Ontology Based Semantics Checking for UML Activity Model

Zhixue Wang, Hongyue He, Li Chen and Ying Zhang
Institute of Command Automation, PLA University of Science and Technology,
Nanjing 210007, China

Abstract: UML activity model is mainly used to model the behaviors of software system and the quality of activity model will influence the quality of software system. But because the UML activity model lacks strictly formal semantics, it is difficult to make formal semantics analysis and checking for activity model. An ontology based method of semantics checking for activity model is proposed. The semantics of activity model is divided into static semantics and dynamic semantics. The static semantics is transformed into OWL DL by an algorithm, and the dynamic semantics is described by DL-Safe rules. Then the consistency of UML activity model is analyzed and some model checking rules are defined, which enables model consistency checking by using an ontology reasoning tool.

Key words: Activity model, semantics checking, OWL DL, DL-Safe

INTRODUCTION

UML which is a modeling language recommended by the OMG provides several views to software designers for modeling and analyzing the different aspects of software system (Booch et al., 2005). And UML is the important part of MDS (Model-Driven Software Development) which is a popular method of software development (Stahl et al., 2006). Based on the above characteristic, UML is widely used in the software development. Especially, the UML activity model is mainly used to model the behaviors of software system. The behaviors of software system compose the functions of software system. So, the quality of activity models finally influences the quality of software system. A software system, especially system of systems, may have several activity models made by different designers focusing on different functions. The descriptions of one object in different activity models may have inconsistency such as redundancy and mistake. For example, Allan makes a shopping activity model of online shopping system which describes that the client must pay before the system creates the bill of delivery; but James make a delivering activity model of this system which describes that the client pay for delivery after he receives the delivery. So, the two activity models have inconsistency.

As the inconsistency of activity model will make fatal mistakes if it exists in the phases of software programming, the inconsistency must be dealt by consistency checking in the phases of software modeling (Lange et al., 2003). But the UML activity model lacks strictly formal semantics, it is difficult to make formal semantics analysis and consistency checking for activity model. And the current UML tools mostly don’t support to check the consistency of activity model.

RELATED WORK

The current inconsistency checking researches can be classified by representation into three categories (Usman et al., 2008): formal representation, extended UML representation, or non-intermediate representation. In formal representation technology, the UML models are represented by a formal language or a formal notation such as Object-Z, Petri net etc. (Kim and Carrington, 2004; Shinkawa, 2006). The extended UML representation technology needs an extension in UML diagram by using the UML Profile mechanisms or Meta-Modeling technology (Mens et al., 2005). The non-intermediate representation technology manipulates the UML models directly by some rules or algorithms in order to discover potential errors, mistakes or inconsistencies (Briand et al., 2006; Egyed, 2006, 2007).

OWL (Web Ontology Language) recommended by W3C (World Wide Web Consortium) is an ontology language with formally defined meaning and provides definitions of classes, properties, individuals and data values in an ontology. OWL DL is a subset of OWL and well expressive, and the inference of OWL DL ontology is decidable. DL-Safe rule is a subset of Horn Rule and the combination of OWL DL and DL-Safe rules is decidable.
(Horrocks and Pateli-Schneder, 2004). Based on the above, the OWL DL ontology and DL-Safe rules can be used to checking the activity model.

**FORMALIZATION OF META MODEL**

According to the OMG's four layered meta-model architecture, a class in model level (M1) is an instance of the meta-class in meta-model level (M2)(OMG, 2004). And in the realm of DL-based ontologies (Baader et al., 2003), an ontology can be captured by the terminology box (T-Box) and the assertion box (A-Box). The T-Box captures the knowledge about the class level and the A-Box captures the knowledge about specific instances in the instance level. So the activity meta model can be formalized into T-Box and the activity model can be formalized into A-Box. Figure 1 shows a simplified activity meta-model which covers the primary notations and constraints used in activity models.

According to the activity meta model (Fig. 1) it isn't wise to abstract a class in T-Box for every meta-class, because some meta-class is the superclass of other meta-classes which are mostly used in model, such as ObjectNode. So, the superclass can be omitted. And some meta-class is an association class connecting two elements according to MOF, such as ActivityEdge. A relation should be abstracted for an association class in meta-model. Based on the above analysis, the classes and relations in T-Box should be abstracted for activity meta-model as follow:

According to the structure of T-Box (Baader et al., 2003), An algorithm-Build-T-Box which can construct a T-Box according to Table 1 and 2 is designed as follow:

<table>
<thead>
<tr>
<th>Table 1: Classes of Activity meta model</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta-class in meta-model</td>
</tr>
<tr>
<td>NameElement</td>
</tr>
<tr>
<td>ActivityNode</td>
</tr>
<tr>
<td>Action</td>
</tr>
<tr>
<td>ControlNode</td>
</tr>
<tr>
<td>Pin</td>
</tr>
<tr>
<td>ValueSpecification</td>
</tr>
<tr>
<td>Constraint</td>
</tr>
<tr>
<td>Class</td>
</tr>
<tr>
<td>GuardCondition</td>
</tr>
<tr>
<td>Precondition</td>
</tr>
<tr>
<td>Postcondition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Relations of Activity meta model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation in T-Box</td>
</tr>
<tr>
<td>C-Flow</td>
</tr>
<tr>
<td>O-Flow</td>
</tr>
<tr>
<td>A-Contain</td>
</tr>
<tr>
<td>A-Guard-S</td>
</tr>
<tr>
<td>A-Guard-T</td>
</tr>
<tr>
<td>A-PreCondition</td>
</tr>
<tr>
<td>A-PostCondition</td>
</tr>
<tr>
<td>Belong</td>
</tr>
<tr>
<td>Perform</td>
</tr>
</tbody>
</table>

![Fig. 1: Meta model of activity diagram](image)

---

302

Algorithm: Build-T-Box  
Input: Table 1, Table 2  
Output: T-Box

begin
T-Box = {};  
for all C1 in column of "class in T-Box" in Table 1, C2 is the superclass of C1, do  
T-Box = T-Box ∪ (C1 ⊆ C2);  
end for;
for all C1, C2 in column of "class in T-Box" in Table 1,  
if C1 ⊆ C2 and C1 ∈ T-Box and C2 ∈ T-Box, do  
T-Box = T-Box ∪ (C1 ∩ C2 = ∅);  
end if;
end for;
for all r = (e1, e2) in Table 2, the relation r−1 is the inverse relation of r, do  
if multiplicity constraints is "0..1" then  
T-Box = T-Box ∪ (e1 ∩ e2);  
else if multiplicity constraints is "1..*" then  
T-Box = T-Box ∪ (e1 ⊆ e2);  
else if multiplicity constraints is "1" then  
T-Box = T-Box ∪ (e1 ⊆ e2, e2 ⊆ e1);  
end if;
end for;
for all the multiplicity constraints in domain of every relation r = (e1, e2) in Table 2, the relation r−1 is the inverse relation of r, do  
if multiplicity constraints is "0..1" then  
T-Box = T-Box ∪ (e1 ⊆ e2);  
else if multiplicity constraints is "1..*" then  
T-Box = T-Box ∪ (e1 ⊆ e2);  
else if multiplicity constraints is "1" then  
T-Box = T-Box ∪ (e1 ⊆ e2, e2 ⊆ e1);  
end if;
end for;
return T-Box;
end

FORMALIZATION OF ACTIVITY MODEL

Formalization of static semantics: The static semantics of activity model comprises the notations and their relationships in activity model. According to the activity meta model, an activity model mainly comprises Pin, Action, ActivityNode, Fork, Join, ActivityPartition, ControlFlow and ObjectFlow. The semantics of these notations are defined and interpreted as follow:

Definition 1: The semantics of Pin can be defined as a tuple (s, o) where:

- s is the name of this Pin
- o is the object which is passed through this Pin

Definition 2: The semantics of Action can be defined as a tuple (s, I, m, P) where:

- s is the name of this Action
- I is the set of parameters of this Action
- m is the return value of this Action
- P is the set of pins which cling on this Action

Definition 3: The semantics of ActivityNode can be defined as a tuple (s, ACTION[NODE, A(N)] where:

- s is the name of this ActivityNode
- if this ActivityNode is simple, ACTION is used and A is the set of actions in this ActivityNode
- if this ActivityNode is composed, NODE is used and N is the set of activity nodes in this ActivityNode

In activity model, there are two kinds of ControlNode, i.e., Join and Fork, which are used to describe the parallel activities.

Definition 4: The semantics of Join can be defined as a tuple (N, n) where:

- N is the set of activity nodes which must be done before the ActivityNode n

Definition 5: The semantics of Fork can be defined as a tuple (n, N) where:

- N is the set of activity nodes which will execute after the ActivityNode n

In activity model, activity nodes can be grouped according to the executor, the ActivityPartition implements the grouping.

Definition 6: The semantics of ActivityPartition can be defined as a tuple (r, N) where:

- N is the set of activity nodes which are executed by the instance of the class r

There are two kinds of ActivityEdges, i.e., ControlFlow and ObjectFlow. The ControlFlow connects Activity nodes and the ObjectFlow connects Pins.

Definition 7: The semantics of ControlFlow can be defined as a tuple (n, c, n_t) where:

- n is the source activity node of this ControlFlow
- c is the guard condition which must be true when this ControlFlow is activated
- n_t is the target activity node of this ControlFlow

Definition 8: The semantics of ObjectFlow can be defined as a tuple (p, o, p_t) where:

- $p_i$ is the source pin of this ObjectFlow
- $o$ is the object passed by this ObjectFlow
- $p_i$ is the target pin of this ObjectFlow

To express the execution sequence of activity nodes, Flow List is defined as follow:

**Definition 9:** The semantics of FlowList can be defined as a tuple $(f_1, \ldots, f_n)$ where:

- $f_i$ is the controlFlow or ObjectFlow
- The target end of $f_i$ is the same as the source end of $f_{i+1}$

To describe the relationships between flow and FlowList, two symbols $e_i$ and $c_i$ are defined as follow:

**Definition 10:** $e_i$: if $f_i \in$ FlowList, $e_i(f_i, f_{i+1} = c_i)$, if and only if $\exists e_i \in f_i, f_i = f_{i+1}$

**Definition 11:** $c_i$: if $f_i \in$ FlowList, $c_i(f_{i+1} = \{c_{i+1}, \ldots, c_n\})$, if and only if $\forall c_{i+1} \in f_i, f_i = f_{i+1}$

Base on the above analysis, the static semantics of activity model can be defined as follow:

**Definition 12:** The static semantics of activity model can be defined as a tuple $(N, P, CF, OF, F, J, FL)$ where:

- $N$ is the set of activity nodes in this activity model
- $P$ is the set of activity partitions in this activity model
- $CF$ is the set of controlFlows in this activity model
- $OF$ is the set of objectFlows in this activity model
- $F$ is the set of forks in this activity model
- $J$ is the set of Join in this activity model
- $FL$ is the set of FlowList in this activity model

To make the static semantics of activity model valid, integrated and non-redundancy, some constraint are defined as follow:

- Every FlowList in FL must be valid, otherwise the semantics is invalid. A FlowList is valid if every flows in it is ControlFlow or ObjectFlow and the target end of flow is the same as the source end of flow.
- Every flow in CF or OF must be contained in a FlowList in FL, otherwise the semantics isn’t integrated.
- If there are two flowlist, and flowlist, flowlist, then the semantics is redundant.

According to the structure of A-Box (Baader et al., 2003), an algorithm Build-A-Box which can construct a A-Box according to the static semantics of activity model is designed as follow:

- **Algorithm:** Build-A-Box
- **Input:** The static semantics of activity model
- **Output:** A-Box

begin
A-Box = {}; for all elements in the static semantics of activity model, do
Add the expression of the assertion describing that the element belongs to one class into the A-Box;
end for;
for all couples of elements in the static semantics of activity model, do
Add the expression of the assertion describing that the couple of elements belongs to one relation into the A-Box;
end for;
return A-Box;
end

**Formalization of dynamic semantics:** The dynamic semantics of activity model is the rules followed when the activity nodes are executing. For example, if one activity node has been done, and the guard condition is true, then the other activity node can execute. To describe these rules, three subclasses of ActivityNode, i.e., NodeDo, NodeDoing and NodeDone, are defined. NodeDo means that the activity node is ready to execute; NodeDoing means that the activity node is executing; NodeDone means that the activity node has been done. These rules are described by DL-Safe rules as follow:

- **Rule 1:** NodeDoing $(a) \land C-Flow (b,a) \land GuardCondition (c) \land A-Guard-T (c,b) \land A-Guard-S (c,a) \land O(a) \land O(b) \land O(c)$, it means that the action $a$ is executing, there is a ControlFlow connecting $a$ and $b$, if the GuardCondition $c$ is true, then the action $b$ will be executing next step.

- **Rule 2:** NodeDone $(a) \land NodeDoing (b) \land C-Flow (a,b) \land O(a) \land O(b)$, it means that there is a ControlFlow connecting $a$ and $b$, if the action $b$ is executing, then the action $a$ has executed.

The $O(a)$ in the rules means that $a$ is an instance in the A-Box. The DL-Safe rules can be expressed by SWRL and then add to the OWL DL ontology. Finally, we construct an ontology using the OWL DL and SWRL, and the ontology contains the semantics of activity models.

**SEMANTICS CHECKING**

**Consistency checking:** According to the meta model, there are multiplicity constraints for the source end and
target end of every relationship. If the activity model
disobey these constraints, then the activity model is
inconsistent. These constraints are formalized to the
multiplicity for domain and range of relationship in the
ontology. If the activity model is inconsistent, then the
ontology containing the semantics of activity models is
also inconsistent. This kind of inconsistency in ontology
can be checked by reasoning of ontology according to
the following rules:

- [R1] a pin aggregates to one activity node
- [R2] an activity node can be only contained by other
  one activity node
- [R3] a guard condition guards only one control flow
- [R4] an object is passed by one object flow
- [R5] an activity node is executed by only one class
- [R6] a precondition or postcondition can be only
  used to one activity node

The reasoning of ontology according to the above
rules can be implemented by Pellet which is a reasoning
tool based on the algorithm-Tableaux.

**Integrity checking:** The integrity of model is as
important as consistency of model. A non-integral model
may not implement some function, so it should be
checked in the process of model checking. In the open
world assumption, the ontology containing the semantics
of non-integral activity model is consistency, but it is
non-integral. The integrity of ontology can be checked
by querying. Several querying rules expressed by
SPARQL are defined as follow:

- [Q1] There is an activity node which can’t execute in
  the activity model. The SPARQL querying rule is:
  UNSAID {?a rdf:type xml:ns: NodeDoing}; } }
- [Q2] There is an activity node which has not done in
  the activity model. The SPARQL querying rule is:
  UNSAID {?a rdf:type xml:ns: NodeDone}; }
- [Q3] There is a invalid object flow in the activity
  model. The SPARQL querying rule is: SELECT ?a
  ?b. UNSAID {?a xml:ns: C-Flow ?b}; }
- [Q4] There is an activity node which contains no
  action, except Start node and End node. The
  SPARQL querying rule is: SELECT ?a WHERE { ?a
  rdf:type xml:ns: Node. UNSAID {?a xml:ns: Contain
  ?b}; }
- [Q5] In the activity model, there is an instance of one
  class which is not defined in the class model. The
  SPARQL querying rule is: SELECT ?a WHERE { ?a
  xml:ns: Perform ?b. UNSAID {?a rdf:type
  xml:ns:Role}; }

The reasoning tool-Pellet can query ontology using
the above querying rules and then return the result to the
model designer.

**A CASE STUDY**

Figure 2 describes a simple case for shopping, there
are three executors Customer, Sales and Warehouse in the
model. At first, the Customer chooses the merchandise,
and the Sales make an order for these merchandises. Then
Warehouse pulls the merchandise according to the order,
and Sales send the bill to the Customer. When the
Customer pays for the bill, the Sales closes the order,
and a process of shopping is finally finished.

In the case, if the control flow connecting Pay bill
and Close order is disappeared, the Sales don’t know
when the Customer pays for the bill. So the Sales will wait
for the message that the Customer has paid for the bill.
And the Close order is ready to execute and can’t cute. So
the model is non-integrality and it can’t implement the
process of shopping. We use Pellet to query the ontology
containing the semantics model using the querying rule
Q1. Figure 3 is the result which describes that Close order
can’t execute.

![Fig. 2: Example of shopping](image-url)
CONCLUSION

Just because UML is a semi-formal model language and UML activity models contain action semantics, it is difficult to check the UML activity models. We propose a method of ontology-based semantics checking for UML activity models. The OWL DL and DL-Safe rules are used to formalized the semantics of activity model which are divided into static semantics and dynamic semantics. Then the consistency and integrality of activity model are checked by reasoning and querying of ontology using rules. The method is extendable, when the rules is rich, more and more consistency and integrality can be checked.

REFERENCES