FAMIX Ada language plug-in 2.2

Author	Robb Nebbe (nebbe@iam.unibe.ch)
Version	2.2
Last modified	1999-08-17

1 Abstract

This document defines a language plug-in for FAMIX, the FAMOOS information exchange model [Deme99]. It extends and interprets the FAMIX core model to cover the essential entities from the Ada programming language.

2 Notation

The common exchange model is modified in three different ways to handle Ada:

- Attributes are generalised and moved higher into the classes comprising the FAMIX model. In particular the class entity is extended with the method *enclosingEntity* and *isPrivate* in order to better handle nesting. This replaces several attributes found lower in the class hierarchy which are retained (to not break tools that work at the language-independent level), but subsequently interpreted to correspond to the more general attributes.
- New attributes are added to existing classes of the common exchange model. In this case the class is marked "modified" and only the new and modified (see below) attributes are listed in the definition of the modified class.
- The definition of attributes of existing classes are modified or their syntax and semantics are interpreted with respect to Ada. To discriminate modified from new attributes, modified attributes are listed without any type information since that information isn't modified anyway.

3 Modified classes

3.1 Model (interpreted)

Model
SourceLanguage SourceDialect

Figure 1: Model

- `SourceLanguage`
  For all Ada models the attribute always contains the string "Ada"
- `SourceDialect`
  This attribute should correspond to the standard defining the version of the language. In the case of the most recent version of Ada this should be "ANSI/ISO/IEC-8652:1995"

3.2 Entity (interpreted and extended)

Entity	
name	
uniqueName	
enclosingEntity (): Name	# new
isPrivate ():Boolean	# new

Figure 2: Entity

The name of an entity corresponds to its simple name in Ada. The unique name corresponds to the fully qualified name in Ada but may also require extra information such as the signature in the case of subprograms[1].

The method *enclosingEntity* returns the unique name of the entity in which another is declared unless the entity is a library level compilation unit in which case it returns an empty string "". The method isPrivate tells whether the entity in question is made public or private with respect to the enclosing entity.

3.3 Package (interpreted)

Package
belongsToPackage

Figure 3: Package

- `belongsToPackage():Name;`
  When a package is nested within another package or in the case of a child package this method returns the name of the enclosing package or the parent package. If a package is nested inside another entity such as a subprogram then the method will return the empty string. If both belongsToPackage and enclosingEntity are defined then they will return the same entity.

3.4 Class (interpreted)

Class
isAbstract belongsToPackage

Figure 2: Class

Ada types are all mapped to the class entity in FAMIX with the exception of access types and subtypes. Neither access types or subtypes introduce a new abstraction in the problem domain. Furthermore, pointers are conventionally implicit in traditional object-oriented languages and metrics would be skewed if access types were not eliminated in Ada.

This means that enumeration, signed integer, modular, floating point, fixed point, decimal fixed point, unconstrained and constrained array, record and tagged record types as well as task and protected types are all mapped to the class entity.

- `isAbstract():Boolean`
  True if the class corresponds to an abstract tagged record type.

[1] Note that that "subprogram" is the generic term for procedures and functions (and thus for methods in FAMIX) in Ada terminology.

- `belongsToPackage():Name;`
  When a class is nested within a package this method returns the name of the enclosing package. If a class is nested inside another entity than a package (such as a subprogram) then the method will return the empty string. If both belongsToPackage and enclosingEntity are defined then they will return the same entity.

3.5 Method (interpreted)

Method
belongsToClass
isAbstract
signature
declaredReturnClass

Figure 3: Method

Methods correspond to Ada's primitive subprograms and class-wide subprograms. This is because both primitive and class-wide subprograms are already defined for or will be derived for any derived types. Task entries as well as the entries, functions and procedures defined for protected types are also considered as methods.

- `belongsToClass():Name`
  Returns the class for which the subprogram is either a primitive method or a class-wide subprogram. Note that in Ada subprograms are not nested in a type so `enclosingEntity` is never the same as `belongsToClass`.
- `isAbstract():Boolean`
  True if the corresponding subprogram is abstract.
- `signature():Qualifier`
  The signature consists of the name of the method and the names and types of its parameters. If there, resulttypes are concatenated at the end of the signature as well. Contrary to other object-oriented languagues parameter names and resulttypes are part of the unique identification of a method in Ada
  The string should be formed by "methodname(parname:partype, …)[:returntype]" where parname is the formal parameter name and partype is the unique name of the class to which the type corresponds. Returntype only appears when available.
  Example:
  If we take the following Ada code:

```
package P is
  type T is tagged private;
  function Method_A( X:Integer; Y:Boolean ) return T;
  procedure Method_B( A_T: in out T; Y: in Boolean );
  procedure Method_B( A_T: in out T; X: in Integer );

private ...
end P;
```

for Method_A the signature is:

```
"Method_A(X:Integer,Y:Boolean):T"
```

for the two methods Method_B the signatures are:

```
"Method_B(A_T:T,Y:Boolean)"
"Method_B(A_T:T,X:Integer)"
```

- `declaredReturnClass():Name`
  The result of a method corresponding to a function in Ada is obtained through `declaredReturnClass`, which returns the class corresponding to the Ada type returned by the function.

3.6 Function (interpreted)

Function
signature
declaredReturnClass

Figure 4: Function

A FAMIX function corresponds to any subprogram that does not qualify as a method. In Ada terminology the choice of function is rather unfortunate but it is kept for compatibility. It is essentially the same as a method except it does not have the method belongsToClass or the method isAbstract.

3.7 Attribute (extended and interpreted)

Attribute	
belongsToClass	
signature():Qualifier	# new
declaredReturnClass():Name	# new

Figure 5: Attribute

A FAMIX attribute corresponds to an Ada record component. It is similar to a method except that an attribute may not be abstract. In Ada a private type declaration leads to a situation where the type is not private but the attributes (record components) are private. An extension is that attributes are considered as having a signature consisting of a single parameter whose class is the class to which the attribute belongs. Since record components are always nested in a type the enclosing entity will always be the same as the result of `belongsToClass`.

3.8 GlobalVariable (interpreted)

GlobalVariable
declaredClass

Figure 6: GlobalVariable

A global variable corresponds to any variable or constant declared in a package thus having a lifetime corresponding to that of the program.

- `declaredClass():Name`
  The type of the variable or in the case of an access type the type accessed.

3.9 LocalVariable (interpreted)

LocalVariable
declaredClass

Figure 7: LocalVariable

A global variable corresponds to any variable or constant declared local to a subprogram thus having a lifetime corresponding to an invocation of the subprogram.

- `declaredClass():Name`
  The type of the variable or in the case of an access type the type accessed.

3.10 FormalParameter (interpreted)

FormalParameter
declaredClass
position

Figure 8: FormalParameter

A formal parameter means the same thing in both FAMIX and Ada. The parameter modes as well as access parameters are not carried over into the FAMIX model. A formal parameter is never private as it corresponds to an association formed between a visible entity and those defined by the parameter. If the formal parameter is an access parameter, an access type or a subtype then *declaredClass* returns the type accessed or the base type of the subtype.

- `declaredClass`
  The type of the formal parameter. Access parameters are contrued as being of the type accessed.

3.11 InheritanceDefinition (interpreted)

InheritanceDefinition
subclass
superclass

Figure 9: InheritanceDefinition

An inheritance definition corresponds to a type derivation in Ada. The subclass corresponds to the derived type and the superclass to the base type. All type derivations are considered as an inheritance definition even if the types involved are not tagged types.

- `subclass():Name`
  Returns the uniqueName of the derived type.
- `superclass():Name`
  Returns the uniqueName of the base type.

3.12 Access (interpreted)

Access
accesses
accessedIn

Figure 10: Access

An access corresponds to any time that an attribute (record component) is used in an expression or statement. Accesses is the attribute being accessed and accessedIn is the entity from which the access is made.

- `accesses():Name`
  The uniqueName of the attribute.
- `accessedIn():Name`
  The uniqueName of the behavioural entity from which that attribute is accessed.

3.13 Invocation (interpreted)

Invocation
invokes invokedBy

Figure 11: Invocation

An invocation corresponds to a subprogram call in Ada. The method invokes is the name of the subprogram being called but not the unique name because this may be impossible due to polymorphism. The method invokedBy is the unique name of the behavioural entity from which the subprogram is invoked.

- `invokes():Name`
  The uniqueName of the subprogram that is being called. In the case of polymorphism this subprogram may have been overridden by a derived type so another implementation may actually be executed.
- `invokedBy():Name`
  The uniqueName of the subprogram that calls the other subprogram.

4 Miscellaneous

Generic entities are not covered by this plug-in. Their instantiations are however, and they are treated as if they were written from scratch rather than instantiated from a generic. Implicit declarations occurring in the case of type derivation should be handled the same as explicit declarations.

5 References

[Deme99] Serge Demeyer, Sander Tichelaar and Patrick Steyaert, FAMIX – The FAMOOS Information Exchange Model, version 2.0 alpha, July 1999. See http://www.iam.unibe.ch/~famoos/FAMIX/.

Cover Pages

Achievement 2.4.1c

FAMIX Ada language plug-in 2.2

1) Identification

Project Id:	Esprit IV #21975 “FAMOOS”
Deliverable Id:	D 2.2 – FINALFHB Final FAMOOS Methodology Handbook
Date for delivery:	31.08.99
Planned date for delivery:	31.08.99
WP(s) contributing to:	1
Author(s):	Robb Nebbe

2) Abstract

This document defines a language plug-in for FAMIX, the FAMOOS information exchange model [Deme99]. It extends and interprets the FAMIX core model to cover the essential entities from the Ada programming language

3) Keywords

Object-oriented, reengineering, reverse engineering, code repository, round-trip engineering, FAMOOS, FAMIX, Ada.

4) Version History

Ver	Date	Editor(s)	Status & Notes
2.2 alpha	24.08.99	Robb Nebbe	
2.2	25.08.99	Sander Tichelaar	

5) Issues for future releases

6) Table of Contents

FAMIX Ada language plug-in 2.2 .... **1**

1 Abstract **1**

2 Notation **1**

3 Modified classes **1**

3.1 Model (interpreted) 1
3.2 Entity (interpreted and extended) 2
3.3 Package (interpreted) 2
3.4 Class (interpreted) 2
3.5 Method (interpreted) 3
3.6 Function (interpreted) 4
3.7 Attribute (extended and interpreted) 4
3.8 GlobalVariable (interpreted) 4

3.9 LocalVariable (interpreted) 5
3.10 FormalParameter (interpreted) 5
3.11 InheritanceDefinition (interpreted) 5
3.12 Access (interpreted) 5
3.13 Invocation (interpreted) 6

4 Miscellaneous 6

5 References 6

Cover Pages **7**

1) Identification 7
2) Abstract 7
3) Keywords 7
4) Version History 7
5) Issues for future releases 7
6) Table of Contents 7
7) List of Figures 8
8) List of Tables 8

7) List of Figures

Figure 1: Model 1
Figure 2: Entity 2
Figure 3: Package 2
Figure 2: Class 2
Figure 3: Method 3
Figure 4: Function 4
Figure 5: Attribute 4
Figure 6: GlobalVariable 4
Figure 7: LocalVariable 5
Figure 8: FormalParameter 5
Figure 9: InheritanceDefinition 5
Figure 10: Access 5
Figure 11: Invocation 6

8) List of Tables