Formal Definition of Measures for UML Statechart Diagrams Using OCL

Luis Reynoso
Department of Computer Science, University of Comahue
Buenos Aires 1400, 8300, Neuquén, Argentina
+54 02994490313
lreynoso@uncoma.edu.ar

Juan Antonio Cruz-Lemus, Marcela Genero, Mario Piattini
Alarcos Research Group
Department of Technologies and Information Systems, University of Castilla-La Mancha,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
+32 926295300 - Ext. 3740, 13071
{JoseAntonio.Cruz, Marcela.Genero, Mario.Piattini}@uclm.es

ABSTRACT
The informal definition of a measure in natural language is ambiguous, so it must be accompanied by a precise and formal definition, for avoiding misunderstanding and misinterpretation. In this paper we show the formal definition of measures for UML statechart diagrams using OCL, upon the UML statechart metamodel. The use of a formal definition upon a metamodel (where the main concepts and relationships are modelled) assure that measures capture the concepts they intend for and could facilitate the implementation of measures extraction tools.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *Product metrics.*

General Terms
Statechart Diagrams, Measurement, Metamodeling, UML.

Keywords
Measures, UML, OCL, Statechart Diagrams, Understandability, Structural Properties, Formal Definition, Metamodeling.

1. INTRODUCTION
The quality of UML models should be evaluated through quality indicators or measures [7]. However, when measures are defined in an unclear or imprecise way many difficulties may arise. The lack of precision of what is captured by a measure may produce that the persons who build the measure extraction tool make their own decision during implementation. In this way, they can arrive at incorrect values of the measure. This situation arise when measures are not repeatable (the same result would not be produced each time a measure is repeatedly applied to a same artifact by a different person). Consequently, when measures are not repeatable, quality evaluators of models can take incorrect and undesirable decisions of the external quality attributes of their models. We believe that the understandability of what is captured by the measure should be defined not only in natural language but also in formal language, because how well a measure is understood will influence the way the measure is implemented and used.

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

Software measures can be defined through query operations using the Object Constraint Language (OCL) [4] upon a particular metamodel of the measured software artifact. The usage of the meta modeling approach for defining model-specific measures have been previously introduced for defining class diagram measures [1] and OCL measures [6] upon the UML metamodel. The contribution of this paper is the formal definition of Statechart Diagram (SD) measures using a meta-modeling approach. Even though many proposals of SD measures exist (see [3]), none of them has formally defined the measures using this formal approach.

A thoroughly definition of a set of measures for structural properties of UML SD is presented in [3] based on the hypothesis that structural properties of an UML SD (the software artifacts measured) have an impact on the cognitive complexity of modelers (subjects), and high cognitive complexity leads the UML SD to exhibit undesirable external qualities, such as less understandability or a reduced maintainability [2]. These measures are supposed to be good indicators of the understandability of such diagrams. This fact was empirically validated in [3].

In the next section the UML SD metamodel is briefly introduced. Section 3 provides the formal specification of two measures. Finally, the last section presents some concluding remarks.

2. THE UML SD METAMODEL
The abstract syntax for state machines is expressed graphically in UML SD metamodel [5], which covers all the basic concepts of state machine graphs such as states, transitions, guards, etc. Every state machine has a *top* state, usually a composite state, that contains all the other elements of the entire state machine. The graphical rendering of this top state within an SD is optional.

The *State* hierarchy has a *State* superclass and three subclasses, *CompositeState*, *SimpleState* and *FinalState*. This hierarchy, in fact, is part of the *StateVertex* hierarchy in the SD metamodel,

which also includes the *PseudoState*, *SynchState* and *SubState* classes. The composite state may contain any state of the *StateVertex* hierarchy. All the classes, attributes of classes and relationships previously mentioned are part of the UML SD metamodel [5]. Each *State* in an SD may have associated actions, such as entry, exit or a do-activity actions (see the relationships between the *State* and *Actions* classes with the *entry*, *exit* and *doActivity* rolenames in the SD metamodel [5]).

Transitions usually connect two states, for example two Simple States, a Simple State with a Final State, etc. These connections are described through two relationships between the *StateVertex* and *Transition* classes, where each of them identifies the *source* and *target StateVertex* which is connected through the transition. So, any transition connects exactly a *source* to a *target* S*tateVertex*. Within an SD, transitions may also be labeled with Guard and Events, modelled through the *Guard* and *Event* classes which are related to the *Transition* class. The set of all transitions within an SD is modeled through a relationship between the *StateMachine* and the *Transition* classes.

3. SPECIFICATION OF SD MEASURES

In this section we will present a general overview of the specification of SD measures using the UML SD metamodel [5]. For illustrating our approach we will show the formal definition of two measures.

The specification of the measures relies on three query operations:

1. Alltransitions operation, defined in the *StateMachine* metaclass, obtains the set of transitions in an SD.
2. AllStates operation, defined in the *StateMachine* metaclass, selects the set of all the states within an SD.
3. AllSubStates operation, used by the two previous operations and defined in the *StateVertex* metaclass, obtains the set of all *Subvertex* included in a SD. It is recursively defined.

Their OCL definitions are shown below:

```
context StateMachine::allTransitions::Set(Transition)
body: result =  self.transitions>
        union(self.allSubStates().internaltransitions)
context StateMachine::allStates:Set(State)
body : result = self.top.allSubstates()

context StateVertex::allSubstates::Set(StateVertex)
body: result =if self.oclIsKindOf(CompositeState)
        then self.oclAsType(CompositeState).subvertex->union
        self.oclAsType(CompositeState).subvertex
        -> select (s:StateVertex| s.allSubstates())
        else Set{} endif
```

For obtaining the value of each SD measure (described in [3]) we defined in the *StateMachine* metaclass an operation with the same name as the measure. So, 14 operations were defined, one for each defined measure. Using the *allSubState* operation the value of many SD measures are specified. This operation returns the set of all the states (of different kinds: *Initial*, *Final*, *Simples*, etc.) included in a diagram, even those states which are part of composite states.

For example, selecting from the *allsubstate* operation result, those states of an SD which have associated a *doActivity* action it is possible to obtain the value of the Number of Activity (NA) measure. The quantity of objects selected represents the value of the NA measure.

```
context StateMachine::NA():Integer
body: result = self.top.allSubstates()->select(s | s.oclIsType(State) and
s.doActivity->notEmpty() )-> size()
```

The Number of Transition (NT) measure is specified through the use of the *alltransitions* operation. The cardinality of the set represents the quantity of transitions within a SD.

```
context StateMachine::NT():Integer
body: result = self.allTransitions() -> size()
```

4. CONCLUSIONS

The main contribution of this paper is the formal definition of measures for UML SD proposed in [3] using OCL upon the UML SD metamodel. A formal definition of the measures is useful to avoid misunderstanding and misinterpretation between the stakeholders within the measurement process. This is one of the main requirements to start any measurement program in software development organizations. Refactoring techniques which improve the design of SD (such as [8]) can take advantage of using the formally defined measures of this paper. Measure value can be computed before and after the refactoring (or model transformation) is applied, to express the quality of the diagram [7] and to evaluate a change. Moreover, the formal definition of measures using OCL can be introduced in MDA compliant tools to extract the measures values for UML models.

5. ACKNOWLEDGMENTS

This research is part of the COMPETISOFT project (506AC0287), the MECENAS project (PBI06-0024) and ESFINGE project (MEC, TIN 2006-15175-C05-05).

6. REFERENCES

[1] Baroni, A. L. , Braz, S., Brito e Abreu, F.. Using OCL to Formalize Object-Oriented Design Metrics Definitions. *In Proceedings of QUAOOSE'2002*, Malaga, Spain, 2002.

[2] Briand, L. C., Bunse, L. C., Daly, J. W. A Controlled Experiment for evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. *IEEE Trans. on Softw. Eng.*, Vol. 27 Nº 6, 2001, pp. 513-530.

[3] Cruz-Lemus, J. A. A Measurement-Based Approach for Assessing UML Statechart Diagrams Understandability. *Ph.D. Thesis.* 2007.

[4] Object Management Group. UML 2.0 OCL 2nd revised submission. *OMG Document.* http://www.omg.org

[5] Object Management Group. UML Specification Version 1.4.2*, OMG Document formal04-07-02.* http://www.omg.org

[6] Reynoso, L., Genero, M., Piattini, M. OCL2: Using OCL in the Formal Definition of OCL Expression Measures, In *Proceedings of the 1st Workshop on Quality in Modeling QIM* co-located with the ACM/IEEE MODELs 2006. 2006.

[7] Saeki, M., Kaiya, H. Model Metrics and Metrics of Model Transformation. In *Proceedings of 1st Workshop on Quality in Modeling*, pp. 31-45, Genova, Italy, Oct. 1, 2006.

[8] Sunyé, G., Pollet, D., Traon, Y., Jézéquel, J.: Refactoring UML Models, UML'01: In *Proceedings of the 4th Int. Conference on UML*. 2001. pp. 134--148.