. 51, No. 9, pp. 54–59, September 2008.

[11] G. Kotonya and I. Sommerville, "Requirements engineering with viewpoints," Software Engineering Journal, Vol. 11, No. 1, pp. 5–18, January 1996.

[12] A. Sutcliffe, "Scenario-based requirements engineering," in Proceedings of the 11th IEEE International Conference on Requirements Engineering, Monterey, California, pp. 320–329, September 2003.

[13] A. Isazadeh, D. A. Lamb, and G. H. MacEwen, "Viewcharts: A behavioral specification language for complex systems," Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems, Honolulu, Hawaii, pp. 208–215, April 1996.

[14] A. Isazadeh and J. Karimpour, "Viewcharts: Syntax and semantic," Informatica, Vol. 19, No. 3, pp. 345–362, March 2008.

[15] A. E. Ansary, "Requirements definition of real-time system using the Behavioral Patterns Analysis (BPA) approach: The elevator control system," Proceedings of the Second International Conference on Software and Data Technologies, Barcelona, pp. 371–377, July 2007.

[16] K. Hashim and J. Yousoff, "A behavioral requirements specification approach for interactive multimedia applications," Proceedings of the 19th Australian Conference on Software Engineering, Perth, West Australia, pp. 696–699, March 2008.

[17] "OMG Systems Modeling Language (OMG SysML) Version 1.1," Object Management Group, 2008.

[18] L. Lavazza and V. D. Bianco, "Combining problem frames and UML in the description of software requirements," In: B. Luciano, H. Reiko, Ed., Lecture Notes in Computer Science, Vol. 3922, pp. 199–213, 2006.

[19] P. Colombo, V. del Bianco, and L. Lavazza, "Towards the integration of sysml and problem frames," Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames, Leipzig, pp. 1–8, May 2008.

[20] A. Refsdal and K. Stolen, "Extending UML sequence diagrams to model trust-dependent behavior with the aim to support risk analysis," Science of Computer Programming, Vol. 74, No. 1–2, pp. 34–42, January 2008.

[21] S. Schenider, "The B-method: An introduction," Palgrave, 2001.

[22] C. B. Jones, "Systematic software development using VDM," Prentice Hall, 1990.