2012 4th International Conference on Computer Engineering and Technology (ICCET 2012)
IPCSIT vol.40 (2012) © (2012) IACSIT Press, Singapore

Designing a Domain Specific Language for UML Activity Diagram

Charoensak Narkngam and Yachai Limpiyakorn [+]

Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand

Abstract. Activity diagrams are widely used for describing procedural logic, business processes, and workflows. However, it is difficult to accurately model the behaviors of the large-scale and complex systems. This paper presents a method for generating activity diagrams using a domain specific language, called action description language. Validation and verification rules are defined to prevent data and behavior inconsistency, as well as nonconformance to UML specification, respectively.

Keywords: domain specific language, activity diagram, process modelling.

1. Introduction

Activity diagrams are widely used in various domains, especially in the area of software engineering. The diagrams can be used for describing procedural logic, business processes, and workflows, with the focus on both sequential and parallel behaviours [1, 5]. However, for the large-scale and complex systems, it is difficult to create the activity diagrams that accurately describe the behaviours of the systems, and conform to UML specification [2]. Moreover, the construction of complex activity diagrams is error prone and resource consuming.

The design of activity diagrams with graphic notation can be incomplete, inconsistent, and incorrect. This research proposes an approach to generating activity diagrams from a specification language that could prevent data and behaviour inconsistency, as well as non-conformance to UML specification. Domain Specific Language (DSL) is a specification language dedicated to a particular problem domain. This paper designs a DSL for activity diagrams called Action Description Language (ADL).

2. Domain Specific Language

DSL contains the syntax and semantics that model a concept at the same level of abstraction provided by the problem domain [3]. Some DSLs use a parser to populate a semantic model or an object model, as it provides a clear separation of concerns between parsing a language and the resulting semantics [4]. A semantic model can be used to generate results such as activity diagrams and XML Metadata Interchange (XMI) that are used for documentation and data analysis, respectively. Figure 1 shows the transformation from a DSL script to the target artefact.

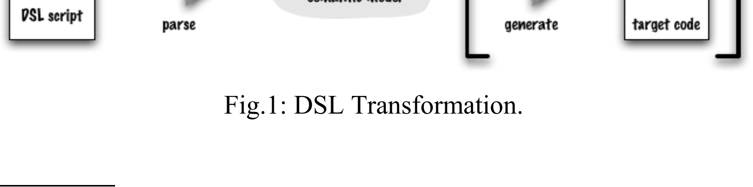


Fig.1: DSL Transformation.

[+] Corresponding author. Tel.: + 662-218-6968; fax: + 662-218-6955.
E-mail address: Yachai.L@chula.ac.th.

3. Activity Diagram

Activity diagrams are the blueprints used for describing system behaviours. The diagram contains activities and flows which can be separated into control flows and object flows. There are seven levels of activities including fundamental, basic, intermediate, complete, structured, complete structured, and extra structured. In this work, ADL is designed to cover the levels of fundamental, basic, and intermediate.

The fundamental level defines activities as containing nodes which includes actions [2]. In this level, activity, activity group and action are introduced where: activity is a containment which consists of name and actions; activity group is containment but it is not used in this level, that is, it is not shown in the diagram.

The basic level includes control sequencing and data flow between actions [2]. In this level, control flow, object flow, object node, activity parameter node, initial node and final node are introduced where: control flow is a sequence of two actions; object flow is an edge which has an action and an object node at either end; activity parameter node is an object which is used as an activity parameter; initial and final node are the point which the activity is started and terminated, respectively.

The intermediate level supports modelling of activity diagrams that include concurrent control and data flow, and decisions, of which the metamodel is shown in Figure 2 [2]. In this level, fork node, join node, decision node, merge node, flow final node, and activity partition are introduced where: fork and join node is used to create concurrent flows and synchronize concurrent flows, respectively; decision and merge node are used to create alternative flows and accept one of alternative flows, respectively. flow final node is the point which the flow is terminated; activity partition is a containment which contains actions and sub partitions.

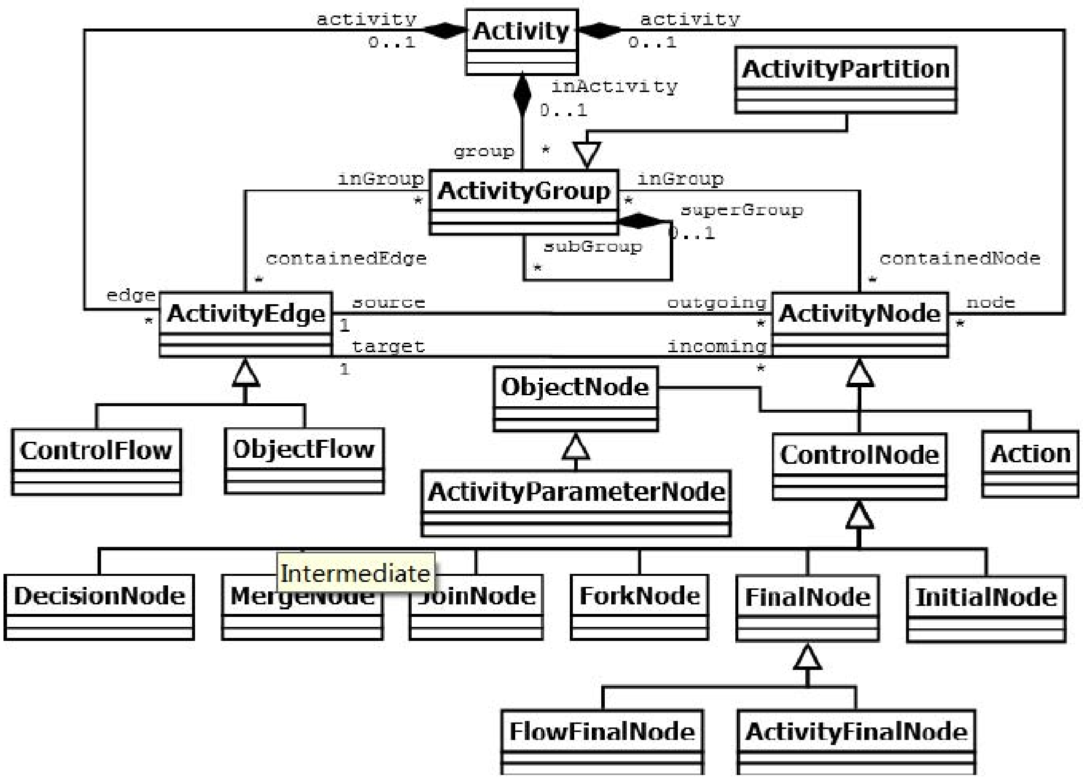


Fig.2: Intermediate Activity Diagram Metamodel.

4. Action Description Language

This section describes the syntax of ADL designed in this research. Activity diagram metamodel must be defined to support the creation of semantic models from the parsed ADL scripts. Validation and verification rules are asserted to prevent data and behaviour inconsistency, as well as non-conformance to UML specification, respectively.

Since the fundamental level supports activities and actions defining where an activity is a containment for actions. A custom syntax with example is created as follows:

```
diagram                                          action sendInvoice end
```

```
02     activity processOrder
03         action receiveOrder end
04         action fillOrder end
05         action shipOrder end
06         action closeOrder end
08         action makePayment end
09         action acceptPayment end
10     end
11 end
```

To avoid noisy syntax, an identifier is used as an action and its name. An identifier should be unique and refers to an activity node which can be object node, control node, or action. An example is shown as follows:

```
01 diagram
02     activity processOrder
03         receiveOrder
04         fillOrder
05         shipOrder
06         closeOrder
07         sendInvoice
08         makePayment
09         acceptPayment
10     end
11 end
```

The main concept of the basic level is flows defining (control and object flows) where: control flow is a sequential of two actions; object flow is passing an object between two actions. Thus, "then" and "with" are used to create grammar for control flows and object flows. "and" is used to reduce repetition of scripts. An example is shown as follows in Figure 3.

```
01 diagram
02     activity processOrder requires Invoice
03         initiate then receiveOrder with RequestedOrder
03         receiveOrder then fillOrder
04         fillOrder then shipOrder and sendInvoice
05         shipOrder then closeOrder
06         sendInvoice then makePayment with RequestedOrder
07         makePayment then acceptPayment
08         acceptPayment then closeOrder
09         closeOrder then terminate
10     end
11 end
```

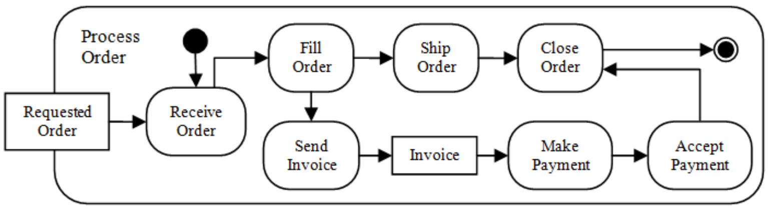


Fig.3: Example Basic Activity Diagram.

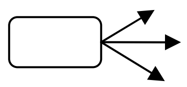
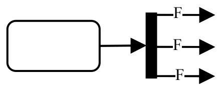
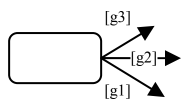
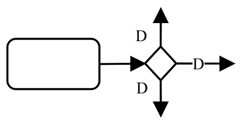
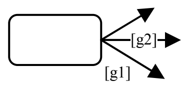
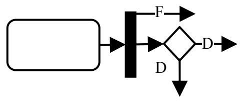
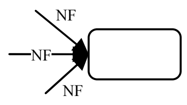
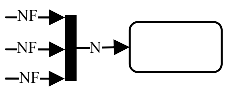
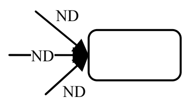
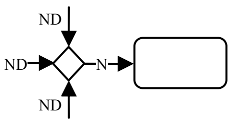
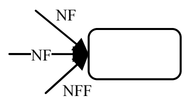
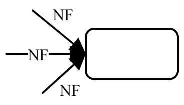
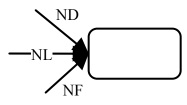
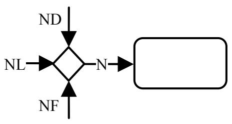
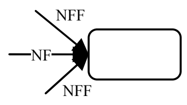
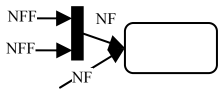
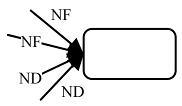
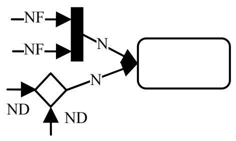
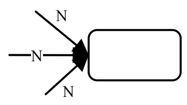
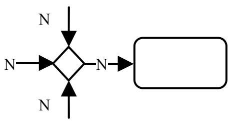
To create intermediate level activity diagrams which support concurrent and alternative flows, behaviours are derived from the basic level to construct control nodes by using patterns in Table 1. Fork and decision nodes can be detected from multiple outgoing flows. Since decision nodes require at least two guard conditions, square brackets are used as labelled edge which is a guard condition. An example is shown as follows:

```
03 receiveOrder then fillOrder [order accepted] and closeOrder [order rejected]
```

To construct join and merge nodes, it requires the inspection of the flow on each incoming edge, namely depth of flow and flow type of which the possible values are N (normal), F (fork), D (decision), L (loop), and T (terminate). The flow type can be determined from the derivative of control types except L and T. In turn, L and T can be determined from the edge which has an ancestor node and terminates as the destination, respectively.

Based on the script in Figure 3 modified by the above script, edges can be created as follows: intiate → receiveOrder; receiveOrder → fillOrder labelled with "order accepted"; receiveOrder → closeOrder labelled with "order rejected"; fillOrder → sendInvoice; fillOrder → shipOrder; makePayment → acceptPayment; sendInvoice → Invoice; Invoice → makePayment; acceptPayment → closeOrder; shipOrder → closeOrder where initiate denotes initial node and Invoice denotes an object. The resulting diagram is shown in Figure 4.

Tab.1: Patterns for constructing control nodes.

Pattern	**Name**	**Description**	**Pattern Result**
	Fork	All outgoing edges are normal edges.	
	Decision	All outgoing edges are labelled edges.	
	Fork-Decision	Some outgoing edges are normal edges and some are labelled edges.	
	Join	All incoming flows are F (Fork) type.	
	Merge	All incoming flows are D (Decision) type.	
	Invalid	An incoming deepest flow has no pair.	
	Different	All incoming flows have no pair.	
	Nested Control	Some incoming flows have different level.	
	Set	All incoming flows in the same level have more than one group.	
	Normal	All incoming flows are N (Normal) type.	

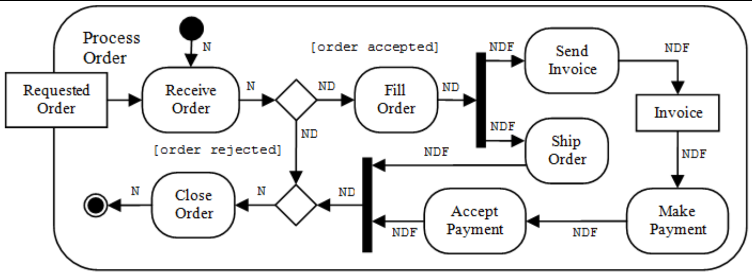


Fig.4: Example Intermediate Level Activity Diagram with Flow Details.

Example validation rule is: An action must be unique to prevent duplicate data that can lead to inconsistent processes. Let a_{new} be a new action, and i_{new} be the index of a_{new}. A new action can be created, given $I = \{i \mid i \in a \wedge a \in A\}$ then $I \cap i_{new} = \emptyset \Longrightarrow A' = A \cup a_{new}$ is asserted. Example verification rule is: An object should be hidden if it is adjacent to control nodes, that is, Given $N1 \rightarrow O \rightarrow N2$ and O denotes an implicit object, then $N1 \rightarrow O \rightarrow N2 \overset{def}{\equiv} N1 \rightarrow N2$ (Determination of the relation of three nodes connected with two arcs is used in this work).

5. Summaries

In this paper, a domain specific language for UML activity diagrams, called action description language, is introduced with the assertion of validation and verification. The syntax of ADL covers the activities at the levels: fundamental, basic, and intermediate. Generating activity diagrams using specification languages could reduce resource consumption, and promote process consistency as well as conformance to specification, compared to graphic notation approach.

6. References

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. 2nd Edition. Addison-Wesley Professional, 2005.

[2] OMG. *Unified Modeling Language™(OMG UML), Superstructure Version 2.3*. Object Management Group, Inc. 2010, pp. 303-430.

[3] Debasish Ghosh. *DSLs in Action*. Manning Publications Co., 2011.

[4] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[5] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd Edition. Addison-Wesley, 2003. pp. 117-130.