

Towards the UML Evaluation Using Taxonomic Patterns on Meta-Classes

Haohai Ma^{1,2}, Zhe Ji¹, Weizhong Shao¹, Lu Zhang¹

¹Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P. R. China

²Department of Computer Science, Inner Mongolia University, Hohhot 010021, P. R. China
mahh@sei.pku.edu.cn

Abstract

In order to evaluate the design quality of the UML, understanding meta-classes is a key activity as they are the primary weapons by which the UML specifies the application domains. The paper introduces taxonomic patterns for clustering the UML meta-classes based on the observation of their evolution and fitness. The result sets of the patterns assist in finding the evidence of the concerns about the UML design and quality. It not only helps to find out problematic meta-classes, possible design defects of the UML and the inconsistency between the UML meta-models and the application domains; but also provides valuable information for guiding the development and evaluation of the UML. The work can be the basis of further quality analysis of the UML meta-models.

Keywords: Software evaluation, Software Quality, UML, Meta-class

1. Introduction

A typical problem of software evolution is that software systems must evolve to satisfy new requirements. After many enhancements a critical point is reached when the new release decreases the quality and evolvability but increases the complexity of the system. The system becomes so complex and difficult to maintain that new releases require exorbitant costs.

In a way similar to develop arbitrary software, the Unified Modeling Language (UML) has been under development for years: it is adapted to changing requirements of domain modeling and many new features are added as the result of catering for new application development paradigms and the long-term investment effort of companies [6][7]. The continuous

development increases both the size and the complexity of the UML dramatically. Taking the UML 2.0 as an example, its meta-model includes hundreds of meta-classes and up to 1000+ pages documented. There is undoubted a definite need for an effective approach which helps in meta-model understanding and problem detection.

The syntactical view of the UML meta-model is the abstract syntax, which is composed of meta-class diagrams in a similar way as constructing Object-oriented class diagrams [10]. In such a context, understanding meta-classes is a key activity as they are the primary weapons by which applications are built in the modeling process, as well as the primary abstraction into which domains are characterized in the meta-modeling process. Therefore, there is a definite need to support the understanding of meta-classes and their evaluation along with the development of the UML.

Early knowledge representation has adopted taxonomic classification for clustering information [12]. This paper, therefore, introduces taxonomic patterns for identifying the UML meta-classes based on the observation of their evolution and fitness. One part of the patterns is derived from the evolutionary trace of meta-classes, referred to as the *Fixed Star*, *Comet*, *Pulsar*, *New Star* and *White Dwarf* patterns. The *Fixed Star* pattern identifies a category of meta-classes that exist in all UML versions. The *Comet* pattern collects the meta-classes that exist only during one version of the UML. The *Pulsar* pattern labels a category of meta-classes that appear and disappear repeatedly over versions of the UML. The *New Star* pattern identifies the meta-classes that once augmented into the meta-model will be kept in the UML since then. The *White Dwarf* pattern means that the meta-classes are deleted from the UML meta-model after they survive in two versions or more.

Another taxonomic pattern, referred to as the *Island*, identifies a category of meta-classes on the basis of whether they possess enough definitions itself, including attributes, well-formed rules and associations to others. The *Island* pattern can be complementary to other patterns since it predicts meta-classes that exhibit a potential for elimination (to enter the *Comet* or *White Dwarf* category) or refinement (to enter the *Fixed Star* or *New Star* category).

The result sets of the meta-classes based on the patterns assist in finding the evidence of the concerns about the UML design and quality. It not only can identify problematic meta-classes, possible design defects in the UML and the inconsistency between the UML meta-models and the application domains; but also provide valuable information for guiding the development and evaluation of the UML. The work can be the basis of further quality analysis of the UML.

The remaining of this paper is organized as follows. Section 2 overviews what and how the meta-classes are collected. The construction principles of taxonomic patterns and the result sets of meta-classes are presented in section 3 and section 4. Section 5 summarizes the contribution of each pattern to evaluate the UML. Section 6 introduces some related research with the paper. Finally the conclusion of the paper and further works are provided in section 7.

2. Data Collection

The meta-class data are collected from [15], [16], [17], [18], [19] and [20] by four M.S. students who are familiar with the UML specifications. Furthermore, these data have been verified consistency with corresponding versions of the code package pertaining to the UML Meta-Model in the JBOO modeling tool [9], by which reducing the possibility of human errors.

Table 1 summarizes the number of meta-classes in each UML version. The total number of meta-classes is 394, if one meta-class appears in one or several UML versions, it will be counted only once.

Table 1. The number of meta-classes in the UML versions

UML 1.1	UML 1.2	UML 1.3	UML 1.4	UML 1.5	UML 2.0	Total
120	118	133	192	194	260	394

3. Taxonomic Patterns based on the Meta-Class Lifetime

3.1. Lifetime bit-map

In order to keep track of the evolution of meta-classes along with the development of the UML, we use a lifetime bit-map (see Figure.1), in which each 6-bit binary corresponds to a meta-class with one bit per version for the meta-class lifetime. One bit being set means the meta-class exists in the corresponding UML version. For example, bit 0 is set if the meta-class is in the UML 2.0, bit 1 is set if it is in the UML 1.5, and so on.

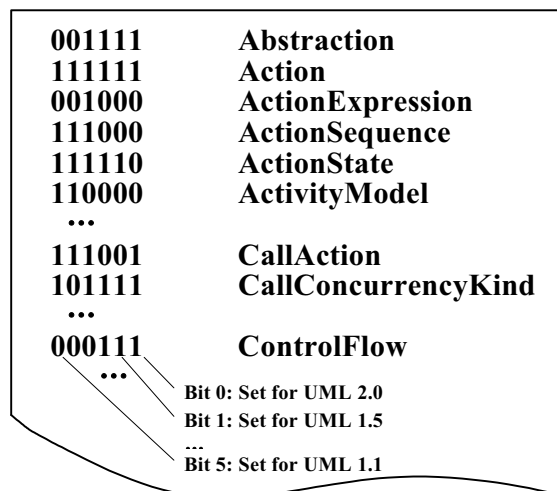


Figure 1. A lifetime bit-map for meta-classes

3.2. Patterns

Based on the observation of the lifetime bit-map, especially where “0”s and “1”s locate in the 6-bit binary of a meta-class, we can summarize the following five taxonomic patterns. We adopt the vocabulary of the domain of astronomy and assign them to the patterns for purpose of the memory and communication, since the names from this domain convey well the described patterns. Similar names with different meaning are also used in [8].

Fixed Star. The *Fixed Star* pattern identifies a category of meta-classes whose 6-bit evolutionary binary is “111111”, that is, the meta-classes survive in all UML versions, such as the meta-class of *Action* in Figure 1. There are total 42 meta-classes in this category listed in Table 2.

Comet. The *Comet* pattern collects a category of meta-classes that there is only one “1” in their 6-bit evolutionary binary, that is, the meta-classes conforming to the pattern have a very short lifetime, and exist only during one version of the UML, such as the meta-class of *ActionExpression* in Figure 1. Although there are

total 172 meta-classes in the category most of them are those newly added into the UML 2.0. We rule out the newcomers to the UML 2.0 in order to let our category be more meaningful. Table 3 lists the rest meta-classes in the category.

Table 2. The Fixed star meta-classes

Meta-classes in all versions (Total 42)		
Action	Dependency	ParameterDirectionKind
Actor	Element	PseudoState
AggregationKind	Expression	PseudostateKind
Association	Feature	Reception
AssociationClass	Generalization	Signal
BehavioralFeature	Integer	State
Boolean	Interaction	StateMachine
Class	Interface	String
Classifier	Message	StructuralFeature
Collaboration	Model	TimeExpression
Comment	Namespace	Transition
Component	Operation	Usage
Constraint	Package	UseCase
DataType	Parameter	VisibilityKind

Table 3. The Comet meta-classes

Meta-classes	Only Exists in
ActionExpression	UML1.3
ImplementationClass	UML1.3
IterationExpression	UML1.3
CallProcedureAction	UML1.5

Pulsar. The *Pulsar* pattern labels a category of meta-classes that there is the alternation of “0” (“0”s) and “1” (“1”s) in their 6-bit evolutionary binary, that is, the meta-classes appear and disappear repeatedly during the development of the UML, such as the meta-class of *CallAction* and *CallConcurrencyKind* in Figure 1. There are total 6 meta-classes in this category listed in Table 4.

New Star. The *New Star* pattern identifies a category of meta-classes that in their 6-bit evolutionary binary “0” only appears in the left side and more than one “1” in the right side. It means that the meta-classes are augmented into the meta-model during the development of the UML and kept since then, such as the meta-class of *Abstraction* and *ControlFlow* in Figure 1. According to our analysis of meta-classes lifetime, there are only two *New Star* patterns found in

meta-classes evolution, i.e. the patterns of “001111” and “000111”. The former means the meta-classes originated from the UML 1.3 and the latter from the UML 1.4, seeing Table 5.

Table 4. The Pulsar meta-classes

Meta-classes	Originated from	Deleted in	Resumed from
CallAction	UML11	UML14	UML20
CallConcurrencyKind	UML11	UML12	UML13
Enumeration	UML11	UML13	UML14
EnumerationLiteral	UML11	UML13	UML14
Node	UML11	UML12	UML13
Primitive	UML11	UML13	UML14

Table 5. The New Star meta-classes

Originated from 1.3 (Total 9)	Abstraction, ElementImport, Extend, ExtensionPoint, FinalState, Include, Permission, Relationship, TemplateParameter
Originated from 1.4 (Total 36)	AddVariableValueAction, ApplyFunctionAction, Artifact, CallOperationAction, Clause, ClearAssociationAction, ClearVariableAction, ControlFlow, CreateLinkAction, CreateLinkObjectAction, CreateObjectAction, DestroyLinkAction, DestroyObjectAction, InputPin, InvocationAction, LinkAction, LinkEndCreationData, LinkEndData, OutputPin, Pin, PrimitiveFunction, QualifierValue, ReadExtentAction, ReadIsClassifiedObjectAction, ReadLinkAction, ReadLinkObjectEndAction, ReadSelfAction, ReadVariableAction, ReclassifyObjectAction, RemoveVariableValueAction, SendSignalAction, TestIdentityAction, Variable, VariableAction, WriteLinkAction, WriteVariableAction

White Dwarf. The *White Dwarf* pattern identifies a category of meta-classes that there are more than one “1” and “0” must appear in the right side in their 6-bit evolutionary binary. The *White Dwarf* pattern means that the meta-classes are deleted sometime from the UML meta-model after they have survived in at least two versions of the UML. *ActionSequence*, *ActionState* and *ActivityModel* in Figure 1 conform to the pattern. There are five categories of meta-classes found which belong to the *White Dwarf* pattern, i.e. the patterns of “110000”, “111000”, “111110”, “001110” and “000110” separately, seeing Table 6.

Table 6. The White Dwarf meta-classes

From 1.1 to 1.2	ActivityModel, ActivityState, ElementReference, EventOriginKind, GraphicMarker, LocalInvocation, MessageInstance, OperationDirectionKind, Presentation, Refinement, Request, Structure, SynchronousKind, ViewElement
From 1.1 to 1.3	ActionSequence, Argument, CreateAction, DestroyAction, GeneralizableElement, MessageDirectionKind, ObjectSetExpression, ReturnAction, SendAction, TerminateAction, Time, Trace, Uninterpreted, UninterpretedAction
From 1.1 to 1.5	AssociationEndRole, AssociationRole, Attribute, AttributeLink, Binding, BooleanExpression, CallEvent, ChangeableKind, ChangeEvent, ClassifierInState, ClassifierRole, CompositeState, DataValue, ElementOwnership, Event, Exception, Geometry, Guard, Instance, Link, LinkEnd, LinkObject, Mapping, Method, ModelElement, Multiplicity, MultiplicityRange, Name, Object, ObjectFlowState, Partition, ProcedureExpression, ScopeKind, SignalEvent, SimpleState, StateVertex, Stereotype, SubmachineState, Subsystem, TaggedValue, TimeEvent, UseCaseInstance, ActionState
From 1.3 to 1.5	AssociationEnd, ActivityGraph, ArgListsExpression, CallState, ComponentInstance, ElementResidence, Flow, LocationReference, MappingExpression, NodeInstance, OrderingKind, PresentationElement, Stimulus, StubState, SubactivityState, SynchState, TypeExpression, UnlimitedInteger
From 1.4 to 1.5	AddAttributeValueAction, ArgumentSpecification, AsynchronousInvocationAction, AttributeAction, BroadcastSignalAction, ClearAttributeAction, CodeAction, CollaborationInstanceSet, CollectionAction, ConditionalAction, DataFlow, ExplicitInvocationAction, FilterAction, GeneralizableElement, GroupAction, HandlerAction, InteractionInstanceSet, IterateAction, JumpAction, JumpHandler, LiteralValueAction, LoopAction, MapAction, MarshalAction, NullAction, PrimitiveAction, Procedure, ProgrammingLanguageDataType, ReadAttributeAction, ReadLinkObjectQualifierAction, ReduceAction, RemoveAttributeValueAction, StartObjectStateMachineAction, SubsystemInstance, SynchronousInvocationAction, TagDefinition, TemplateArgument, UnmarshalAction, WriteAttributeAction

Table 7. The Island meta-classes in each version

UML 1.1	Action, ChangeEvent, Comment, CreateAction, Dependency, DestroyAction, Element, ElementOwnership, Enumeration, Event, Exception, Expression, Feature, Generalization, Geometry, GraphicMarker, Instance, Interaction, Interface, LocalInvocation, Mapping, Message, ObjectFlowState, SimpleState, Structure
UML 1.2	ActivityState, BooleanExpression, Comment, Element, ElementOwnership, Model, ObjectSetExpression, Primitive, ProcedureExpression, ReturnAction, SimpleState, Structure, Time, TimeExpression, Uninterpreted, Usage, UseCaseInstance, ViewElement
UML 1.3	ActionExpression, ArgListsExpression, BooleanExpression, Element, Geometry, IterationExpression, LocationReference, MappingExpression, Model, Multiplicity, MultiplicityRange, ObjectSetExpression, Permission, ProcedureExpression, Relationship, ReturnAction, SimpleState, Time, TimeExpression, TypeExpression, Uninterpreted, UninterpretedAction, Usage
UML 1.4	ArgListsExpression, BooleanExpression, ClearAttributeAction, ClearVariableAction, DestroyLinkAction, FilterAction, Geometry, LocationReference, MappingExpression, Model, Name, NullAction, Permission, Primitive, ProcedureExpression, Relationship, RemoveAttributeValueAction, RemoveVariableValueAction, SimpleState, TimeExpression, TypeExpression, Usage
UML 1.5	ArgListsExpression, BooleanExpression, ClearAttributeAction, ClearVariableAction, DestroyLinkAction, Geometry, LocationReference, MappingExpression, Name, NullAction, Permission, Primitive, ProcedureExpression, Relationship, RemoveAttributeValueAction, RemoveVariableValueAction, SimpleState, TimeExpression, TypeExpression, Usage
UML 2.0	ActivityFinalNode, AnyTrigger, CentralBufferNode, ClearStructuralFeatureAction, ClearVariableAction, ConditionalNode1, DecisionNode, DeployedArtifact, DestroyLinkAction, Device, DurationConstraint, ExecutionEnvironment, FlowFinalNode, InteractionOperator, IntervalConstraint, JoinNode1, MessageEnd, MessageTrigger, PrimitiveType, Realization, RemoveStructuralFeatureValueAction, RemoveVariableValueAction, Signal, StructuredActivityNode1, TimeConstraint, Type, Usage

4. Taxonomic Patterns based on the Meta-Class Fitness

We define a fit meta-class as one with enough definitions itself, i.e. embracing attributes, well-formed rules and associations to others. Thus, we use a fitness bit-map (seeing Figure 2), in which each 3-bit binary corresponds to a meta-class in one version of the UML. Each bit being set means the meta-class has one of three properties in corresponding UML version. For example, bit 0 is set if the meta-class in the version has well-formed rules, bit 1 is set if it has associations to others, bit 2 is set if it has attributes as well. A meta-class obtains at most six such binary corresponding to six UML versions so far. The *Abstraction* assigned by “100” in version of UML 1.3 means it possesses attributes but no associations to other meta-classes and no definition of well-formed rules.

UML Versions	1.1	1.2	1.3	1.4	1.5	2.0
Abstraction			100	100	100	010
Action	000	110	110	111	111	110
ActionExpression			000			
...						
Actor	110	001	001	001	011	001

Bit0: Set for WF rules
 Bit1: Set for Assoc.
 Bit2: Set for Attr.

Figure 2. A fitness bit-map for meta-classes

Although 3-bit binary can produce up to 8 patterns we just pay attention to an extreme one, i.e. the pattern of “000” and refer it as *Island* for the convenience of memory and communication. The category of meta-classes identified by the *Island* pattern is in Table 7, where meta-classes for basic data type, literal and enumeration are omitted.

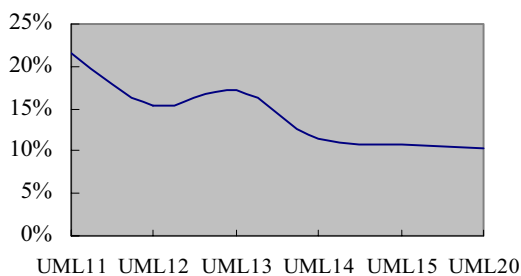


Figure 3. The ratio of the Island meta-classes in each the UML version

Figure 3 shows the percent of the *Island* meta-classes in each UML version. To compute the ratio, the number of the *Island* meta-classes is required to compare with the total number of meta-classes in the version. It shows that more than 20 percent of meta-classes in the UML 1.1 belong to the *Island* pattern. However, the UML 2.0 just owns ten percent of the meta-classes.

Figure 4 shows the percent of excluded *Island* meta-classes from one version to the next using the pair of adjacent version in the X axis. The same phenomena happens when the UML upgrades from 1.1 to 1.2 and from 1.4 to 1.5, that is, all the *Island* meta-classes in 1.1 are accepted by 1.2, so do 1.4 and 1.5. On the other hand, almost 70 percent of the *Island* meta-classes in the UML 1.5 are ruled out when the UML reaches the version of 2.0.

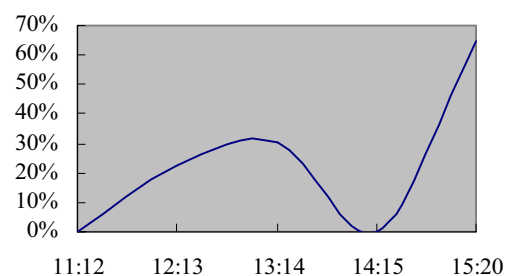


Figure 4. The ratio of the Island meta-classes eliminated from one version to the next

5. Evaluation of the UML based on the patterns

The classification for clustering meta-classes based on the patterns helps us finding evidence of the concerns on design quality of the UML. Furthermore, the patterns contribute to find problematic meta-classes, possible design defects and the inconsistent phenomenon between the UML meta-model and the domain that it specifies.

The *Fixed Star* pattern might classify a set of kernel model constructs that researchers are looking for [3]. Meta-classes in Table 2 are almost in accordance with the Meta Object Facility (MOF) [13] and the minimal kernel defined in Meta-modeling Language [3]. Therefore the set of the *Fixed Star* meta-classes can be the candidates for defining the UML itself. However, there are no meta-classes for instance specification in the category such as *Object* and *Link*, boiling down to the fact that constructs for specifying instances are always in the center of debate [2].

The *New Star* meta-classes together with ones in the *Fixed Star* pattern would aggregate a stable part of the UML meta-model. Most of total 87 meta-classes in two patterns can find semantic correspondences to the major constructs identified by the research of Opdahl and Henderson-Sellers [14].

The meta-classes in the *Comet* pattern may mean unmindful of augment to the UML and it results in construct redundancy or excess [21]. For example, the *ActionExpression* in the UML 1.3 is a redundant construct since it has no significant requirement that the *Expression* meta-class can not satisfy. On the other hand, the absence of the *Pulsar* meta-classes implies the UML is not power enough or convenient for modeling applications or meta-modeling itself. For instance, the *Enumeration* and *Primitive* play the role only as stereotypes serving other meta-classes in the UML 1.3, rather than as independent meta-classes as in the UML 1.1 and 1.4. This leads to the fact that users have no way to define new enumeration type and primitive type accordingly.

The *New Star* pattern not only introduces a category of meta-classes that represent the augment to the specification power of the UML, but also has proved the necessity of these meta-classes since they have been embraced by at least three versions, especially surviving successfully after a long process of examination and major modification to the UML 2.0. Such meta-classes therefore has vital force, such as the abstract *Relationship* for specifying connecting between elements, the *Extend* and *Include* for Use Case modeling, as well as a series of *Action-oriented* meta-classes for behavior specification.

The *White Dwarf* pattern reveals a category of meta-classes that might be design flaws. Based on the observation of the meta-classes, the reason behind the phenomenon can be concluded as following:

- Some constructs are introduced based on common sense and intuition, especially abstracted bottom-up from OO programming concepts that there is short of system-theoretical ontological foundation regardless of domains [4], such meta-classes as *Structure*, *Argument*, *Object* and *Link*, *StubState*, *Procedure* and so on.
- Some meta-classes might be "design-by-committee" compromises [7] that there is short of theoretical foundations and validation in the design of the UML meta-model. To enhance the UML tends to adopt a 'mud-packing' approach or makes direct amendments the UML. The obvious evidence of the phenomenon is that many *Action-oriented* meta-classes were introduced in the UML 1.4 but eliminated again in the UML 2.0.

There are some meta-classes in the *White Dwarf* pattern that means redundant or excess to the domain that the UML supports, or overlapped with other meta-classes, or renamed for understandability. To understand the matter human comprehension to the UML standards are needed. Some examples are as follows. The *Property* now substitutes the role that the *AssociationEnd* played; the *Flow* is refined into *ObjectFlow* and *ControlFlow*; the *Collaboration* with internal structure supersedes the *ClassifierRole*, *AssociationRole* and *AssociationEndRole*; the *Multiplicity* is renamed as *MultiplicityElement* and so on.

The *Island* pattern can be complementary to other patterns mentioned above since it predicts meta-classes that exhibit a potential for elimination (to enter the *Comet* or *White Dwarf* category) or refinement (to enter the *Fixed Star* or *New Star* category). Seeing Figure 2 and Table 7, the *ActionExpression* is the example of the former; the *Abstraction*, *Action* and *Actor* belongs to the latter.

The *Island* pattern can help to find out meta-classes that play the role as "predefined" constructs [1] or just as "placeholder" for defining concepts that would be used in modeling. The meta-classes possess nothing but a concept name, thus have no properties and relationship with others. For example, the *Element* meta-class acts as a "placeholder" for being root of the hierarchical tree in the early UML meta-models. According to [1], these "predefined" meta-classes, if will not be refined later, should be removed from the meta-model, instead locating them in the level of model and fulfilling their function using traditional OO inheritance technology.

Furthermore, observing the meta-classes in the *Island* pattern would reflect the quality changing of the UML versions. Figure 3 shows that the ratio of the *Island* meta-classes reaches the high point in the UML 1.1 and 1.3 separately, but decreases from 1.4 to 2.0. It implies the later versions of the UML have the higher quality than the formers. Figure 4 further prove the implication since it reveals that more *Island* meta-classes are removed when the UML upgrades from 1.5 to 2.0. Figure 4 also implies that the stability of the UML happens between from 1.1 to 1.2 and from 1.4 to 1.5 since no *Island* meta-classes are removed during the time. The implication is in accordance with the quality analysis in [10].

6. Related Works

Wand and Weber in [21] define that there are four kinds of inconsistency between the conceptual model

and the domains, which are construct overloading, construct deficit, construct excess and construct redundancy. The similar way to evaluate the UML is conducted in [11] and [14]. The papers try to find out the four kinds of meta-classes through leveraging the taxonomic patterns based on meta-classes lifetime.

Opdahl and Henderson-Sellers [14] introduce a category of meta-classes based on how well they match with the BWV-model [21]. The result of their comparisons between 47 ontological concepts in the BWV-model and 216 modelling constructs in the UML finds out 67 major constructs relevant for representing concrete problem domains. As mentioned in section 5, the *New Star* meta-classes together with ones in the *Fixed Star* pattern would be the candidates of these major constructs.

How to identify classes from legacy OO systems has been discussed for years. The most recent work is present in Ducasse and Lanza's [5]. They propose a visualization of classes called the class blueprint approach. It allows a software engineer to build a first mental model of a class that he validates. Although our work is conducted in the level of the meta-model the similar mental model of the meta-classes might be obtained using the taxonomic patterns.

Our previous work in [10] focuses on the assessment of the UML meta-models using OO metrics. The approach can identify and characterize stability and design quality of different meta-models. The current paper validates to some degree the previous result in [10]. Two works are complementary to each other.

7. Conclusion

The paper proposes taxonomic patterns for evaluating the UML based on the insights we obtained during observations on history of the UML meta-classes from the UML 1.1 to 2.0. The interesting vocabulary is accordingly attached to the patterns for identifying them conveniently.

Each pattern has its contribution to the concerns of design quality of the UML:

- They can help to locate and understand different meta-classes and their historical trace.
- They can help to find out meta-classes that are stable, important, promising, or exceptional, ill-defined.
- They can help to identify inconsistency between the UML meta-model and application domains that it specifies.
- They can help to validate the design quality of the UML.

- They provide valuable information for guiding the development and evolution of new versions of the UML and can be used to further develop heuristics for estimation activities.

Since in this paper we only concern the observation and classification of the UML meta-classes, it would be extend to examine arbitrary members of the family of the UML-based modeling languages.

Acknowledgements The work is funded by the National High-Tech Research and Development Plan of China (No.2004AA112070) and the National Grand Fundamental Research 973 Program of China (No.2002CB31200003), and the National Science Foundation of China No.60473064.

References

- [1] Atkinson, C. and Kühne, T. Strict Profiles: Why and How, In *3th International Conference of the UML (UML'00)*, LNCS 1939, 2000, 309-322.
- [2] Atkinson, C. and Kühne, T. Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation*. Vol. 12, No. 4, 2002:290– 321.
- [3] Clark, T. Evans, A. Kent S. Brodsky, B. and Cook, S. A Feasibility Study in Rearchitecting the UML as a Family of Languages using a Precise OO Meta-Modeling Approach, Version 1.0. September 2000. Available from www.puml.org
- [4] Dori, D. Why Significant the UML Change is Unlikely. *Communications of the ACM*, Vol. 45, No.11, 2002:82– 85.
- [5] Ducasse, S. and Lanza, M. The class blueprint: visually supporting the understanding of glasses. *IEEE Transactions on Software Engineering*. Volume 31, Issue 1, 2005:75– 90.
- [6] Duddy, K. the UML2 must enable a family of languages. *Communications of the ACM*, Vol. 45, No.11, 2002:73 – 75.
- [7] Kobryn, C. Will the UML 2.0 Be Agile or Awkward? *Communications of the ACM*, Vol. 45, No. 1, 2002:107– 110.
- [8] Lanza, M. The evolution matrix: recovering software evolution using software visualization techniques. *Proc. Int. Workshop on Principles of Software Evolution*, Vienna, Austria, September 2001.
- [9] Ma, Z. Jang, Y. Li, J. and Dai, Y. Research and Implementation of JBOO Based on the UML. *ACTA ELECTRONICA SINICA*, Vol.12A. 2002. (in Chinese)
- [10] Ma, H. Shao, W. Zhang, L. Applying OO Metrics to Assess the UML Meta-Models. In: T. Baar et al, eds. *Proc. of the 7th International Conference of the UML (UML'04)*. Springer-Verlag, LNCS 3273, 2004, 12-26.
- [11] Ma, H. Shao, W. and Zhang, L. A Quality Framework of the UML-based Modelling Languages. *Submitted to MoDELS'05*. 2005.

- [12] Minsky, M. A framework for representing knowledge. In: *The Psychology of Computer Vision*, ed. P. Winston, McGraw-Hill, New York. 1975.
- [13] Meta Object Facility (MOF) 2.0 Core Proposal, *OMG Document*: ad/03-04-07
- [14] Opdahl, A. and Henderson-Sellers, B. Ontological Evaluation of the UML Using the Bunge-Wand-Weber Model. *Software and Systems Modeling*, Volume 1, Number 1, 2002, Pages: 43 - 67.
- [15] Unified Modeling Language Semantics, Version 1.1, *OMG Document*: ad/97-08-04.
- [16] Unified Modeling Language Specification, Version1.2, *OMG Document*: March 1998.
- [17] Unified Modeling Language Specification, Version1.3, *OMG Document*: formal/00-03-01
- [18] Unified Modeling Language Specification, Version 1.4 with Action Semantics, *OMG Document*: formal/01-09-67
- [19] Unified Modeling Language Specification, Version 1.5, *OMG Document*: formal/03-03-01
- [20] The UML 2.0 Superstructure Specification, *OMG Document*: ptc/03-08-02.
- [21] Wand, Y. and Weber, R. On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. *Journal of Information Systems*, Vol. 3, 1993:217–237.