Using *Profiles* to Re-architect the UML

Tony Clark*
Andy Evans†
Stuart Kent‡

April 11, 2000

Draft

1 Introduction

Currently, UML version 1.3 is defined a collection of UML meta-models (a definition of UML in a subset of itself). Each meta-model describes the structure of part of the language and provides a collection of well-formedness constraints. The semantics of the language are given as informal text. The definition is unsatisfactory because it is partial, unstructured and introduces questions relating to the soundness of such a meta-circular language definition.

Under the auspices of the precise UML (pUML) group we have proposed a re-structuring and semantic definition of the current version of UML (1.3) [5]. This work aims to provide a modular definition of the semantics that can support a wide variety of profiles. There are number of components to this definition: a kernel library, which provides a collection of modelling concepts essential to the building of UML profiles, an extension mechanism for constructing profiles as extensions of the kernel library or other profiles, and a constraint language for expressing invariant properties of UML models. It is intended that once completed, the kernel and associated domain specific profiles will provide a standard reference library for the UML.

In order to re-architect the UML we intend to perform the following:

- Define a Meta-Modelling Sub-Language (MMSL) and use it to define UML. The MMSL is essentially a replacement for the MOF.
- Define a profile mechanism that allows UML to be constructed from semantically rich packages.
- Define a semantics for the MMSL and OCL.

*Department of Computing, University of Bradford
†Department of Computer Science, University of York, UK
‡Computing Laboratory, University of Kent, UK

One of the key features of this approach is the notion of a *profile*. The current UML 1.3 definition suffers from being syntax-bound and semantics-free. OCL is used to express syntactic well-formedness constraints on UML models. However, there is no way of determining what a model *means*. Profiles are a way of defining model components in terms of both syntax *and* semantics. This paper aims to describe the concept of profile by an example. The paper is structured as follows: section 2 describes key issues that must be addressed in order to re-architect UML 1.3; section 3 defines an architecture for re-structuring UML; section 4 defines the term *profile* as used in our approach; section 5 defines a profile that provides examples each of the key components of the approach; finally, section 6 discusses issues of the approach and describes current and future work.

2 Limitations of UML 1.3

2.1 Semantic Foundation

To define a language requires (at least) an abstract syntax, a semantics domain and a relationship between the two to be defined (see [2] for more details). In the current UML semantics document, the abstract syntax is defined using a meta-model approach (class diagrams + OCL constraints), the semantics domain is given in natural language as is the relationship between the syntax and semantics. Thus the semantics document is not a precise or formal description of the language. Such a description is required in order to facilitate: analysis; tool construction; modularity and composition; language extension; rigorous proof; and detailed comparison with other approaches.

Our approach uses the MMSL to define profiles. The MMSL has a meta-circular definition and an external semantics using an object calculus [8]. The object calculus is hidden; its existence is essential in order to resolve fundamental semantic issues.

2.2 Multiple Modelling Languages

The current version of UML provides a large number of modelling facilities. Because of this, there is a danger of becoming overloaded with too many concepts, many of which are not widely used except in very specific circumstances. For example, the definition of class diagrams (static model elements) supports a wide variety of facilities for expressing constraints. In practice, these facilities are rarely used, or may be used inappropriately.

In practice, it is very important to be able to construct different semantic definitions for specific modelling domains. Some examples that have already been proposed for UML are: real-time, business and networking domains, among others. This has led to the notion of a UML profile[3]: a semantics definition which is specifically aimed at supporting a single modelling domain.

In order to re-architect the UML we require extension mechanisms for constructing profiles from pre-existing ones, thus enabling profile reuse. For example, it should be possible to have a core or kernel profile (introducing common UML semantic concepts: classes, associations, operations, etc.), then for each profile to import from the core, adding further concepts and placing restrictions on the use of imported concepts.

2.3 Syntactic Limitations

The UML 1.3 definition assumes that all concrete syntax has been transformed into abstract syntax structures such as those manipulated internally by tools. Whilst this approach is acceptable if the mapping from concrete syntax to abstract syntax is one-to-one, this may not always be the case. In the proposed profiling approach we define a simple MMSL and treat many aspects of UML concrete syntax as *sugar*. This raises the issue of how to define a mapping between concrete and abstract syntax. A corollary is the issue of alternative syntaxes for UML. Two examples where alternative syntaxes are desirable: application domain specific patterns; and providing graphical notations for OCL.

2.4 Constraint Language

OCL is currently used to define well-formedness constraints in UML 1.3. Our approach views OCL as a fundamental component of the definition of UML. UML is constructed by composing profiles. Each profile is a collection of classes whose instances are constrained using OCL expressions. OCL is therefore a mechanism for expressing both syntactic properties (i.e. well-formedness) and semantic properties.

We make one simple but powerful extension to OCL. The current definition of OCL is weighted towards its use in defining invariants, operation pre- and post-conditions and guards on state transitions. To determine whether any given OCL expression is satisfied requires a context containing information such as the values of free variables. The context is *implicit* in the current definition of OCL. We extend this by allowing OCL expressions to have *explicit* contexts; essentially this is achieved by allowing parameterisation over OCL expressions, turning OCL expressions into functions from a number of values to `true` or `false`.

This extension allows users much greater freedom over where they place OCL expressions. It is essential to allow OCL to be used to its full potential in meta-models.

3 An Architecture for UML Semantics

This section briefly summarises recent work done to provide a semantics architecture for UML, which supports the precise definition of UML profiles. This work was presented as a response to the OMG's request for information (RFI)

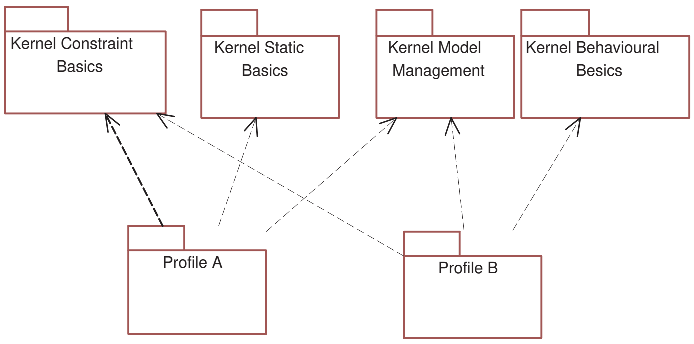

Figure 1: Profile architecture

regarding the next major release of UML (version 2.0) by the precise UML group [5].

The semantics architecture presented in [5] is based upon the use of meta-modelling to provide a precise denotational description of UML concepts. The definition is structured into packages, based on a kernel library of language definition tools and components. A profile is a definition of a language that may specialise and/or extend other profiles, and incorporate components from the kernel library. Figure 1, shows the general architecture.

The kernel library consists of a number of basic packages containing fundamental UML concepts. These include:

Static basics - generalised constructs for modelling the static properties of systems.

Constraint basics - constructs relating to the expression of constraints.

Dynamic basics - constructs for modelling the behaviour of systems.

Model management basics - general mechanisms for extending and specialising the components of the language.

As shown, profiles are extensions of these basic packages. An extension mechanism, similar to that proposed in the Catalysis method [6] is used to copy elements from one package into another, whilst also permitting extension of their features.

Each profile is organised into abstract syntax, semantics domain and a satisfaction/denotation relationship between the two (see Figure 2). Both abstract syntax and semantics domain may have many concrete representations.

The definition of UML using profiles is underway. Our aims are to contribute to the UML 2.0 revision process to produce a modelling notation that is manageable, extensible and semantics-rich. Profiles are the building blocks

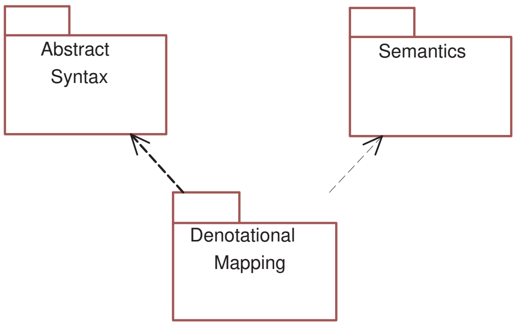

Figure 2: Profile semantics

of the approach. The rest of this paper gives the step by step construction of a simple profile. The example UML diagrams have been constructed using the Argo/UML tool [7].

4 Profiles

A *profile* defines a language in terms of its syntax and semantics. Both the syntax and semantics are described using the Meta-Modelling Sub-Language which is the fundamental core language for UML. The *syntax* of a language defines the components used to construct *phrases* in the domain of discourse. For example, if UML is the domain of discourse then phrases are composed from classes, objects and messages. If e-commerce is the domain of discourse then we may talk about web-pages, input-fields and data validation.

The syntax of a language provides a weak modelling notation. We can use OCL to describe well-formed-ness constraints on the syntax. These constraints are essentially meaningless and cannot be validated since each syntactic phrase exists in a semantic vacuum. Modelling is strengthened by associating each syntactic phrase with meaning. Meaning is the *semantics* of a language, usually consisting of a collection of objects whose properties are simple and well-understood.

A UML profile consists of three essential components: a model of the syntax; a model of the semantics; a relationship between well-formed syntax phrases and objects in the semantic domain. Each component is defined using the MMSL which consists of classes, associations, inheritance and OCL. A profile is therefore:

- A model of the syntax domain given as a class diagram and well-formedness constraints in OCL.
- A model of the semantic domain given as a class diagram and well-

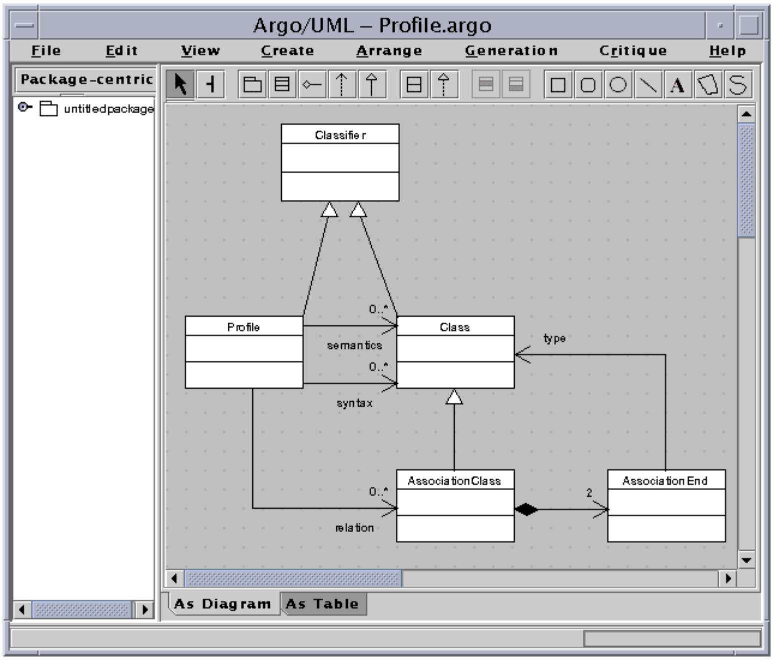

Figure 3: Meta Model for Profile

formedness constraints in OCL.

- A semantic mapping that defines relationships between components of the syntax and semantic models. OCL is used to define constraints on the relationships.

Figure 3 shows a definition of the class `Profile` as part of the MMSL. A `Classifier` is a description of a category of data values. All profiles are classifiers; they define a relationship between syntactic objects and semantic objects, or *models*. Objects classified by a profile are defined to be those objects classified by the semantic mapping of the profile.

`Profile` is defined as a class with super-class `Classifier`. The associations `syntax` and `semantics` define the classes that are used to represent the syntactic and semantic objects in the profile. A profile is associated via `relation` to a collection of binary association classes. Each class holds between a syntax class and a semantics class:

```
Profile
relation->forall(r |
   syntax.includes(r.associationEnd[1].type) and
   semantics.includes(r.associationEnd[2].type)
```

Invariants placed on the values of `relation` define the classification constraints for the profile. Typically there will be a single semantics class and a relation between it and each syntax class. If the relation is total and many to one then this constitutes a semantic function in the sense of denotational semantics.

Our approach re-architects the UML using profiles. Example profiles are:

- The static profile whose syntax is class diagrams and whose semantics is objects with slots. The semantic mapping ensures that objects have slots corresponding to the associations defined by the classes.
- The OCL profile whose syntax is OCL and whose semantics is the two element value domain containing `true` and `false`. The semantic mapping *evaluates* the OCL expressions producing the outcome `true` or `false`.
- The interaction profile whose syntax is interaction diagrams and whose semantics is traces of messages between objects. The semantic mapping will make particular choices of ordering and alternative interactions.

Since profiles encode the semantics of a modelling language they may be used to check the equivalence of two or more profile instances. This technique can be used to establish the rumoured equivalence of collaboration diagrams and sequence diagrams.

5 Example Profile

Profiles may be used to define UML or to define application specific languages and their semantics. This section constructs a simple example profile from the domain of web commerce. The essential features of the `ECommerce` profile are:

- The syntactic domain is a language for expressing web pages supporting text and user input.
- The semantic domain denotes executions of a machine that records user input and modifies a collection of customer accounts.
- The relationships between syntax and semantics link 'evaluations' of the web page in terms of user interactions with the corresponding changes in customer accounts.

5.1 Syntax Domain

Figure 4 defines the classes in the syntax domain of the `ECommerce` profile. Each class represents a web page. `WebPage` is an abstract super-class. `Text` contains the text that appears on a web page. `Seq` composes two web pages together; sophisticated displays are not modelled: web pages occur in sequence, users access `first` and then press a **next** button to proceed to `second`. `InputBox` is an area for user input. An input box is labelled and has two conditions (see below). `Cond` is a conditional component it has a guard and two components. If the guard is true (see below) then the page displayed is described by `conseq` otherwise `alt` is displayed.

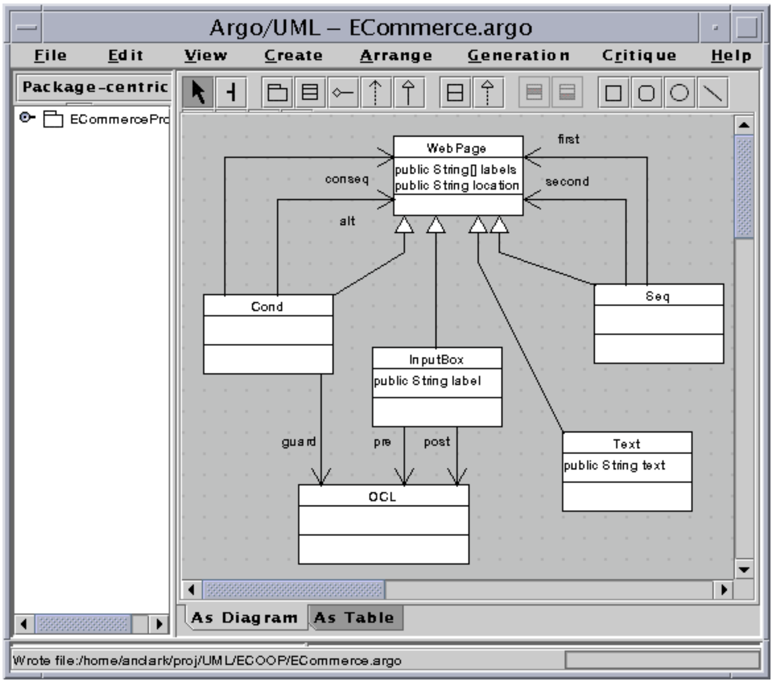

Figure 4: Syntax for ECommerce Profile

Conditions are used to control the consequences of user input in a conditional web page. In both cases, the conditions are expressed using *parameterised* OCL. OCL must be evaluated in a context. The context of OCL is implicit in the case where OCL is used to express class invariants, pre- and post-conditions and guards on state transitions. Parameterised OCL allows the context to be explicit, the values for the parameters are supplied by applying the OCL expression to a value.

The pre-condition of an input box is parameterised with respect to the pre-state and a value string. The pre-condition must be true with respect to the current (pre-) state of the system and the current input value in order for the web-page to allow the operator to proceed. The post-condition of an input box is parameterised with respect to the pre- and post-state of the system. The post-condition will usually specify a state change in terms of the pre-state values.

Syntax domains typically have *well-formedness* constraints. An example constraint for a web page is that all of the input box labels must be different. Well-formedness constraints are defined using OCL. Firstly, we constrain each of the syntactic domain classes to be associated with a set of labels:

```
WebPage
  labels = Set{}
```

```
InputBox
   labels = Set{label}
Seq
   labels = first.labels->union(second.labels)
Cond
   labels = conseq.labels->union(alt.labels)
```

The well-formedness constraint that all labels must be unique on a web page can be expressed as follows:

```
Seq
first.labels->intersection(second.labels) = Set{}
```

5.2 Semantic Domain

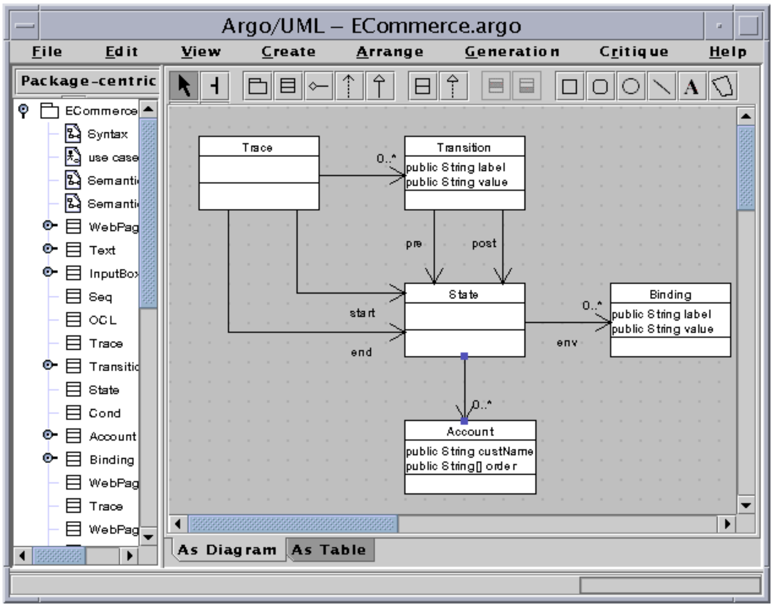

Figure 5: Semantics for ECommerce Profile

Figure 5 defines the classes in the semantic domain of the `ECommerce` profile. A `Trace` is a sequence of `Transitions`. Each transition has a pre-state and a post-state. A `State` is a collection of `Bindings` and information on customer `Accounts`.

Objects in the semantic domain are historical records of single user interactions with a web page that controls an interface to a collection of customer accounts. Each transition occurs in response to a user input. Each user input is recorded as a binding in the state environment. Typically a user input will cause the environment to be extended and possibly cause a change to the current state of the customer accounts.

A semantic domain has well-formedness constraints. These are expressed using OCL. For example, the start and end transitions of any trace must be part of the trace:

```
Trace
   transitions->exists(t | t.pre = start) and
   transitions->exists(t | t.post = end)
```

Any transition which is not at the start of a trace must have a unique preceding transition and any transition which is not at the end must have a unique succeeding transition:

```
Trace
   transitions->forall(t1 |
      not(t1.pre = start) implies transitions->exists(t2 |
         t1.pre = t2.post and transitions->forall(t3 |
            t1.pre = t3.post implies t2 = t3))) and
   transitions->forall(t1 |
      not(t1.post = end) implies transitions->exists(t2 |
         t1.post = t2.pre and transitions->forall(t3 |
            t1.post = t3.pre implies t2 = t3)))
```

All the names in the environment of a state must be unique:

```
State
   env->forall(b1 |
      env->forall(b2 |
         b1.name = b2.name implies b1 = b2))
```

Transitions can either leave the current environment alone or may extend it with a single binding:

```
Transition
   post.env = pre.env or
   ( post.env->setDifference(pre.env)->size = 1 and
     post.env->setDifference(pre.env).name = label and
     post.env->setDifference(pre.env).value = value)
```

5.3 Semantic Mapping

Figure 6 shows the semantic mapping from elements of the ECommerce syntax domain to elements of the ECommerce semantics domain. The mapping consists of 5 association classes. Each association class represents a relationship between a different syntax class and instances of the semantic domain class `Trace`. The constraints on each association class define a mappings between instances of the syntax classes and instances of the semantics classes. Each of these constraints is described in turn.

When a user observes text on a web page there is no change in the underlying system state. This is recorded in the semantics as being no observable system transition:

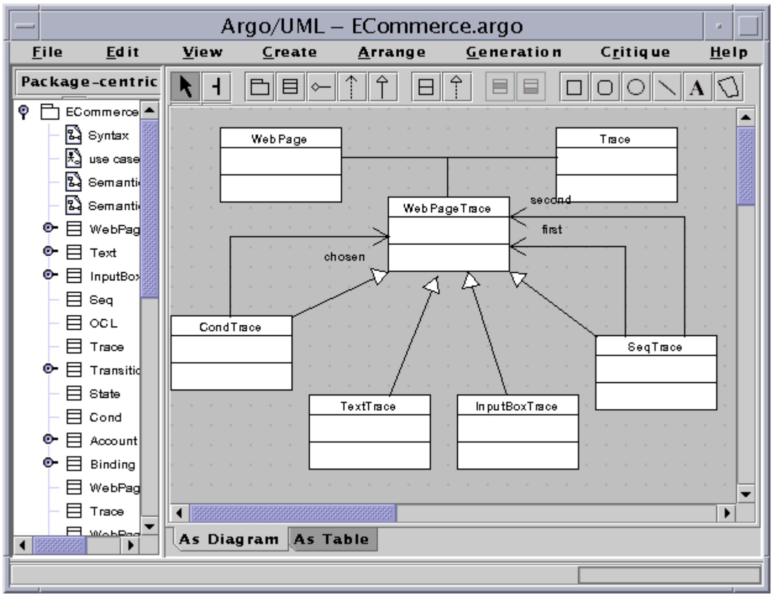

Figure 6: Semantic Mapping for ECommerce Profile

```
TextTrace
   trace.transitions->size = 0 and
   trace.start = trace.end
```

A conditional web page displays one of two alternatives depending on the outcome of a test. A conditional trace therefore has an extra sub-trace recording whether the consequent or alternative web page was displayed:

```
CondTrace
   webPage.guard(trace.start) implies
      chosen.webPage = webPage.conseq and
      trace = chosen.trace and
   not(webPage.guard(trace.start)) implies
      chosen.webPage = webPage.alt and
      trace = chosen.trace
```

The invariant on `CondTrace` provides an example of a parameterised OCL expression. The expression `webPage.guard` is parameterised with respect to a state. An example guard that checks for no accounts is:

```
fun(s:State) = s.accounts->size = 0
```

A sequence of web pages is defined to be the concatenation of two sub-traces recorded as the `first` and `second` attributes of an instance of `SeqTrace`:

```
SeqTrace
```

```
  first.webPage = webPage.first and
  second.webPage = second.first and
  first.trace.last = second.trace.first and
  trace.start = first.trace.start and
  trace.end = second.trace.end and
  trace.transitions = first.trace.transitions->
    union(second.trace.transitions)
```

An input box is the only web page component that performs a state transition. A state change is performed only when the pre-condition of the input box is true. If the pre-condition is false then the transition occurs but performs no state change. Otherwise the state change must satisfy the post-condition of the input box and the transition records the input data by adding it to the state environment.

```
InputBoxTrace
  let binding = trace.end.env->setDifference(trace.start.env)
  in trace.transitions->size = 1 and
    trace.transitions.label = webPage.label and
    webPage.pre(trace.start,trace.transitions.value) implies
      (binding.name = webPage.label and
       binding.value = trace.transitions.value and
       webPage.post(trace.start,trace.end)) and
    not(webPage.pre(trace.start,trace.transitions.value)) implies
      trace.start = trace.end
```

5.4 Example Model

Consider a web page that is a single input box `b` labelled with `name`. The idea is that the operator can type in their name and this will create a new account for them if one does not already exist. The input will always succeed so the precondition on the input box is `true`:

```
b.pre = fun(s:State,n:String) = true
```

The post-condition will only be true for transitions where the account already existed or has been created:

```
b.post = fun(pre:State,post:State) =
  pre.accounts->exists(a |
    a.name = lookup(post.env,"name")) implies
      pre.accounts = post.accounts and
  not pre.accounts->exists(a |
    a.name = lookup(post.env,"name")) implies
      let newAccount = post.accounts->setDifference(pre.accounts)
      in newAccount->size = 1 and
         newAcount.name = lookup(post.env,"name") and
         newAccount.orders = Set{}
```

6 Analysis and Current Work

This paper has motivated a requirement for re-architecting the current definition of UML. It has proposed an approach to this task based on profiles. A profile is a language definition given in terms of a syntax domain, a semantics domain and a semantic mapping. Using profiles to define UML leads to a family of modelling languages each constructed by composing library profiles and introducing new application specific profiles. The paper has given a simple example of a simple e-commerce profile.

Re-architecting the UML is a very large task. We believe it is needed because the current definition (1.3) is not manageable and does not precisely address the issue of semantics. We intend to show that our approach is viable by defining the meta-circular MMSL profile and using it to define profiles for representative sub-components of UML (such as class diagrams).

Our longer-term aim is to provide a reference implementation for the UML. One manifestation of this would be a tool that could be used to check syntax and semantic components that are used by other tools. The common interchange format would be and XMI variant that is based on the MMSL. A semantic architecture, such as that described in this paper, is a precursor to such a tool.

References

[1] OMG Unified Modeling Language Specification (1.3), Available from `http://www.omg.org`, 1999.

[2] Evans, A. S. & Kent, S: Meta-modelling semantics of UML: the pUML approach. 2nd International Conference on the Unified Modeling Language. Editors: Rumpe, B. & France, R. Colorado, LNCS, 1723, 1999.

[3] Cook S., Kleppe A. Mitchell R., Rumpe B., Warmer J. & Wills A.: Defining UML Family Members Using Prefaces. In Technology of Object-Oriented Languages and Systems, TOOL '99 PacificEditors: Mingins Ch. & Meyer B. IEEE Computer Society.

[4] Warmer J. & Kleppe A.: The Object Constraint Language: Precise Modelling with UML. Addison-Wesley, 1998.

[5] Clark A., Evans A., France R., Kent S. & Rumpe B.: Response to UML 2.0 Request for Information. Available at `http://www.cs.york.ac.uk`. 1999.

[6] D'Souza D. F. & Wills A.: Objects, Components and Frameworks with UML. Addison-Wesley, 1999.

[7] The ArgoUML Case Tool available at `http://argouml.tigris.org/`.

[8] Clark A., Evans A. & Kent S.: The Specification of A Reference Implementation for the Unified Modelling Language. Submitted to the L'Objet Journal, February 2000.