UML CORBAfacility Interface Definition 5

This chapter specifies the interfaces for a CORBAfacility for the Unified Modeling Language, version 1.3. The UML CORBAfacility (or UML Facility for short) is a repository for models expressed in the UML. The facility enables the creation, storage, and manipulation of UML models. The facility enables clients to be developed that provide a wide variety of model-based development capabilities, including:

- Drawing and animation of UML models in UML and other notations
- Enforcement of process and method style guidelines
- Metrics, queries, and reports
- Automation of certain development lifecycle activities (e.g., through design wizards and code generation).

Contents

5 UML CORBAfacility Interface Definition

5.1 Service Description

There are two sets of interfaces provided: 1) generic and 2) tailored. Both sets of interfaces enable the creation and traversal of UML model elements. The generic interfaces are included in the Reflective module.

This is a set of general-purpose interfaces that provide utility for browser type functionality and as a base for the tailored interfaces. They are more fully described in the Meta-Object Facility (MOF) specification.

A set of tailored interfaces that are specifically typed to the UML metamodel elements is defined. The tailored interfaces inherit from the generic interfaces. The tailored interfaces provide capabilities necessary to instantiate, traverse, and modify UML model elements in the facility, directly in terms of the UML metamodel, with type safety. The specifications of the tailored interfaces were generated by applying a set of transformations to the UML semantic metamodel. Because the tailored interfaces were generated consistently from a set of patterns (described more fully in the MOF specification), they are easy to understand and program against. It is feasible to generate automatically the implementation for the UML Facility, for the most part, because of these patterns and because the UML metamodel is strictly structural.

The UML is designed with a layered architecture. Implementors can choose which layers to implement, and whether to implement only the generic interfaces or the generic and tailored interfaces.

One of the primary goals was to advance the state of the industry by enabling OO modeling tool interoperability. This UML Facility defines a set of interfaces to provide that tool interoperability. However, enabling meaningful exchange of model information between tools requires agreement on semantics and their visualization. The metamodel documenting the UML semantics and notation is defined in the UML Semantics chapter. Most of the IDL defined in this document is a direct mapping of the UML v 1.3 metamodel, based on the IDL mapping defined in the MOF specification. Because the UML semantics are sufficiently complex, they are documented separately in the UML Semantics chapter, whereas this chapter is void of explanations of semantics.

5.1.1 Tool Sharing Options

A major goal is to achieve semantic interoperability between UML tools. Figure 5-1 depicts several viable alternatives to exchanging model information between tools.

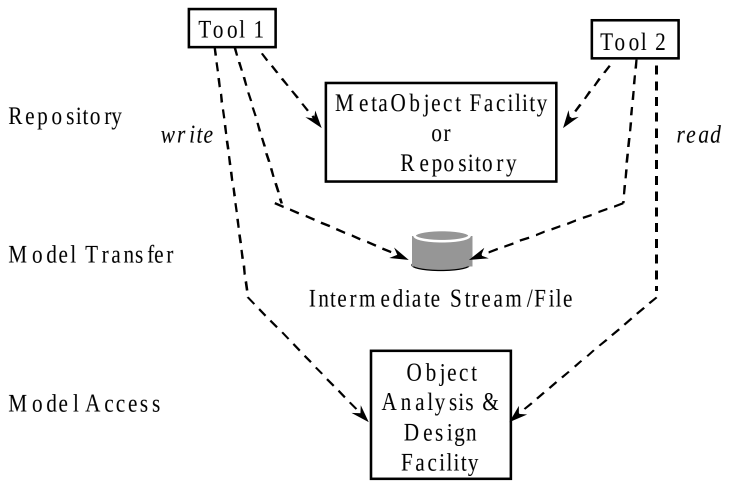

Figure 5-1 Model Sharing Alternatives between UML Tools

General-purpose Repository

Two tools could interface to the same repository and access a model there. The MetaObject Facility (MOF) could be this repository. This mapping is very implementation dependent, since the MOF cannot necessarily enforce the richer semantics defined in a UML-compliant tool. This approach is not described in this response, although the mapping to the MOF is described in the Preface.

Model Transfer

Two tools could understand the same stream format and exchange models via that stream, which could be a file. This is referred to as an “import facility.” Having this interface would be necessary to provide a path for tools that are not implemented in an API (CORBA or non-CORBA) or repository environment. The Preface discusses stream format and CDIF further.

Model Access

Two tools could exchange models on a detail-by-detail basis. This is referred to as a “connection facility.” Although this would not be the most efficient method for sharing an entire model, this type of access enables semantic interoperability to the greatest degree and is extremely useful for client applications. This is also a repository, but its interfaces are specific to UML tools. A set of IDL interfaces is defined in this document to provide model access.

In summary, the UML Facility defines IDL interfaces for clients to use in a Model Access mode. The interface is consistent with the UML metamodel contained in this response.

5.2 Mapping of UML Semantics to Facility Interfaces

Understanding the process used to generate the IDL for this facility is helpful in understanding the resulting IDL. The process was as follows:

1. Converted the UML Semantics Metamodel into the Interface Metamodel, making necessary refinements for CORBA interfaces.
2. Stored the Interface Metamodel into a MetaObject Facility prototype as an instance of the MOF meta-metamodel elements.
3. Generated IDL from the MOF, based on the mapping defined in the MOF proposal.

5.2.1 Transformation of UML Semantics Metamodel into Interfaces Metamodel

A model was created representing the interfaces required on the UML Facility. This interface metamodel is nearly identical to the UML Semantics metamodel, so it is not documented explicitly. The following list summarizes the conversions made from the UML Semantics metamodel:

- Named associations and their ends, where names were missing.
- Deleted derived associations, since they would have resulted in redundant interfaces.
- Mapped all UML data types and select classes to CORBA data types.
- Transformed association classes into more fundamental structures. (A goal of the MOF was simplicity, so it does not support association classes.)
- Combined the UML DataTypes and Extension Mechanisms packages into Core, resulting in easier-to-use name scoping for the interfaces.
- Renamed certain classifiers, association ends, and attributes to avoid conflicts with words reserved in Reflective interfaces, CORBA, and MOF.
- Set navigability for associations to be uni-directional between classifiers that crossed packages. This was necessary to permit implementations of the more fundamental packages/modules without requiring a full UML implementation[1] [2].

1. All other navigability is assumed to be useful. For example, although an implementation environment class should not know about its child classes, it is useful for an UML tool.
2. The UML Facility interfaces exclude navigation from classes in base packages/modules to classes in dependent packages/modules. This is deliberate. For example, an instance of a UML class does not know about a State Machine that might be attached to it. Vendors should consider adding interfaces to support such navigation when implementing multiple modules.

- Renamed AssociationClass to UmlAssociationClass. (The MOF IDL generation creates a FooClass for every Foo, so the UML class 'Association' would have created an 'AssociationClass' interface which would have clashed.)
- Renamed enumeration literal names so they would be unique within the resulting IDL modules.

The IDL generation from the MOF assures that all root classes in the interface metamodel are specializations of Reflective::RefObject, so this relationship is assumed to be present in the interface metamodel.

The remainder of this chapter describes the transformation of Association Classes as in the UML Semantics metamodel to the interface metamodel, summarizes the usage of CORBA data types, and summarizes the source and purpose of the MOF Reflective interfaces.

Transformation for Association Classes

Since the MOF does not represent the semantics of association classes directly, we needed to convert the association classes in the UML into a simple class and add necessary relationships to enable complete navigation (in the resulting facility IDL). Figure 5-2 shows an example association class as it would appear in the semantic metamodel.

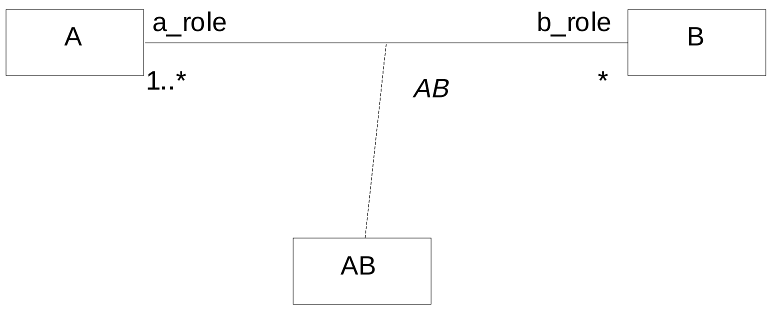

Figure 5-2 An Association Class in a Semantic Metamodel

Figure 5-3 shows the corresponding transformed structure in the interface model.

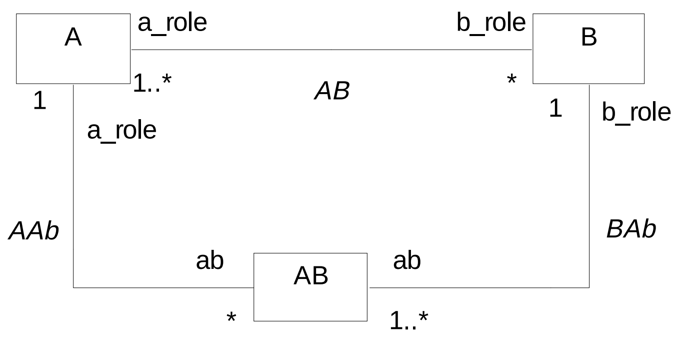

Figure 5-3 Corresponding Association Class in an Interface Metamodel

MOF Generic Interfaces

The MOF specification fully describes the generic interfaces. As a summary, the generic interfaces in the Reflective module provide the following:

- consistent treatment of type information,
- exception handling (including constraint violations, missing parameters, etc.), and
- generic creation and traversal of objects.

Note – The MOF specification replaces the definition of the Reflective module contained in this specification.

DataTypes for Interface

UML itself is platform independent; therefore, during the translation to the interface model, specific CORBA data types were selected as the structural base. These are listed in Table 5-1.

Table 5-1 Data Types

UML DataType	**IDL Declaration**
String	//string
Integer	typedef short Integer;
Uninterpreted	typedef any Uninterpreted;
Time	typedef float Time;
Name	struct Name { string body ; };
GraphicMarker	struct GraphicMarker { any body ; } ;
Geometry	struct Geometry { any body ; } ;
TimeExpression	struct TimeExpression { Name language ; any body ; } ;
ObjectSetExpression	struct ObjectSetExpression { Name language ; any body ; } ;

Table 5-1 Data Types

ProcedureExpression	struct ProcedureExpression { Name language ; any body ; } ;
Expression	struct Expression { Name language ; any body ; } ;
BooleanExpression	struct BooleanExpression { Name language ; any body ; } ;
Mapping	struct Mapping { any body ; } ;
MultiplicityRange	struct MultiplicityRange { lower short; upper short ; } ;
Multiplicity	sequence < MultiplicityRange > Multiplicity;
ChaneableKind	enum ChangeableKind { ck_none, ck_frozen, ck_addOnly };
OperationDirectionKind	enum OperationDirectionKind { odk_provide, odk_require };
ParameterDirectionKind	enum ParameterDirectionKind {pdk_in, pdk_inout, pdk_out, pdk_return};
MessageDirectionKind	enum MessageDirectionKind { mdk_activation, mdk_return };
SynchronousKind	enum SynchronousKind { sk_synchronous, sk_asynchronous};
ScopeKind	enum ScopeKind {sk_instance, sk_type};
VisibilityKind	enum VisibilityKind { vk_publc, vk_protected, vk_private };
PseudostateKind	enum PseudostateKind { pk_initial, pk_final, pk_shallowHistory, pk_deepHistory, pk_join, pk_fork, pk_branch, pk_or };
CallConcurrencyKind	enum CallConcurrencyKind { cck_sequential, cck_guarded, cck_concurrent } ;
AggregationKind	enum AggregationKind {ak_none, ak_shared, ak_composite};

5.2.2 Mapping of Interface Model into MOF

The UML metamodel elements can be expressed as instances of MOF meta-metamodel elements. This mapping is summarized in the table below for the relevant elements in the interface metamodel.

Table 5-2 Relevant Elements in the Interface Metamodel

Package	**Package, Contains**
Class	Class, Contains
Attribute	Attribute, Contains, IsOfType
DataType	DataType
Association	Association, AssociationEnd, Reference, Contains, RefersTo, IsOfType
Generalization	Generalizes

These mappings are relatively straightforward, with the exception of how an Association is instantiated. Figure 5-4 on page 5-9 shows an example association as it would appear in the interface model. Figure 5-5 on page 5-9 illustrates a relevant view of the MOF meta-metamodel.

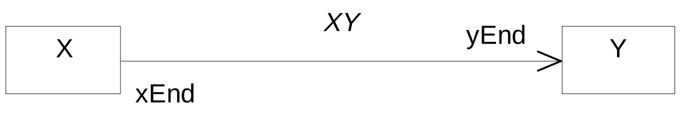

Figure 5-4 Association at Meta-Model Level Example

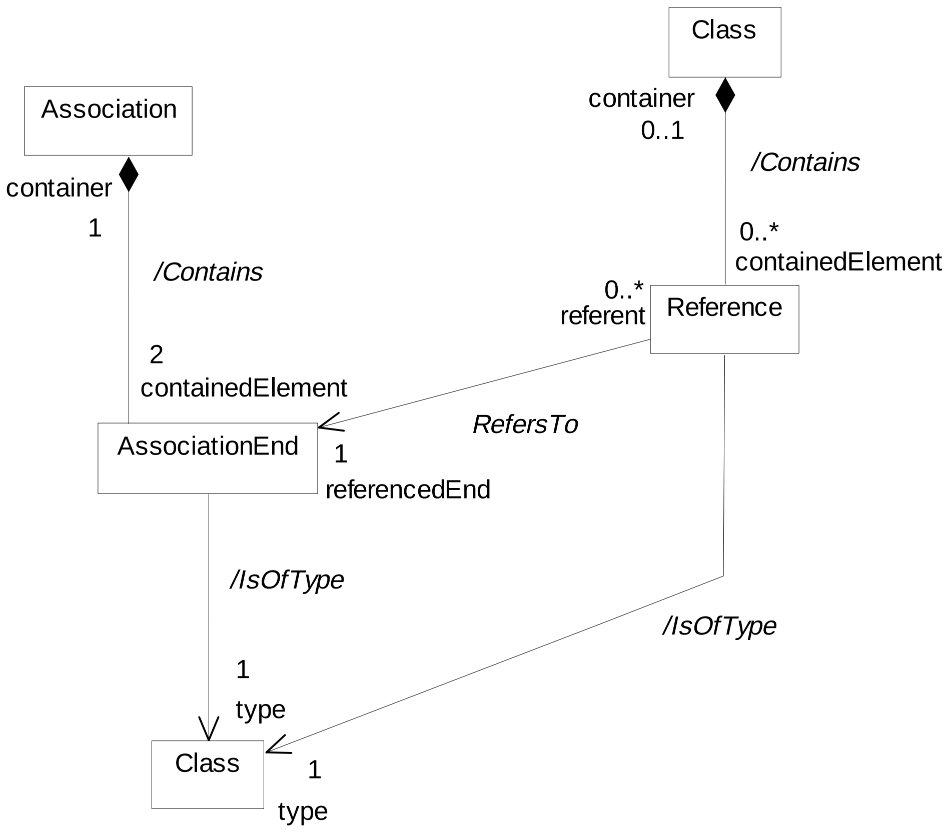

Figure 5-5 Projection of MOF Meta-Metamodel

Figure 5-6 on page 5-10 is a collaboration diagram showing how the association would be instantiated in terms of the MOF.

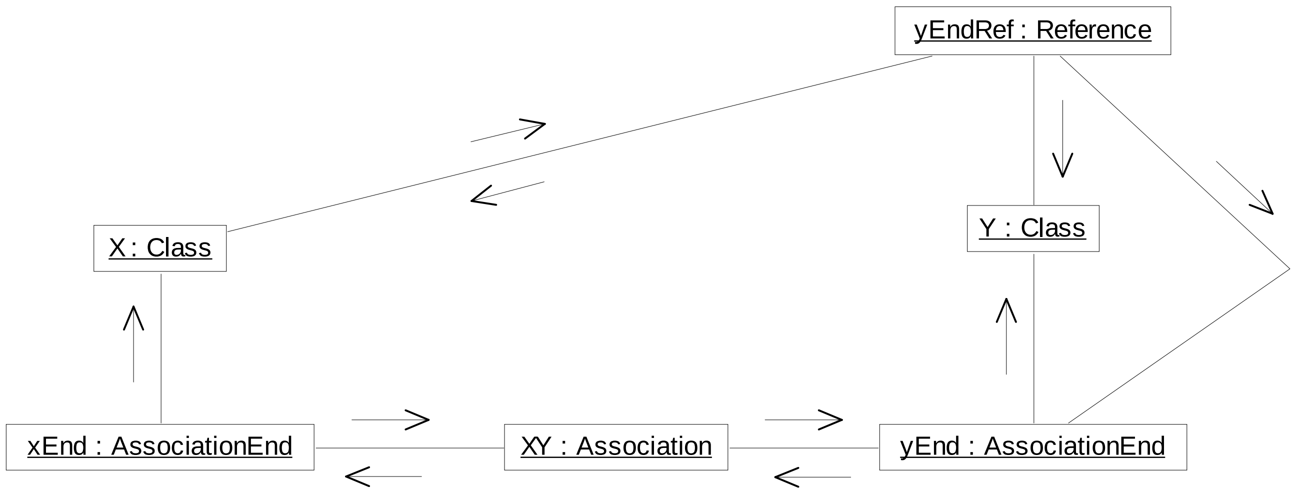

Figure 5-6 Collaboration Diagram showing Association Instantiated in Terms of MOF

In Figure 5-6, the message arrows are based on the navigation in the MOF meta-metamodel and indicate structural knowledge and potential messaging using the resulting interface.

5.2.3 Mapping from MOF to IDL

The description for the mapping from instances of models stored in the MOF is described in detail in the MOF specification. The result of this mapping is the generated IDL in this specification

5.3 Facility Implementation Requirements

Although this chapter focuses on defining the interfaces for the facility and leaves implementation decisions up to the creativity of vendors, there are some implementation requirements.

The UML Standard Elements (stereotypes, constraints, and tags) must be known to a facility implementation, or provided via a load. This is necessary so that the interoperability of these elements can be achieved. The semantics of the standard elements (e.g., containment restrictions) must be enforced. The Standard Elements are documented in the UML Semantics chapter.

The facility interfaces inherit from generic interfaces defined in the Reflective module. These interfaces provide common operations, such as verify(). The verify() operation should be implemented to return well-formedness violations numbered equal to the well-formedness rule following the meta-class definition in the UML Semantics chapter. This includes the semantics for the UML Standard Elements. The Reflective interfaces and exception handling is described in the Meta Object Facility (MOF) specification.

5.4 IDL Modules

5.4.1 Reflective

```
#ifndef REFLECTIVE_IDL
#define REFLECTIVE_IDL

// #include <orb.idl>
module Reflective {

  interface RefBaseObject;

  interface RefObject;
  typedef sequence < RefObject > RefObjectUList;
  typedef sequence < RefObject > RefObjectSet;

  interface RefAssociation;
  interface RefPackage;

  typedef RefObject DesignatorType;
  typedef any ValueType;
  typedef sequence < ValueType > ValueTypeList;
  typedef sequence < RefObject, 2 > Link;
  typedef sequence < ValueType > ErroneousValues;

  const string UNDERFLOW_VIOLATION = "underflow";
  const string OVERFLOW_VIOLATION = "overflow";
  const string DUPLICATE_VIOLATION = "duplicate";
  const string TYPE_CLOSURE_VIOLATION = "type closure";
  const string COMPOSITION_VIOLATION = "composition";
  const string INVALID_OBJECT_VIOLATION = "invalid object";

  struct StructuralViolation {
    string violation_kind;
    RefObject element_designator;
    ErroneousValues offending_values;
  };
  typedef sequence < StructuralViolation > StructuralViolationSet;
  exception StructuralError {
    StructuralViolationSet violations;
  };
  struct ConstraintViolation {
    RefObject constraint_designator;
    ErroneousValues offending_values;
    string explanation_text;
  };
  exception ConstraintError {
    ConstraintViolation violation;
  };
  struct ErrorDescription {
    string error_name;
    ErroneousValues offending_values;
    string explanation_text;
  };
  exception SemanticError {
```

```
  ErrorDescription error;
};

exception NotFound {};
exception NotSet {};
exception BadPosition {};
exception AlreadyCreated {};
exception InvalidLink {};
exception InvalidDesignator {
  DesignatorType designator;
  string element_kind;
};
exception InvalidValue {
  DesignatorType designator;
  string element_kind;
  ValueType value;
  CORBA::TypeCode type_expected;
};
exception InvalidObject {
  DesignatorType designator;
  RefObject obj;
  CORBA::TypeCode type_expected;
};
exception MissingParameter {
  DesignatorType designator;
};
exception TooManyParameters {};
exception OtherException {
  DesignatorType exception_designator;
  ValueTypeList exception_values;
};

interface RefBaseObject {
  DesignatorType meta_object ();
  boolean itself (in RefBaseObject other_object);
  RefBaseObject repository_container ();
}; // end of RefBaseObject


interface RefObject : RefBaseObject {
  boolean is_instance_of (in DesignatorType obj_type,
            in boolean consider_subtypes);
  RefObject create_instance (in ValueTypeList args)
    raises (TooManyParameters,
          MissingParameter,
          InvalidValue,
          AlreadyCreated,
          StructuralError,
            ConstraintError,
        SemanticError);
    RefObjectSet all_objects (in boolean include_subtypes);
```

```
void set_value (in DesignatorType feature,
                in ValueType value)
  raises (InvalidDesignator,
          InvalidValue,
          StructuralError,
          ConstraintError,
           SemanticError);
ValueType value (in DesignatorType feature)
  raises (InvalidDesignator,
          SemanticError);
void add_value (in DesignatorType feature,
                in ValueType value)
  raises (InvalidDesignator,
          InvalidValue,
          StructuralError,
          ConstraintError,
          SemanticError);
void add_value_before (in DesignatorType feature,
                       in ValueType value,
                       in ValueType existing_value)
  raises (InvalidDesignator,
          InvalidValue,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
void add_value_at (in DesignatorType feature,
                   in ValueType value,
                   in long position)
  raises (InvalidDesignator,
          InvalidValue,
          BadPosition,
          StructuralError,
          ConstraintError,
          SemanticError);
void modify_value (in DesignatorType feature,
                   in ValueType existing_value,
                   in ValueType new_value)
  raises (InvalidDesignator,
          InvalidValue,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
void modify_value_at (in DesignatorType feature,
                      in ValueType new_value,
                      in long position)
  raises (InvalidDesignator,
          InvalidValue,
          BadPosition,
          StructuralError,
          ConstraintError,
          SemanticError);
  void remove_value (in DesignatorType feature,
                     in ValueType existing_value)
    raises (InvalidDesignator,
          InvalidValue,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
  void remove_value_at (in DesignatorType feature,
                        in long position)
    raises (InvalidDesignator,
          InvalidValue,
          BadPosition,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
  ValueType invoke_operation (in DesignatorType requested_operation,
                              in ValueTypeList args)
    raises (InvalidDesignator,
          TooManyParameters,
          MissingParameter,
          InvalidValue,
          OtherException,
          ConstraintError,
          SemanticError);
}; // end of interface RefObject

interface RefAssociation : RefBaseObject {
  boolean link_exists (in Link some_link)
    raises (InvalidLink,
          SemanticError);
  RefObjectUList query (in DesignatorType query_end,
                        in RefObject query_object)
    raises (InvalidDesignator,
          InvalidObject,
          SemanticError);
  void add_link (in Link new_link)
    raises (InvalidLink,
          StructuralError,
          ConstraintError,
          SemanticError);
  void add_link_before (in Link new_link,
                        in DesignatorType position_end,
                        in RefObject position_value)
    raises (InvalidDesignator,
          InvalidObject,
          InvalidLink,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
    void modify_link (in Link existing_link,
                in DesignatorType position_end,
                in RefObject position_value)
      raises (InvalidDesignator,
          InvalidObject,
          InvalidLink,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
    void remove_link (in Link existing_link)
      raises (InvalidLink,
          NotFound,
          StructuralError,
          ConstraintError,
          SemanticError);
  }; // end of interface RefAssociation

  interface RefPackage : RefBaseObject {
    RefObject get_class_ref (in DesignatorType type)
      raises (InvalidDesignator);
    RefAssociation get_association (in DesignatorType association)
      raises (InvalidDesignator);
    RefPackage get_nested_package (in DesignatorType nested_package)
      raises (InvalidDesignator);
  }; // end of interface RefPackage
}; // end of module Reflective

#endif


};
```

5.4.2 Foundation

```
#include "Reflective.idl"

module Foundation {
  typedef long Integer;
  typedef long UnlimitedInteger; // -1 means infinity
  typedef float UmlTime;
  enum AggregationKind {ak_none, ak_shared, ak_composite};
  enum CallConcurrencyKind {cck_sequential, cck_guarded, cck_concurrent};
  enum ChangeableKind {ck_none, ck_frozen, ck_addOnly};
enum MessageDirectionKind {mdk_activation, mdk_return};
enum OperationDirectionKind {odk_provide, odk_require};
enum OrderingKind {ok_none, ok_ordered};
enum ParameterDirectionKind {pdk_in, pdk_inout, pdk_out, pdk_return};
enum PseudostateKind {pk_initial, pk_final, pk_shallowHistory,
               pk_deepHistory, pk_join, pk_fork, pk_branch, pk_or};
enum ScopeKind {sk_instance, sk_type};
enum VisibilityKind {vk_public, vk_protected, vk_private};
typedef string LocationReference;
struct Mapping {string body;};
struct MultiplicityRange {Integer lower; UnlimitedInteger upper;};
typedef sequence<MultiplicityRange> Multiplicity;
struct Name {string body;};
struct Expression {Name language; string body;};
typedef Expression ActionExpression;
typedef Expression ArgListsExpression;
typedef Expression BooleanExpression;
typedef Expression IterationExpression;
typedef Expression MappingExpression;
typedef Expression ObjectSetExpression;
typedef Expression ProcedureExpression;
typedef Expression TimeExpression;
typedef Expression TypeExpression;

interface FoundationPackage;

module Core {
  interface ClassifierClass;
  interface Classifier;
  typedef sequence<Classifier> ClassifierSet;
  typedef sequence<Classifier> ClassifierUList;
  interface ClassClass;
  interface Class;
  typedef sequence<Class> ClassUList;
  interface DataTypeClass;
  interface DataType;
  typedef sequence<DataType> DataTypeUList;
  interface StructuralFeatureClass;
  interface StructuralFeature;
  typedef sequence<StructuralFeature> StructuralFeatureUList;
  interface NamespaceClass;
  interface Namespace;
  typedef sequence<Namespace> NamespaceUList;
  interface AssociationEndClass;
  interface AssociationEnd;
  typedef sequence<AssociationEnd> AssociationEndSet;
  typedef sequence<AssociationEnd> AssociationEndUList;
  interface UmlInterfaceClass;
  interface UmlInterface;
  typedef sequence<UmlInterface> UmlInterfaceUList;
  interface UmlConstraintClass;
interface UmlConstraint;
typedef sequence<UmlConstraint> UmlConstraintSet;
typedef sequence<UmlConstraint> UmlConstraintUList;
interface AssociationClass;
interface Association;
typedef sequence<Association> AssociationUList;
interface ElementClass;
interface Element;
typedef sequence<Element> ElementUList;
interface GeneralizableElementClass;
interface GeneralizableElement;
typedef sequence<GeneralizableElement> GeneralizableElementUList;
interface UmlAttributeClass;
interface UmlAttribute;
typedef sequence<UmlAttribute> UmlAttributeUList;
interface OperationClass;
interface Operation;
typedef sequence<Operation> OperationUList;
interface ParameterClass;
interface Parameter;
typedef sequence<Parameter> ParameterSet;
typedef sequence<Parameter> ParameterUList;
interface MethodClass;
interface Method;
typedef sequence<Method> MethodSet;
typedef sequence<Method> MethodUList;
interface GeneralizationClass;
interface Generalization;
typedef sequence<Generalization> GeneralizationSet;
typedef sequence<Generalization> GeneralizationUList;
interface UmlAssociationClassClass;
interface UmlAssociationClass;
typedef sequence<UmlAssociationClass> UmlAssociationClassUList;
interface FeatureClass;
interface Feature;
typedef sequence<Feature> FeatureUList;
interface BehavioralFeatureClass;
interface BehavioralFeature;
typedef sequence<BehavioralFeature> BehavioralFeatureUList;
interface ModelElementClass;
interface ModelElement;
typedef sequence<ModelElement> ModelElementSet;
typedef sequence<ModelElement> ModelElementUList;
interface DependencyClass;
interface Dependency;
typedef sequence<Dependency> DependencySet;
typedef sequence<Dependency> DependencyUList;
interface AbstractionClass;
interface Abstraction;
typedef sequence<Abstraction> AbstractionUList;
interface PresentationElementClass;
interface PresentationElement;
typedef sequence<PresentationElement> PresentationElementSet;
typedef sequence<PresentationElement> PresentationElementUList;
interface UsageClass;
interface Usage;
typedef sequence<Usage> UsageUList;
interface BindingClass;
interface Binding;
typedef sequence<Binding> BindingUList;
interface ComponentClass;
interface Component;
typedef sequence<Component> ComponentSet;
typedef sequence<Component> ComponentUList;
interface NodeClass;
interface Node;
typedef sequence<Node> NodeSet;
typedef sequence<Node> NodeUList;
interface PermissionClass;
interface Permission;
typedef sequence<Permission> PermissionUList;
interface CommentClass;
interface Comment;
typedef sequence<Comment> CommentUList;
interface FlowClass;
interface Flow;
typedef sequence<Flow> FlowSet;
typedef sequence<Flow> FlowUList;
interface RelationshipClass;
interface Relationship;
typedef sequence<Relationship> RelationshipUList;
interface ElementResidenceClass;
interface ElementResidence;
typedef sequence<ElementResidence> ElementResidenceSet;
typedef sequence<ElementResidence> ElementResidenceUList;
interface TemplateParameterClass;
interface TemplateParameter;
typedef sequence<TemplateParameter> TemplateParameterUList;
interface CorePackage;

interface ElementClass : Reflective::RefObject {
  readonly attribute ElementUList all_of_type_element;
};

interface Element : ElementClass {
}; // end of interface Element

interface ModelElementClass : ElementClass {
  readonly attribute ModelElementUList all_of_type_model_element;
};

interface ModelElement : ModelElementClass, Element {
```

```
Foundation::Name name ()
  raises (Reflective::SemanticError);
void set_name (in Foundation::Name new_value)
  raises (Reflective::SemanticError);
VisibilityKind visibility ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_visibility (in VisibilityKind new_value)
  raises (Reflective::SemanticError);
void unset_visibility ()
  raises (Reflective::SemanticError);
Core::Namespace namespace ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_namespace (in Core::Namespace new_value)
  raises (Reflective::SemanticError);
void unset_namespace ()
  raises (Reflective::SemanticError);
DependencySet client_dependency ()
  raises (Reflective::SemanticError);
void set_client_dependency (in DependencySet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_client_dependency (in Dependency new_value)
  raises (Reflective::StructuralError);
void modify_client_dependency (
  in Dependency old_value,
  in Dependency new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_client_dependency (in Dependency old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
UmlConstraintSet uml_constraint ()
  raises (Reflective::SemanticError);
void set_uml_constraint (in UmlConstraintSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_uml_constraint (in UmlConstraint new_value)
  raises (Reflective::StructuralError);
void modify_uml_constraint (
  in UmlConstraint old_value,
  in UmlConstraint new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_uml_constraint (in UmlConstraint old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
DependencySet supplier_dependency ()
  raises (Reflective::SemanticError);
void set_supplier_dependency (in DependencySet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_supplier_dependency (in Dependency new_value)
  raises (Reflective::StructuralError);
void modify_supplier_dependency (
  in Dependency old_value,
  in Dependency new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_supplier_dependency (in Dependency old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
PresentationElementSet presentation ()
  raises (Reflective::SemanticError);
void set_presentation (in PresentationElementSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_presentation (in PresentationElement new_value)
  raises (Reflective::StructuralError);
void modify_presentation (
  in PresentationElement old_value,
  in PresentationElement new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_presentation (in PresentationElement old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ComponentSet implementation_location ()
  raises (Reflective::SemanticError);
```

```
void set_implementation_location (in ComponentSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_implementation_location (in Component new_value)
  raises (Reflective::StructuralError);
void modify_implementation_location (
  in Component old_value,
  in Component new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_implementation_location (in Component old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ModelElementUList template_parameter ()
  raises (Reflective::SemanticError);
void set_template_parameter (in ModelElementUList new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_template_parameter (in ModelElement new_value)
  raises (Reflective::StructuralError);
void add_template_parameter_before (
  in ModelElement new_value,
  in ModelElement before)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void modify_template_parameter (
  in ModelElement old_value,
  in ModelElement new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_template_parameter (in ModelElement old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ModelElement template ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_template (in ModelElement new_value)
  raises (Reflective::SemanticError);
```

```
void unset_template ()
  raises (Reflective::SemanticError);
Core::Binding binding ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_binding (in Core::Binding new_value)
  raises (Reflective::SemanticError);
void unset_binding ()
  raises (Reflective::SemanticError);
FlowSet target_flow ()
  raises (Reflective::SemanticError);
void set_target_flow (in FlowSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_target_flow (in Flow new_value)
  raises (Reflective::StructuralError);
void modify_target_flow (
  in Flow old_value,
  in Flow new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_target_flow (in Flow old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
FlowSet source_flow ()
  raises (Reflective::SemanticError);
void set_source_flow (in FlowSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_source_flow (in Flow new_value)
  raises (Reflective::StructuralError);
void modify_source_flow (
  in Flow old_value,
  in Flow new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_source_flow (in Flow old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
TemplateParameter default_for_template_parameter ()
```

```
          raises (
            Reflective::NotSet,
            Reflective::SemanticError);
        void set_default_for_template_parameter (
          in TemplateParameter new_value)
          raises (Reflective::SemanticError);
        void unset_default_for_template_parameter ()
          raises (Reflective::SemanticError);
        Core::Binding binding1 ()
          raises (
            Reflective::NotSet,
            Reflective::SemanticError);
        void set_binding1 (in Core::Binding new_value)
          raises (Reflective::SemanticError);
        void unset_binding1 ()
          raises (Reflective::SemanticError);
        ElementResidenceSet implementation_residence ()
          raises (Reflective::SemanticError);
        void set_implementation_residence (in ElementResidenceSet
new_value)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void add_implementation_residence (in ElementResidence new_value)
          raises (Reflective::StructuralError);
        void modify_implementation_residence (
          in ElementResidence old_value,
          in ElementResidence new_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove_implementation_residence (in ElementResidence
old_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        TemplateParameter used_as_template_parameter ()
          raises (
            Reflective::NotSet,
            Reflective::SemanticError);
        void set_used_as_template_parameter (in TemplateParameter
new_value)
          raises (Reflective::SemanticError);
        void unset_used_as_template_parameter ()
          raises (Reflective::SemanticError);
        TemplateParameterUList owned_template_parameter ()
          raises (Reflective::SemanticError);
        void set_owned_template_parameter (
          in TemplateParameterUList new_value)
```

```
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_owned_template_parameter (in TemplateParameter
new_value)
      raises (Reflective::StructuralError);
    void add_owned_template_parameter_before (
      in TemplateParameter new_value,
      in TemplateParameter before)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_owned_template_parameter (
      in TemplateParameter old_value,
      in TemplateParameter new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_owned_template_parameter (
      in TemplateParameter old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ModelElement

  interface GeneralizableElementClass : ModelElementClass {
    readonly attribute
      GeneralizableElementUList all_of_type_generalizable_element;
  };

  interface GeneralizableElement : GeneralizableElementClass,
                                   ModelElement {
    boolean is_root ()
      raises (Reflective::SemanticError);
    void set_is_root (in boolean new_value)
      raises (Reflective::SemanticError);
    boolean is_leaf ()
      raises (Reflective::SemanticError);
    void set_is_leaf (in boolean new_value)
      raises (Reflective::SemanticError);
    boolean is_abstract ()
      raises (Reflective::SemanticError);
    void set_is_abstract (in boolean new_value)
      raises (Reflective::SemanticError);
    GeneralizationSet generalization ()
      raises (Reflective::SemanticError);
    void set_generalization (in GeneralizationSet new_value)
      raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_generalization (in Core::Generalization new_value)
    raises (Reflective::StructuralError);
  void modify_generalization (
    in Core::Generalization old_value,
    in Core::Generalization new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_generalization (in Core::Generalization old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  GeneralizationSet specialization ()
    raises (Reflective::SemanticError);
  void set_specialization (in GeneralizationSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_specialization (in Core::Generalization new_value)
    raises (Reflective::StructuralError);
  void modify_specialization (
    in Core::Generalization old_value,
    in Core::Generalization new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_specialization (in Core::Generalization old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface GeneralizableElement

interface NamespaceClass : ModelElementClass {
  readonly attribute Core::NamespaceUList all_of_type_namespace;
  Core::Namespace create_namespace (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Namespace : NamespaceClass, ModelElement {
  ModelElementSet owned_element ()
    raises (Reflective::SemanticError);
  void set_owned_element (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_owned_element (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_owned_element (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_owned_element (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Namespace

interface ClassifierClass : GeneralizableElementClass,
                   Core::NamespaceClass {
  readonly attribute ClassifierUList all_of_type_classifier;
  Classifier create_classifier (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Classifier : ClassifierClass, GeneralizableElement,
                Core::Namespace {
  FeatureUList feature ()
    raises (Reflective::SemanticError);
  void set_feature (in FeatureUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_feature (in Core::Feature new_value)
    raises (Reflective::StructuralError);
  void add_feature_before (
    in Core::Feature new_value,
    in Core::Feature before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_feature (
    in Core::Feature old_value,
    in Core::Feature new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_feature (in Core::Feature old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  AssociationEndSet participant ()
    raises (Reflective::SemanticError);
  void set_participant (in AssociationEndSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_participant (in AssociationEnd new_value)
    raises (Reflective::StructuralError);
  void modify_participant (
    in AssociationEnd old_value,
    in AssociationEnd new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_participant (in AssociationEnd old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Classifier

interface ClassClass : ClassifierClass {
  readonly attribute ClassUList all_of_type_class;
  Class create_class (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract,
    in boolean is_active)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Class : ClassClass, Classifier {
  boolean is_active ()
    raises (Reflective::SemanticError);
  void set_is_active (in boolean new_value)
    raises (Reflective::SemanticError);
}; // end of interface Class

interface DataTypeClass : ClassifierClass {
  readonly attribute DataTypeUList all_of_type_data_type;
  DataType create_data_type (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface DataType : DataTypeClass, Classifier {
}; // end of interface DataType

interface FeatureClass : ModelElementClass {
  readonly attribute FeatureUList all_of_type_feature;
};

interface Feature : FeatureClass, ModelElement {
  ScopeKind owner_scope ()
    raises (Reflective::SemanticError);
  void set_owner_scope (in ScopeKind new_value)
    raises (Reflective::SemanticError);
  Classifier owner ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_owner (in Classifier new_value)
    raises (Reflective::SemanticError);
  void unset_owner ()
    raises (Reflective::SemanticError);
}; // end of interface Feature

interface StructuralFeatureClass : FeatureClass {
  readonly attribute StructuralFeatureUList
    all_of_type_structural_feature;
};

interface StructuralFeature : StructuralFeatureClass, Feature {
  Foundation::Multiplicity multiplicity ()
    raises (Reflective::SemanticError);
  void set_multiplicity (in Foundation::Multiplicity new_value)
    raises (Reflective::SemanticError);
  ChangeableKind changeability ()
    raises (Reflective::SemanticError);
  void set_changeability (in ChangeableKind new_value)
    raises (Reflective::SemanticError);
  ScopeKind target_scope ()
    raises (Reflective::SemanticError);
  void set_target_scope (in ScopeKind new_value)
    raises (Reflective::SemanticError);
  Classifier type ()
    raises (Reflective::SemanticError);
  void set_type (in Classifier new_value)
    raises (Reflective::SemanticError);
}; // end of interface StructuralFeature

interface AssociationEndClass : ModelElementClass {
  readonly attribute AssociationEndUList all_of_type_association_end;
  AssociationEnd create_association_end (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_navigable,
    in OrderingKind ordering,
    in AggregationKind aggregation,
    in ScopeKind target_scope,
    in Foundation::Multiplicity multiplicity,
    in ChangeableKind changeability)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface AssociationEnd : AssociationEndClass, ModelElement {
  boolean is_navigable ()
    raises (Reflective::SemanticError);
  void set_is_navigable (in boolean new_value)
    raises (Reflective::SemanticError);
  OrderingKind ordering ()
    raises (Reflective::SemanticError);
  void set_ordering (in OrderingKind new_value)
    raises (Reflective::SemanticError);
  AggregationKind aggregation ()
    raises (Reflective::SemanticError);
  void set_aggregation (in AggregationKind new_value)
    raises (Reflective::SemanticError);
  ScopeKind target_scope ()
    raises (Reflective::SemanticError);
  void set_target_scope (in ScopeKind new_value)
    raises (Reflective::SemanticError);
  Foundation::Multiplicity multiplicity ()
    raises (Reflective::SemanticError);
  void set_multiplicity (in Foundation::Multiplicity new_value)
    raises (Reflective::SemanticError);
  ChangeableKind changeability ()
    raises (Reflective::SemanticError);
```

```
void set_changeability (in ChangeableKind new_value)
  raises (Reflective::SemanticError);
Core::Association association ()
  raises (Reflective::SemanticError);
void set_association (in Core::Association new_value)
  raises (Reflective::SemanticError);
UmlAttributeUList qualifier ()
  raises (Reflective::SemanticError);
void set_qualifier (in UmlAttributeUList new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_qualifier (in UmlAttribute new_value)
  raises (Reflective::StructuralError);
void add_qualifier_before (
  in UmlAttribute new_value,
  in UmlAttribute before)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void modify_qualifier (
  in UmlAttribute old_value,
  in UmlAttribute new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_qualifier (in UmlAttribute old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
Classifier type ()
  raises (Reflective::SemanticError);
void set_type (in Classifier new_value)
  raises (Reflective::SemanticError);
ClassifierSet specification ()
  raises (Reflective::SemanticError);
void set_specification (in ClassifierSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_specification (in Classifier new_value)
  raises (Reflective::StructuralError);
void modify_specification (
  in Classifier old_value,
  in Classifier new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
      Reflective::SemanticError);
  void remove_specification (in Classifier old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AssociationEnd

interface UmlInterfaceClass : ClassifierClass {
  readonly attribute UmlInterfaceUList all_of_type_uml_interface;
  UmlInterface create_uml_interface (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlInterface : UmlInterfaceClass, Classifier {
}; // end of interface UmlInterface

interface UmlConstraintClass : ModelElementClass {
  readonly attribute UmlConstraintUList all_of_type_uml_constraint;
  UmlConstraint create_uml_constraint (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in BooleanExpression body)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlConstraint : UmlConstraintClass, ModelElement {
  BooleanExpression body ()
    raises (Reflective::SemanticError);
  void set_body (in BooleanExpression new_value)
    raises (Reflective::SemanticError);
  ModelElementUList constrained_element ()
    raises (Reflective::SemanticError);
  void set_constrained_element (in ModelElementUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_constrained_element ()
    raises (Reflective::SemanticError);
  void add_constrained_element (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void add_constrained_element_before (
    in ModelElement new_value,
    in ModelElement before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_constrained_element (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_constrained_element (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface UmlConstraint

interface RelationshipClass : ModelElementClass {
  readonly attribute RelationshipUList all_of_type_relationship;
  Relationship create_relationship (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Relationship : RelationshipClass, ModelElement {
}; // end of interface Relationship

interface AssociationClass : GeneralizableElementClass,
                   RelationshipClass {
  readonly attribute AssociationUList all_of_type_association;
  Association create_association (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Association : AssociationClass,
                GeneralizableElement, Relationship {
  AssociationEndSet connection ()
    raises (Reflective::SemanticError);
  void set_connection (in AssociationEndSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_connection (in AssociationEnd new_value)
    raises (Reflective::StructuralError);
  void modify_connection (
    in AssociationEnd old_value,
    in AssociationEnd new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_connection (in AssociationEnd old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Association

interface UmlAttributeClass : StructuralFeatureClass {
  readonly attribute UmlAttributeUList all_of_type_uml_attribute;
  UmlAttribute create_uml_attribute (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in ScopeKind owner_scope,
    in Foundation::Multiplicity multiplicity,
    in ChangeableKind changeability,
    in ScopeKind target_scope,
    in Expression initial_value)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlAttribute : UmlAttributeClass, StructuralFeature {
  Expression initial_value ()
    raises (Reflective::SemanticError);
  void set_initial_value (in Expression new_value)
    raises (Reflective::SemanticError);
  AssociationEnd association_end ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_association_end (in AssociationEnd new_value)
    raises (Reflective::SemanticError);
  void unset_association_end ()
    raises (Reflective::SemanticError);
}; // end of interface UmlAttribute

interface BehavioralFeatureClass : FeatureClass {
  readonly attribute BehavioralFeatureUList
    all_of_type_behavioral_feature;
};

interface BehavioralFeature : BehavioralFeatureClass, Feature {
  boolean is_query ()
    raises (Reflective::SemanticError);
  void set_is_query (in boolean new_value)
    raises (Reflective::SemanticError);
  ParameterUList parameter ()
    raises (Reflective::SemanticError);
  void set_parameter (in ParameterUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_parameter (in Core::Parameter new_value)
    raises (Reflective::StructuralError);
  void add_parameter_before (
    in Core::Parameter new_value,
    in Core::Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameter (
    in Core::Parameter old_value,
    in Core::Parameter new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_parameter (in Core::Parameter old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface BehavioralFeature

interface OperationClass : BehavioralFeatureClass {
  readonly attribute OperationUList all_of_type_operation;
  Operation create_operation (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in ScopeKind owner_scope,
    in boolean is_query,
    in CallConcurrencyKind concurrency,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Operation : OperationClass, BehavioralFeature {
  CallConcurrencyKind concurrency ()
    raises (Reflective::SemanticError);
  void set_concurrency (in CallConcurrencyKind new_value)
    raises (Reflective::SemanticError);
  boolean is_root ()
    raises (Reflective::SemanticError);
  void set_is_root (in boolean new_value)
    raises (Reflective::SemanticError);
  boolean is_leaf ()
    raises (Reflective::SemanticError);
  void set_is_leaf (in boolean new_value)
    raises (Reflective::SemanticError);
  boolean is_abstract ()
    raises (Reflective::SemanticError);
  void set_is_abstract (in boolean new_value)
    raises (Reflective::SemanticError);
  MethodSet method ()
    raises (Reflective::SemanticError);
  void set_method (in MethodSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_method (in Core::Method new_value)
    raises (Reflective::StructuralError);
  void modify_method (
    in Core::Method old_value,
    in Core::Method new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_method (in Core::Method old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Operation

interface ParameterClass : ModelElementClass {
  readonly attribute ParameterUList all_of_type_parameter;
  Parameter create_parameter (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in Expression default_value,
    in ParameterDirectionKind kind)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Parameter : ParameterClass, ModelElement {
  Expression default_value ()
    raises (Reflective::SemanticError);
  void set_default_value (in Expression new_value)
    raises (Reflective::SemanticError);
  ParameterDirectionKind kind ()
    raises (Reflective::SemanticError);
  void set_kind (in ParameterDirectionKind new_value)
    raises (Reflective::SemanticError);
  BehavioralFeature behavioral_feature ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_behavioral_feature (in BehavioralFeature new_value)
    raises (Reflective::SemanticError);
  void unset_behavioral_feature ()
    raises (Reflective::SemanticError);
  Classifier type ()
    raises (Reflective::SemanticError);
  void set_type (in Classifier new_value)
    raises (Reflective::SemanticError);
}; // end of interface Parameter

interface MethodClass : BehavioralFeatureClass {
  readonly attribute MethodUList all_of_type_method;
  Method create_method (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in ScopeKind owner_scope,
    in boolean is_query,
    in ProcedureExpression body)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Method : MethodClass, BehavioralFeature {
  ProcedureExpression body ()
    raises (Reflective::SemanticError);
  void set_body (in ProcedureExpression new_value)
    raises (Reflective::SemanticError);
  Operation specification ()
    raises (Reflective::SemanticError);
  void set_specification (in Operation new_value)
    raises (Reflective::SemanticError);
}; // end of interface Method

interface GeneralizationClass : RelationshipClass {
  readonly attribute GeneralizationUList all_of_type_generalization;
  Generalization create_generalization (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in Foundation::Name discriminator)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Generalization : GeneralizationClass, Relationship {
  Foundation::Name discriminator ()
    raises (Reflective::SemanticError);
  void set_discriminator (in Foundation::Name new_value)
    raises (Reflective::SemanticError);
  GeneralizableElement child ()
    raises (Reflective::SemanticError);
  void set_child (in GeneralizableElement new_value)
    raises (Reflective::SemanticError);
  GeneralizableElement parent ()
    raises (Reflective::SemanticError);
  void set_parent (in GeneralizableElement new_value)
    raises (Reflective::SemanticError);
}; // end of interface Generalization

interface UmlAssociationClassClass : AssociationClass, ClassClass {
  readonly attribute UmlAssociationClassUList
    all_of_type_uml_association_class;
  UmlAssociationClass create_uml_association_class (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract,
    in boolean is_active)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlAssociationClass : UmlAssociationClassClass,
                      Association, Class {
}; // end of interface UmlAssociationClass

interface DependencyClass : RelationshipClass {
  readonly attribute DependencyUList all_of_type_dependency;
};

interface Dependency : DependencyClass, Relationship {
  ModelElementSet client ()
    raises (Reflective::SemanticError);
  void set_client (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_client (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_client (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_client (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ModelElementSet supplier ()
    raises (Reflective::SemanticError);
  void set_supplier (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_supplier (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_supplier (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_supplier (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Dependency

interface AbstractionClass : DependencyClass {
  readonly attribute AbstractionUList all_of_type_abstraction;
  Abstraction create_abstraction (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in MappingExpression mapping)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};
```

```
interface Abstraction : AbstractionClass, Dependency {
  MappingExpression mapping ()
    raises (Reflective::SemanticError);
  void set_mapping (in MappingExpression new_value)
    raises (Reflective::SemanticError);
}; // end of interface Abstraction

interface PresentationElementClass : ElementClass {
  readonly attribute PresentationElementUList
    all_of_type_presentation_element;
};

interface PresentationElement : PresentationElementClass, Element {
  ModelElementSet subject ()
    raises (Reflective::SemanticError);
  void set_subject (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_subject (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_subject (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_subject (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface PresentationElement

interface UsageClass : DependencyClass {
  readonly attribute UsageUList all_of_type_usage;
  Usage create_usage (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Usage : UsageClass, Dependency {
}; // end of interface Usage

interface BindingClass : DependencyClass {
  readonly attribute Core::BindingUList all_of_type_binding;
  Core::Binding create_binding (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Binding : BindingClass, Dependency {
  ModelElementSet argument ()
    raises (Reflective::SemanticError);
  void set_argument (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_argument (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_argument (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_argument (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ModelElementSet argument1 ()
    raises (Reflective::SemanticError);
  void set_argument1 (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_argument1 (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_argument1 (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_argument1 (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Binding

interface ComponentClass : ClassifierClass {
  readonly attribute ComponentUList all_of_type_component;
  Component create_component (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Component : ComponentClass, Classifier {
  NodeSet deployment_location ()
    raises (Reflective::SemanticError);
  void set_deployment_location (in NodeSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_deployment_location (in Node new_value)
    raises (Reflective::StructuralError);
  void modify_deployment_location (
    in Node old_value,
    in Node new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_deployment_location (in Node old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ModelElementSet resident ()
    raises (Reflective::SemanticError);
  void set_resident (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_resident (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_resident (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_resident (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ElementResidenceSet element_residence ()
    raises (Reflective::SemanticError);
  void set_element_residence (in ElementResidenceSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_element_residence (in ElementResidence new_value)
    raises (Reflective::StructuralError);
  void modify_element_residence (
    in ElementResidence old_value,
    in ElementResidence new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_element_residence (in ElementResidence old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Component

interface NodeClass : ClassifierClass {
  readonly attribute NodeUList all_of_type_node;
  Node create_node (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Node : NodeClass, Classifier {
  ComponentSet resident ()
    raises (Reflective::SemanticError);
  void set_resident (in ComponentSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_resident (in Component new_value)
    raises (Reflective::StructuralError);
  void modify_resident (
    in Component old_value,
    in Component new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_resident (in Component old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Node

interface PermissionClass : DependencyClass {
  readonly attribute PermissionUList all_of_type_permission;
  Permission create_permission (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Permission : PermissionClass, Dependency {
}; // end of interface Permission

interface CommentClass : ModelElementClass {
  readonly attribute CommentUList all_of_type_comment;
  Comment create_comment (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Comment : CommentClass, ModelElement {
}; // end of interface Comment

interface FlowClass : RelationshipClass {
  readonly attribute FlowUList all_of_type_flow;
  Flow create_flow (
    in Foundation::Name name,
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Flow : FlowClass, Relationship {
  ModelElementSet target ()
    raises (Reflective::SemanticError);
  void set_target (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_target (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_target (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_target (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ModelElementSet source ()
    raises (Reflective::SemanticError);
  void set_source (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_source (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_source (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_source (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Flow

interface ElementResidenceClass : Reflective::RefObject {
  readonly attribute ElementResidenceUList
    all_of_type_element_residence;
  ElementResidence create_element_residence (
    in VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface ElementResidence : ElementResidenceClass {
  VisibilityKind visibility ()
    raises (Reflective::SemanticError);
  void set_visibility (in VisibilityKind new_value)
```

```
    raises (Reflective::SemanticError);
  Core::Component component ()
    raises (Reflective::SemanticError);
  void set_component (in Core::Component new_value)
    raises (Reflective::SemanticError);
  ModelElement model_element ()
    raises (Reflective::SemanticError);
  void set_model_element (in ModelElement new_value)
    raises (Reflective::SemanticError);
}; // end of interface ElementResidence

interface TemplateParameterClass : Reflective::RefObject {
  readonly attribute TemplateParameterUList
    all_of_type_template_parameter;
  TemplateParameter create_template_parameter ()
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface TemplateParameter : TemplateParameterClass {
  ModelElementSet default_element ()
    raises (Reflective::SemanticError);
  void set_default_element (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_default_element ()
    raises (Reflective::SemanticError);
  void add_default_element (in ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_default_element (
    in ModelElement old_value,
    in ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_default_element (in ModelElement old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ModelElement parameter_element ()
    raises (Reflective::SemanticError);
  void set_parameter_element (in ModelElement new_value)
    raises (Reflective::SemanticError);
  ModelElement template_element ()
    raises (Reflective::SemanticError);
  void set_template_element (in ModelElement new_value)
    raises (Reflective::SemanticError);
```

```
}; // end of interface TemplateParameter

struct AAssociationConnectionLink {
  Association association;
  AssociationEnd connection;
};
typedef sequence<AAssociationConnectionLink>
  AAssociationConnectionLinkSet;

interface AAssociationConnection : Reflective::RefAssociation {
 AAssociationConnectionLinkSet all_a_association_connection_links();
  boolean exists (
    in Association association,
    in AssociationEnd connection);
  AssociationEndSet with_association (
    in Association association);
  Association with_connection (
    in AssociationEnd connection);
  void add (
    in Association association,
    in AssociationEnd connection)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_association (
    in Association association,
    in AssociationEnd connection,
    in Association new_association)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_connection (
    in Association association,
    in AssociationEnd connection,
    in AssociationEnd new_connection)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Association association,
    in AssociationEnd connection)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AAssociationConnection

struct AOwnerFeatureLink {
  Classifier owner;
  Feature feature;
};
typedef sequence<AOwnerFeatureLink> AOwnerFeatureLinkSet;

interface AOwnerFeature : Reflective::RefAssociation {
  AOwnerFeatureLinkSet all_a_owner_feature_links();
  boolean exists (
    in Classifier owner,
    in Feature feature);
  FeatureUList with_owner (
    in Classifier owner);
  Classifier with_feature (
    in Feature feature);
  void add (
    in Classifier owner,
    in Feature feature)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_feature (
    in Classifier owner,
    in Feature feature,
    in Feature before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_owner (
    in Classifier owner,
    in Feature feature,
    in Classifier new_owner)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_feature (
    in Classifier owner,
    in Feature feature,
    in Feature new_feature)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Classifier owner,
    in Feature feature)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AOwnerFeature
```

```
  struct ASpecificationMethodLink {
    Operation specification;
    Method method;
  };
  typedef sequence<ASpecificationMethodLink> ASpecification-
MethodLinkSet;

  interface ASpecificationMethod : Reflective::RefAssociation {
    ASpecificationMethodLinkSet all_a_specification_method_links();
    boolean exists (
      in Operation specification,
      in Method method);
    MethodSet with_specification (
      in Operation specification);
    Operation with_method (
      in Method method);
    void add (
      in Operation specification,
      in Method method)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_specification (
      in Operation specification,
      in Method method,
      in Operation new_specification)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_method (
      in Operation specification,
      in Method method,
      in Method new_method)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Operation specification,
      in Method method)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ASpecificationMethod

  struct AStructuralFeatureTypeLink {
    StructuralFeature structural_feature;
    Classifier type;
```

```
};
typedef sequence<AStructuralFeatureTypeLink>
  AStructuralFeatureTypeLinkSet;

interface AStructuralFeatureType : Reflective::RefAssociation {
  AStructuralFeatureTypeLinkSet
    all_a_structural_feature_type_links();
  boolean exists (
    in StructuralFeature structural_feature,
    in Classifier type);
  Classifier with_structural_feature (
    in StructuralFeature structural_feature);
  StructuralFeatureUList with_type (
    in Classifier type);
  void add (
    in StructuralFeature structural_feature,
    in Classifier type)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_structural_feature (
    in StructuralFeature structural_feature,
    in Classifier type,
    in StructuralFeature before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_structural_feature (
    in StructuralFeature structural_feature,
    in Classifier type,
    in StructuralFeature new_structural_feature)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_type (
    in StructuralFeature structural_feature,
    in Classifier type,
    in Classifier new_type)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in StructuralFeature structural_feature,
    in Classifier type)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
```

```
}; // end of interface AStructuralFeatureType

struct ANamespaceOwnedElementLink {
  Namespace namespace;
  ModelElement owned_element;
};
typedef sequence<ANamespaceOwnedElementLink>
  ANamespaceOwnedElementLinkSet;

interface ANamespaceOwnedElement : Reflective::RefAssociation {
  ANamespaceOwnedElementLinkSet
    all_a_namespace_owned_element_links();
  boolean exists (
    in Namespace namespace,
    in ModelElement owned_element);
  ModelElementSet with_namespace (
    in Namespace namespace);
  Namespace with_owned_element (
    in ModelElement owned_element);
  void add (
    in Namespace namespace,
    in ModelElement owned_element)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_namespace (
    in Namespace namespace,
    in ModelElement owned_element,
    in Namespace new_namespace)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_owned_element (
    in Namespace namespace,
    in ModelElement owned_element,
    in ModelElement new_owned_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Namespace namespace,
    in ModelElement owned_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ANamespaceOwnedElement

struct ABehavioralFeatureParameterLink {
  BehavioralFeature behavioral_feature;
  Parameter parameter;
};
typedef sequence<ABehavioralFeatureParameterLink>
  ABehavioralFeatureParameterLinkSet;

interface ABehavioralFeatureParameter : Reflective::RefAssociation {
  ABehavioralFeatureParameterLinkSet
    all_a_behavioral_feature_parameter_links();
  boolean exists (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter);
  ParameterUList with_behavioral_feature (
    in BehavioralFeature behavioral_feature);
  BehavioralFeature with_parameter (
    in Parameter parameter);
  void add (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_parameter (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter,
    in Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_behavioral_feature (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter,
    in BehavioralFeature new_behavioral_feature)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameter (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter,
    in Parameter new_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in BehavioralFeature behavioral_feature,
    in Parameter parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ABehavioralFeatureParameter

struct AParameterTypeLink {
  Parameter parameter;
  Classifier type;
};
typedef sequence<AParameterTypeLink> AParameterTypeLinkSet;

interface AParameterType : Reflective::RefAssociation {
  AParameterTypeLinkSet all_a_parameter_type_links();
  boolean exists (
    in Parameter parameter,
    in Classifier type);
  Classifier with_parameter (
    in Parameter parameter);
  ParameterSet with_type (
    in Classifier type);
  void add (
    in Parameter parameter,
    in Classifier type)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_parameter (
    in Parameter parameter,
    in Classifier type,
    in Parameter new_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_type (
    in Parameter parameter,
    in Classifier type,
    in Classifier new_type)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Parameter parameter,
    in Classifier type)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AParameterType

struct AChildGeneralizationLink {
  GeneralizableElement child;
  Generalization generalization;
};
typedef sequence<AChildGeneralizationLink>
  AChildGeneralizationLinkSet;

interface AChildGeneralization : Reflective::RefAssociation {
  AChildGeneralizationLinkSet all_a_child_generalization_links();
  boolean exists (
    in GeneralizableElement child,
    in Generalization generalization);
  GeneralizationSet with_child (
    in GeneralizableElement child);
  GeneralizableElement with_generalization (
    in Generalization generalization);
  void add (
    in GeneralizableElement child,
    in Generalization generalization)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_child (
    in GeneralizableElement child,
    in Generalization generalization,
    in GeneralizableElement new_child)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_generalization (
    in GeneralizableElement child,
    in Generalization generalization,
    in Generalization new_generalization)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in GeneralizableElement child,
    in Generalization generalization)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AChildGeneralization

struct AParentSpecializationLink {
  GeneralizableElement parent;
  Generalization specialization;
};
typedef sequence<AParentSpecializationLink>
```

```
  AParentSpecializationLinkSet;

interface AParentSpecialization : Reflective::RefAssociation {
  AParentSpecializationLinkSet all_a_parent_specialization_links();
  boolean exists (
    in GeneralizableElement parent,
    in Generalization specialization);
  GeneralizationSet with_parent (
    in GeneralizableElement parent);
  GeneralizableElement with_specialization (
    in Generalization specialization);
  void add (
    in GeneralizableElement parent,
    in Generalization specialization)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_parent (
    in GeneralizableElement parent,
    in Generalization specialization,
    in GeneralizableElement new_parent)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_specialization (
    in GeneralizableElement parent,
    in Generalization specialization,
    in Generalization new_specialization)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in GeneralizableElement parent,
    in Generalization specialization)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AParentSpecialization

struct AQualifierAssociationEndLink {
  UmlAttribute qualifier;
  AssociationEnd association_end;
};
typedef sequence<AQualifierAssociationEndLink>
  AQualifierAssociationEndLinkSet;

interface AQualifierAssociationEnd : Reflective::RefAssociation {
  AQualifierAssociationEndLinkSet
```

```
    all_a_qualifier_association_end_links();
  boolean exists (
    in UmlAttribute qualifier,
    in AssociationEnd association_end);
  AssociationEnd with_qualifier (
    in UmlAttribute qualifier);
  UmlAttributeUList with_association_end (
    in AssociationEnd association_end);
  void add (
    in UmlAttribute qualifier,
    in AssociationEnd association_end)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_qualifier (
    in UmlAttribute qualifier,
    in AssociationEnd association_end,
    in UmlAttribute before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_qualifier (
    in UmlAttribute qualifier,
    in AssociationEnd association_end,
    in UmlAttribute new_qualifier)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_association_end (
    in UmlAttribute qualifier,
    in AssociationEnd association_end,
    in AssociationEnd new_association_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in UmlAttribute qualifier,
    in AssociationEnd association_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AQualifierAssociationEnd

struct ATypeAssociationEndLink {
  Classifier type;
  AssociationEnd association_end;
};
```

```
  typedef sequence<ATypeAssociationEndLink> ATypeAssociation-
EndLinkSet;

  interface ATypeAssociationEnd : Reflective::RefAssociation {
    ATypeAssociationEndLinkSet all_a_type_association_end_links();
    boolean exists (
      in Classifier type,
      in AssociationEnd association_end);
    AssociationEndSet with_type (
      in Classifier type);
    Classifier with_association_end (
      in AssociationEnd association_end);
    void add (
      in Classifier type,
      in AssociationEnd association_end)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_type (
      in Classifier type,
      in AssociationEnd association_end,
      in Classifier new_type)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_association_end (
      in Classifier type,
      in AssociationEnd association_end,
      in AssociationEnd new_association_end)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Classifier type,
      in AssociationEnd association_end)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ATypeAssociationEnd

  struct AParticipantSpecificationLink {
    AssociationEnd participant;
    Classifier specification;
  };
  typedef sequence<AParticipantSpecificationLink>
    AParticipantSpecificationLinkSet;

  interface AParticipantSpecification : Reflective::RefAssociation {
```

```
  AParticipantSpecificationLinkSet
    all_a_participant_specification_links();
  boolean exists (
    in AssociationEnd participant,
    in Classifier specification);
  ClassifierSet with_participant (
    in AssociationEnd participant);
  AssociationEndSet with_specification (
    in Classifier specification);
  void add (
    in AssociationEnd participant,
    in Classifier specification)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_participant (
    in AssociationEnd participant,
    in Classifier specification,
    in AssociationEnd new_participant)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_specification (
    in AssociationEnd participant,
    in Classifier specification,
    in Classifier new_specification)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in AssociationEnd participant,
    in Classifier specification)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AParticipantSpecification

struct AClientClientDependencyLink {
  ModelElement client;
  Dependency client_dependency;
};
typedef sequence<AClientClientDependencyLink>
  AClientClientDependencyLinkSet;

interface AClientClientDependency : Reflective::RefAssociation {
  AClientClientDependencyLinkSet
    all_a_client_client_dependency_links();
  boolean exists (
    in ModelElement client,
    in Dependency client_dependency);
  DependencySet with_client (
    in ModelElement client);
  ModelElementSet with_client_dependency (
    in Dependency client_dependency);
  void add (
    in ModelElement client,
    in Dependency client_dependency)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_client (
    in ModelElement client,
    in Dependency client_dependency,
    in ModelElement new_client)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_client_dependency (
    in ModelElement client,
    in Dependency client_dependency,
    in Dependency new_client_dependency)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement client,
    in Dependency client_dependency)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AClientClientDependency

struct AConstrainedElementConstraintLink {
  ModelElement constrained_element;
  UmlConstraint uml_constraint;
};
typedef sequence<AConstrainedElementConstraintLink>
  AConstrainedElementConstraintLinkSet;

interface AConstrainedElementConstraint : Reflective::RefAssociation {
  AConstrainedElementConstraintLinkSet
    all_a_constrained_element_constraint_links();
  boolean exists (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint);
  UmlConstraintSet with_constrained_element (
    in ModelElement constrained_element);
  ModelElementUList with_uml_constraint (
    in UmlConstraint uml_constraint);
  void add (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_constrained_element (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint,
    in ModelElement before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_constrained_element (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint,
    in ModelElement new_constrained_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_uml_constraint (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint,
    in UmlConstraint new_uml_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement constrained_element,
    in UmlConstraint uml_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AConstrainedElementConstraint

struct ASupplierSupplierDependencyLink {
  ModelElement supplier;
  Dependency supplier_dependency;
};
typedef sequence<ASupplierSupplierDependencyLink>
  ASupplierSupplierDependencyLinkSet;

interface ASupplierSupplierDependency : Reflective::RefAssociation {
  ASupplierSupplierDependencyLinkSet
```

```
            all_a_supplier_supplier_dependency_links();
        boolean exists (
            in ModelElement supplier,
            in Dependency supplier_dependency);
        DependencySet with_supplier (
            in ModelElement supplier);
        ModelElementSet with_supplier_dependency (
            in Dependency supplier_dependency);
        void add (
            in ModelElement supplier,
            in Dependency supplier_dependency)
            raises (
                Reflective::StructuralError,
                Reflective::SemanticError);
        void modify_supplier (
            in ModelElement supplier,
            in Dependency supplier_dependency,
            in ModelElement new_supplier)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
        void modify_supplier_dependency (
            in ModelElement supplier,
            in Dependency supplier_dependency,
            in Dependency new_supplier_dependency)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
        void remove (
            in ModelElement supplier,
            in Dependency supplier_dependency)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
    }; // end of interface ASupplierSupplierDependency

    struct APresentationSubjectLink {
        PresentationElement presentation;
        ModelElement subject;
    };
    typedef sequence<APresentationSubjectLink> APresentationSub-
jectLinkSet;

    interface APresentationSubject : Reflective::RefAssociation {
        APresentationSubjectLinkSet all_a_presentation_subject_links();
        boolean exists (
            in PresentationElement presentation,
            in ModelElement subject);
```

```
  ModelElementSet with_presentation (
    in PresentationElement presentation);
  PresentationElementSet with_subject (
    in ModelElement subject);
  void add (
    in PresentationElement presentation,
    in ModelElement subject)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_presentation (
    in PresentationElement presentation,
    in ModelElement subject,
    in PresentationElement new_presentation)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_subject (
    in PresentationElement presentation,
    in ModelElement subject,
    in ModelElement new_subject)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in PresentationElement presentation,
    in ModelElement subject)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface APresentationSubject

struct ADeploymentLocationResidentLink {
  Node deployment_location;
  Component resident;
};
typedef sequence<ADeploymentLocationResidentLink>
  ADeploymentLocationResidentLinkSet;

interface ADeploymentLocationResident : Reflective::RefAssociation {
  ADeploymentLocationResidentLinkSet
    all_a_deployment_location_resident_links();
  boolean exists (
    in Node deployment_location,
    in Component resident);
  ComponentSet with_deployment_location (
    in Node deployment_location);
  NodeSet with_resident (
    in Component resident);
  void add (
    in Node deployment_location,
    in Component resident)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_deployment_location (
    in Node deployment_location,
    in Component resident,
    in Node new_deployment_location)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_resident (
    in Node deployment_location,
    in Component resident,
    in Component new_resident)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Node deployment_location,
    in Component resident)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ADeploymentLocationResident

struct AImplementationLocationResidentLink {
  Component implementation_location;
  ModelElement resident;
};
typedef sequence<AImplementationLocationResidentLink>
  AImplementationLocationResidentLinkSet;

interface AImplementationLocationResident : Reflective::RefAssociation
{
  AImplementationLocationResidentLinkSet
    all_a_implementation_location_resident_links();
  boolean exists (
    in Component implementation_location,
    in ModelElement resident);
  ModelElementSet with_implementation_location (
    in Component implementation_location);
  ComponentSet with_resident (
    in ModelElement resident);
  void add (
    in Component implementation_location,
    in ModelElement resident)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_implementation_location (
    in Component implementation_location,
    in ModelElement resident,
    in Component new_implementation_location)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_resident (
    in Component implementation_location,
    in ModelElement resident,
    in ModelElement new_resident)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Component implementation_location,
    in ModelElement resident)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AImplementationLocationResident

struct ATemplateTemplateParameterLink {
  ModelElement template;
  ModelElement template_parameter;
};
typedef sequence<ATemplateTemplateParameterLink>
  ATemplateTemplateParameterLinkSet;

interface ATemplateTemplateParameter : Reflective::RefAssociation {
  ATemplateTemplateParameterLinkSet
    all_a_template_template_parameter_links();
  boolean exists (
    in ModelElement template,
    in ModelElement template_parameter);
  ModelElementUList with_template (
    in ModelElement template);
  ModelElement with_template_parameter (
    in ModelElement template_parameter);
  void add (
    in ModelElement template,
    in ModelElement template_parameter)
    raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
  void add_before_template_parameter (
    in ModelElement template,
    in ModelElement template_parameter,
    in ModelElement before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_template (
    in ModelElement template,
    in ModelElement template_parameter,
    in ModelElement new_template)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_template_parameter (
    in ModelElement template,
    in ModelElement template_parameter,
    in ModelElement new_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement template,
    in ModelElement template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ATemplateTemplateParameter

struct ABindingArgumentLink {
  Binding binding;
  ModelElement argument;
};
typedef sequence<ABindingArgumentLink> ABindingArgumentLinkSet;

interface ABindingArgument : Reflective::RefAssociation {
  ABindingArgumentLinkSet all_a_binding_argument_links();
  boolean exists (
    in Binding binding,
    in ModelElement argument);
  ModelElementSet with_binding (
    in Binding binding);
  Binding with_argument (
    in ModelElement argument);
  void add (
      in Binding binding,
      in ModelElement argument)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_binding (
      in Binding binding,
      in ModelElement argument,
      in Binding new_binding)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_argument (
      in Binding binding,
      in ModelElement argument,
      in ModelElement new_argument)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Binding binding,
      in ModelElement argument)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ABindingArgument

  struct ATargetFlowTargetLink {
    Flow target_flow;
    ModelElement target;
  };
  typedef sequence<ATargetFlowTargetLink> ATargetFlowTargetLinkSet;

  interface ATargetFlowTarget : Reflective::RefAssociation {
    ATargetFlowTargetLinkSet all_a_target_flow_target_links();
    boolean exists (
      in Flow target_flow,
      in ModelElement target);
    ModelElementSet with_target_flow (
      in Flow target_flow);
    FlowSet with_target (
      in ModelElement target);
    void add (
      in Flow target_flow,
      in ModelElement target)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_target_flow (
      in Flow target_flow,
      in ModelElement target,
      in Flow new_target_flow)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_target (
      in Flow target_flow,
      in ModelElement target,
      in ModelElement new_target)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Flow target_flow,
      in ModelElement target)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ATargetFlowTarget

  struct ASourceFlowSourceLink {
    Flow source_flow;
    ModelElement source;
  };
  typedef sequence<ASourceFlowSourceLink> ASourceFlowSourceLink-
Set;

  interface ASourceFlowSource : Reflective::RefAssociation {
    ASourceFlowSourceLinkSet all_a_source_flow_source_links();
    boolean exists (
      in Flow source_flow,
      in ModelElement source);
    ModelElementSet with_source_flow (
      in Flow source_flow);
    FlowSet with_source (
      in ModelElement source);
    void add (
      in Flow source_flow,
      in ModelElement source)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_source_flow (
      in Flow source_flow,
      in ModelElement source,
      in Flow new_source_flow)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_source (
      in Flow source_flow,
      in ModelElement source,
      in ModelElement new_source)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Flow source_flow,
      in ModelElement source)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ASourceFlowSource

  struct ADefaultElementDefaultForTemplateParameterLink {
    ModelElement default_element;
    TemplateParameter default_for_template_parameter;
  };
  typedef sequence<ADefaultElementDefaultForTemplateParameterLink>
    ADefaultElementDefaultForTemplateParameterLinkSet;

  interface ADefaultElementDefaultForTemplateParameter :
    Reflective::RefAssociation {
    ADefaultElementDefaultForTemplateParameterLinkSet
      all_a_default_element_default_for_template_parameter_links();
    boolean exists (
      in ModelElement default_element,
      in TemplateParameter default_for_template_parameter);
    TemplateParameter with_default_element (
      in ModelElement default_element);
    ModelElementSet with_default_for_template_parameter (
      in TemplateParameter default_for_template_parameter);
    void add (
      in ModelElement default_element,
      in TemplateParameter default_for_template_parameter)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_default_element (
      in ModelElement default_element,
      in TemplateParameter default_for_template_parameter,
      in ModelElement new_default_element)
      raises (
        Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_default_for_template_parameter (
    in ModelElement default_element,
    in TemplateParameter default_for_template_parameter,
    in TemplateParameter new_default_for_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement default_element,
    in TemplateParameter default_for_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ADefaultElementDefaultForTemplateParameter

struct AElementResidenceComponentLink {
  ElementResidence element_residence;
  Component component;
};
typedef sequence<AElementResidenceComponentLink>
  AElementResidenceComponentLinkSet;

interface AElementResidenceComponent : Reflective::RefAssociation {
  AElementResidenceComponentLinkSet
    all_a_element_residence_component_links();
  boolean exists (
    in ElementResidence element_residence,
    in Component component);
  Component with_element_residence (
    in ElementResidence element_residence);
  ElementResidenceSet with_component (
    in Component component);
  void add (
    in ElementResidence element_residence,
    in Component component)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_element_residence (
    in ElementResidence element_residence,
    in Component component,
    in ElementResidence new_element_residence)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_component (
    in ElementResidence element_residence,
    in Component component,
    in Component new_component)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ElementResidence element_residence,
    in Component component)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AElementResidenceComponent

struct AModelElementImplementationResidenceLink {
  ModelElement model_element;
  ElementResidence implementation_residence;
};
typedef sequence<AModelElementImplementationResidenceLink>
  AModelElementImplementationResidenceLinkSet;

interface AModelElementImplementationResidence :
  Reflective::RefAssociation {
  AModelElementImplementationResidenceLinkSet
    all_a_model_element_implementation_residence_links();
  boolean exists (
    in ModelElement model_element,
    in ElementResidence implementation_residence);
  ElementResidenceSet with_model_element (
    in ModelElement model_element);
  ModelElement with_implementation_residence (
    in ElementResidence implementation_residence);
  void add (
    in ModelElement model_element,
    in ElementResidence implementation_residence)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_model_element (
    in ModelElement model_element,
    in ElementResidence implementation_residence,
    in ModelElement new_model_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_implementation_residence (
    in ModelElement model_element,
    in ElementResidence implementation_residence,
    in ElementResidence new_implementation_residence)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement model_element,
    in ElementResidence implementation_residence)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AModelElementImplementationResidence

struct AParameterElementUsedAsTemplateParameterLink {
  ModelElement parameter_element;
  TemplateParameter used_as_template_parameter;
};
typedef sequence<AParameterElementUsedAsTemplateParameterLink>
  AParameterElementUsedAsTemplateParameterLinkSet;

interface AParameterElementUsedAsTemplateParameter :
  Reflective::RefAssociation {
  AParameterElementUsedAsTemplateParameterLinkSet
    all_a_parameter_element_used_as_template_parameter_links();
  boolean exists (
    in ModelElement parameter_element,
    in TemplateParameter used_as_template_parameter);
  TemplateParameter with_parameter_element (
    in ModelElement parameter_element);
  ModelElement with_used_as_template_parameter (
    in TemplateParameter used_as_template_parameter);
  void add (
    in ModelElement parameter_element,
    in TemplateParameter used_as_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_parameter_element (
    in ModelElement parameter_element,
    in TemplateParameter used_as_template_parameter,
    in ModelElement new_parameter_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_used_as_template_parameter (
    in ModelElement parameter_element,
    in TemplateParameter used_as_template_parameter,
    in TemplateParameter new_used_as_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement parameter_element,
    in TemplateParameter used_as_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AParameterElementUsedAsTemplateParameter

struct ATemplateElementOwnedTemplateParameterLink {
  ModelElement template_element;
  TemplateParameter owned_template_parameter;
};
typedef sequence<ATemplateElementOwnedTemplateParameterLink>
  ATemplateElementOwnedTemplateParameterLinkSet;

interface ATemplateElementOwnedTemplateParameter :
  Reflective::RefAssociation {
  ATemplateElementOwnedTemplateParameterLinkSet
    all_a_template_element_owned_template_parameter_links();
  boolean exists (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter);
  TemplateParameterUList with_template_element (
    in ModelElement template_element);
  ModelElement with_owned_template_parameter (
    in TemplateParameter owned_template_parameter);
  void add (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_owned_template_parameter (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter,
    in TemplateParameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_template_element (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter,
    in ModelElement new_template_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_owned_template_parameter (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter,
    in TemplateParameter new_owned_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ModelElement template_element,
    in TemplateParameter owned_template_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ATemplateElementOwnedTemplateParameter

interface CorePackageFactory {
  CorePackage create_core_package ()
    raises (Reflective::SemanticError);
};

interface CorePackage : Reflective::RefPackage {
  readonly attribute ClassifierClass classifier_class_ref;
  readonly attribute ClassClass class_class_ref;
  readonly attribute DataTypeClass data_type_class_ref;
  readonly attribute
    StructuralFeatureClass structural_feature_class_ref;
  readonly attribute NamespaceClass namespace_class_ref;
  readonly attribute AssociationEndClass association_end_class_ref;
  readonly attribute UmlInterfaceClass uml_interface_class_ref;
  readonly attribute UmlConstraintClass uml_constraint_class_ref;
  readonly attribute AssociationClass association_class_ref;
  readonly attribute ElementClass element_class_ref;
  readonly attribute
    GeneralizableElementClass generalizable_element_class_ref;
  readonly attribute UmlAttributeClass uml_attribute_class_ref;
  readonly attribute OperationClass operation_class_ref;
  readonly attribute ParameterClass parameter_class_ref;
  readonly attribute MethodClass method_class_ref;
  readonly attribute GeneralizationClass generalization_class_ref;
  readonly attribute UmlAssociationClassClass
    uml_association_class_class_ref;
  readonly attribute FeatureClass feature_class_ref;
  readonly attribute BehavioralFeatureClass
    behavioral_feature_class_ref;
  readonly attribute ModelElementClass model_element_class_ref;
  readonly attribute DependencyClass dependency_class_ref;
  readonly attribute AbstractionClass abstraction_class_ref;
  readonly attribute
```

```
              PresentationElementClass presentation_element_class_ref;
            readonly attribute UsageClass usage_class_ref;
            readonly attribute BindingClass binding_class_ref;
            readonly attribute ComponentClass component_class_ref;
            readonly attribute NodeClass node_class_ref;
            readonly attribute PermissionClass permission_class_ref;
            readonly attribute CommentClass comment_class_ref;
            readonly attribute FlowClass flow_class_ref;
            readonly attribute RelationshipClass relationship_class_ref;
            readonly attribute ElementResidenceClass
element_residence_class_ref;
            readonly attribute TemplateParameterClass
template_parameter_class_ref;
            readonly attribute
              AAssociationConnection a_association_connection_class_ref;
            readonly attribute AOwnerFeature a_owner_feature_class_ref;
            readonly attribute
              ASpecificationMethod a_specification_method_class_ref;
            readonly attribute AStructuralFeatureType
              a_structural_feature_type_class_ref;
            readonly attribute ANamespaceOwnedElement
              a_namespace_owned_element_class_ref;
            readonly attribute ABehavioralFeatureParameter
              a_behavioral_feature_parameter_class_ref;
            readonly attribute AParameterType a_parameter_type_class_ref;
            readonly attribute AChildGeneralization
              a_child_generalization_class_ref;
            readonly attribute AParentSpecialization
              a_parent_specialization_class_ref;
            readonly attribute AQualifierAssociationEnd
              a_qualifier_association_end_class_ref;
            readonly attribute ATypeAssociationEnd
              a_type_association_end_class_ref;
            readonly attribute AParticipantSpecification
              a_participant_specification_class_ref;
            readonly attribute AClientClientDependency
              a_client_client_dependency_class_ref;
            readonly attribute AConstrainedElementConstraint
              a_constrained_element_constraint_class_ref;
            readonly attribute ASupplierSupplierDependency
              a_supplier_supplier_dependency_class_ref;
            readonly attribute APresentationSubject
              a_presentation_subject_class_ref;
            readonly attribute ADeploymentLocationResident
              a_deployment_location_resident_class_ref;
            readonly attribute AImplementationLocationResident
              a_implementation_location_resident_class_ref;
            readonly attribute ATemplateTemplateParameter
              a_template_template_parameter_class_ref;
            readonly attribute ABindingArgument
              a_binding_argument_class_ref;
```

```
      readonly attribute ATargetFlowTarget
        a_target_flow_target_class_ref;
      readonly attribute ASourceFlowSource
        a_source_flow_source_class_ref;
      readonly attribute ADefaultElementDefaultForTemplateParameter
        a_default_element_default_for_template_parameter_class_ref;
      readonly attribute AElementResidenceComponent
        a_element_residence_component_class_ref;
      readonly attribute AModelElementImplementationResidence
        a_model_element_implementation_residence_class_ref;
      readonly attribute AParameterElementUsedAsTemplateParameter
        a_parameter_element_used_as_template_parameter_class_ref;
      readonly attribute ATemplateElementOwnedTemplateParameter
        a_template_element_owned_template_parameter_class_ref;
    };
  }; // end of module Core

  module DataTypes {
    interface StructureClass;
    interface Structure;
    typedef sequence<Structure> StructureUList;
    interface PrimitiveClass;
    interface Primitive;
    typedef sequence<Primitive> PrimitiveUList;
    interface EnumerationClass;
    interface Enumeration;
    typedef sequence<Enumeration> EnumerationUList;
    interface EnumerationLiteralClass;
    interface EnumerationLiteral;
    typedef sequence<EnumerationLiteral> EnumerationLiteralUList;
    interface ProgrammingLanguageTypeClass;
    interface ProgrammingLanguageType;
    typedef sequence<ProgrammingLanguageType> ProgrammingLanguag-
eTypeUList;
    interface DataTypesPackage;

    interface StructureClass : Core::DataTypeClass {
      readonly attribute StructureUList all_of_type_structure;
      Structure create_structure (
        in Foundation::Name name,
        in VisibilityKind visibility,
        in boolean is_root,
        in boolean is_leaf,
        in boolean is_abstract)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface Structure : StructureClass, Core::DataType {
    }; // end of interface Structure
```

```
interface PrimitiveClass : Core::DataTypeClass {
  readonly attribute PrimitiveUList all_of_type_primitive;
  Primitive create_primitive (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Primitive : PrimitiveClass, Core::DataType {
}; // end of interface Primitive

interface EnumerationClass : Core::DataTypeClass {
  readonly attribute EnumerationUList all_of_type_enumeration;
  Enumeration create_enumeration (
    in Foundation::Name name,
    in VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Enumeration : EnumerationClass, Core::DataType {
  EnumerationLiteralUList literal ()
    raises (Reflective::SemanticError);
  void set_literal (in EnumerationLiteralUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_literal (in EnumerationLiteral new_value)
    raises (Reflective::StructuralError);
  void add_literal_before (
    in EnumerationLiteral new_value,
    in EnumerationLiteral before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_literal (
    in EnumerationLiteral old_value,
    in EnumerationLiteral new_value)
    raises (
      Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void remove_literal (in EnumerationLiteral old_value)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
}; // end of interface Enumeration

interface EnumerationLiteralClass : Reflective::RefObject {
    readonly attribute EnumerationLiteralUList
        all_of_type_enumeration_literal;
    EnumerationLiteral create_enumeration_literal (
        in Foundation::Name name)
        raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
};

interface EnumerationLiteral : EnumerationLiteralClass {
    Foundation::Name name ()
        raises (Reflective::SemanticError);
    void set_name (in Foundation::Name new_value)
        raises (Reflective::SemanticError);
    DataTypes::Enumeration enumeration ()
        raises (Reflective::SemanticError);
    void set_enumeration (in DataTypes::Enumeration new_value)
        raises (Reflective::SemanticError);
}; // end of interface EnumerationLiteral

interface ProgrammingLanguageTypeClass : Core::DataTypeClass {
    readonly attribute ProgrammingLanguageTypeUList
        all_of_type_programming_language_type;
    ProgrammingLanguageType create_programming_language_type (
        in Foundation::Name name,
        in VisibilityKind visibility,
        in boolean is_root,
        in boolean is_leaf,
        in boolean is_abstract,
        in TypeExpression type)
        raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
};

interface ProgrammingLanguageType : ProgrammingLanguageType-
Class,
                            Core::DataType {
    TypeExpression type ()
        raises (Reflective::SemanticError);
    void set_type (in TypeExpression new_value)
```

```
      raises (Reflective::SemanticError);
  }; // end of interface ProgrammingLanguageType

  struct AEnumerationLiteralLink {
    Enumeration enumeration;
    EnumerationLiteral literal;
  };
  typedef sequence<AEnumerationLiteralLink> AEnumerationLiteralLink-
Set;

  interface AEnumerationLiteral : Reflective::RefAssociation {
    AEnumerationLiteralLinkSet all_a_enumeration_literal_links();
    boolean exists (
      in Enumeration enumeration,
      in EnumerationLiteral literal);
    EnumerationLiteralUList with_enumeration (
      in Enumeration enumeration);
    Enumeration with_literal (
      in EnumerationLiteral literal);
    void add (
      in Enumeration enumeration,
      in EnumerationLiteral literal)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_before_literal (
      in Enumeration enumeration,
      in EnumerationLiteral literal,
      in EnumerationLiteral before)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_enumeration (
      in Enumeration enumeration,
      in EnumerationLiteral literal,
      in Enumeration new_enumeration)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_literal (
      in Enumeration enumeration,
      in EnumerationLiteral literal,
      in EnumerationLiteral new_literal)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Enumeration enumeration,
        in EnumerationLiteral literal)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
  }; // end of interface AEnumerationLiteral

  interface DataTypesPackageFactory {
    DataTypesPackage create_data_types_package ()
      raises (Reflective::SemanticError);
  };

  interface DataTypesPackage : Reflective::RefPackage {
    readonly attribute StructureClass structure_class_ref;
    readonly attribute PrimitiveClass primitive_class_ref;
    readonly attribute EnumerationClass enumeration_class_ref;
    readonly attribute
      EnumerationLiteralClass enumeration_literal_class_ref;
    readonly attribute ProgrammingLanguageTypeClass
      programming_language_type_class_ref;
    readonly attribute AEnumerationLiteral
      a_enumeration_literal_class_ref;
  };
}; // end of module DataTypes

module ExtensionMechanisms {
  interface StereotypeClass;
  interface Stereotype;
  typedef sequence<Stereotype> StereotypeUList;
  interface TaggedValueClass;
  interface TaggedValue;
  typedef sequence<TaggedValue> TaggedValueSet;
  typedef sequence<TaggedValue> TaggedValueUList;
  interface ExtensionMechanismsPackage;

  interface StereotypeClass : DataTypes::EnumerationClass,
                   Core::GeneralizableElementClass {
    readonly attribute StereotypeUList all_of_type_stereotype;
    Stereotype create_stereotype (
      in Foundation::Name name,
      in VisibilityKind visibility,
      in boolean is_root,
      in boolean is_leaf,
      in boolean is_abstract,
      in Foundation::Name base_class)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Stereotype : StereotypeClass, DataTypes::Enumeration,
            Core::GeneralizableElement {
Foundation::Name base_class ()
  raises (Reflective::SemanticError);
void set_base_class (in Foundation::Name new_value)
  raises (Reflective::SemanticError);
TaggedValueSet required_tag ()
  raises (Reflective::SemanticError);
void set_required_tag (in TaggedValueSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_required_tag (in TaggedValue new_value)
  raises (Reflective::StructuralError);
void modify_required_tag (
  in TaggedValue old_value,
  in TaggedValue new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_required_tag (in TaggedValue old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
Core::ModelElementSet extended_element ()
  raises (Reflective::SemanticError);
void set_extended_element (in Core::ModelElementSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_extended_element (in Core::ModelElement new_value)
  raises (Reflective::StructuralError);
void modify_extended_element (
  in Core::ModelElement old_value,
  in Core::ModelElement new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_extended_element (in Core::ModelElement old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
Core::UmlConstraintSet stereotype_constraint ()
  raises (Reflective::SemanticError);
void set_stereotype_constraint (in Core::UmlConstraintSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
  void add_stereotype_constraint (in Core::UmlConstraint new_value)
    raises (Reflective::StructuralError);
  void modify_stereotype_constraint (
    in Core::UmlConstraint old_value,
    in Core::UmlConstraint new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_stereotype_constraint (in Core::UmlConstraint old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Stereotype

interface TaggedValueClass : Core::ElementClass {
  readonly attribute TaggedValueUList all_of_type_tagged_value;
  TaggedValue create_tagged_value (
    in Name tag,
    in string uml_value)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface TaggedValue : TaggedValueClass, Core::Element {
  Name tag ()
    raises (Reflective::SemanticError);
  void set_tag (in Name new_value)
    raises (Reflective::SemanticError);
  string uml_value ()
    raises (Reflective::SemanticError);
  void set_uml_value (in string new_value)
    raises (Reflective::SemanticError);
  ExtensionMechanisms::Stereotype stereotype ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_stereotype (in ExtensionMechanisms::Stereotype new_value)
    raises (Reflective::SemanticError);
  void unset_stereotype ()
    raises (Reflective::SemanticError);
  Core::ModelElement model_element ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_model_element (in Core::ModelElement new_value)
    raises (Reflective::SemanticError);
  void unset_model_element ()
    raises (Reflective::SemanticError);
```

```
}; // end of interface TaggedValue

struct ARequiredTagStereotypeLink {
  TaggedValue required_tag;
  Stereotype stereotype;
};
typedef sequence<ARequiredTagStereotypeLink>
  ARequiredTagStereotypeLinkSet;

interface ARequiredTagStereotype : Reflective::RefAssociation {
  ARequiredTagStereotypeLinkSet
    all_a_required_tag_stereotype_links();
  boolean exists (
    in TaggedValue required_tag,
    in Stereotype stereotype);
  Stereotype with_required_tag (
    in TaggedValue required_tag);
  TaggedValueSet with_stereotype (
    in Stereotype stereotype);
  void add (
    in TaggedValue required_tag,
    in Stereotype stereotype)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_required_tag (
    in TaggedValue required_tag,
    in Stereotype stereotype,
    in TaggedValue new_required_tag)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_stereotype (
    in TaggedValue required_tag,
    in Stereotype stereotype,
    in Stereotype new_stereotype)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in TaggedValue required_tag,
    in Stereotype stereotype)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ARequiredTagStereotype

struct AStereotypeExtendedElementLink {
  Stereotype stereotype;
  Core::ModelElement extended_element;
};
typedef sequence<AStereotypeExtendedElementLink>
  AStereotypeExtendedElementLinkSet;

interface AStereotypeExtendedElement : Reflective::RefAssociation {
  AStereotypeExtendedElementLinkSet
    all_a_stereotype_extended_element_links();
  boolean exists (
    in Stereotype stereotype,
    in Core::ModelElement extended_element);
  Core::ModelElementSet with_stereotype (
    in Stereotype stereotype);
  Stereotype with_extended_element (
    in Core::ModelElement extended_element);
  void add (
    in Stereotype stereotype,
    in Core::ModelElement extended_element)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_stereotype (
    in Stereotype stereotype,
    in Core::ModelElement extended_element,
    in Stereotype new_stereotype)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_extended_element (
    in Stereotype stereotype,
    in Core::ModelElement extended_element,
    in Core::ModelElement new_extended_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Stereotype stereotype,
    in Core::ModelElement extended_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AStereotypeExtendedElement

struct AConstrainedStereotypeStereotypeConstraintLink {
  Stereotype constrained_stereotype;
  Core::UmlConstraint stereotype_constraint;
};
```

```
typedef sequence<AConstrainedStereotypeStereotypeConstraintLink>
  AConstrainedStereotypeStereotypeConstraintLinkSet;

interface AConstrainedStereotypeStereotypeConstraint :
  Reflective::RefAssociation {
  AConstrainedStereotypeStereotypeConstraintLinkSet
    all_a_constrained_stereotype_stereotype_constraint_links();
  boolean exists (
    in Stereotype constrained_stereotype,
    in Core::UmlConstraint stereotype_constraint);
  Core::UmlConstraintSet with_constrained_stereotype (
    in Stereotype constrained_stereotype);
  Stereotype with_stereotype_constraint (
    in Core::UmlConstraint stereotype_constraint);
  void add (
    in Stereotype constrained_stereotype,
    in Core::UmlConstraint stereotype_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_constrained_stereotype (
    in Stereotype constrained_stereotype,
    in Core::UmlConstraint stereotype_constraint,
    in Stereotype new_constrained_stereotype)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_stereotype_constraint (
    in Stereotype constrained_stereotype,
    in Core::UmlConstraint stereotype_constraint,
    in Core::UmlConstraint new_stereotype_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Stereotype constrained_stereotype,
    in Core::UmlConstraint stereotype_constraint)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AConstrainedStereotypeStereotypeConstraint

struct AModelElementTaggedValueLink {
  Core::ModelElement model_element;
  TaggedValue tagged_value;
};
typedef sequence<AModelElementTaggedValueLink>
  AModelElementTaggedValueLinkSet;
```

```
interface AModelElementTaggedValue : Reflective::RefAssociation {
  AModelElementTaggedValueLinkSet
    all_a_model_element_tagged_value_links();
  boolean exists (
    in Core::ModelElement model_element,
    in TaggedValue tagged_value);
  TaggedValueSet with_model_element (
    in Core::ModelElement model_element);
  Core::ModelElement with_tagged_value (
    in TaggedValue tagged_value);
  void add (
    in Core::ModelElement model_element,
    in TaggedValue tagged_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_model_element (
    in Core::ModelElement model_element,
    in TaggedValue tagged_value,
    in Core::ModelElement new_model_element)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_tagged_value (
    in Core::ModelElement model_element,
    in TaggedValue tagged_value,
    in TaggedValue new_tagged_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Core::ModelElement model_element,
    in TaggedValue tagged_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AModelElementTaggedValue

interface ExtensionMechanismsPackageFactory {
  ExtensionMechanismsPackage
create_extension_mechanisms_package ()
    raises (Reflective::SemanticError);
};

interface ExtensionMechanismsPackage : Reflective::RefPackage {
  readonly attribute StereotypeClass stereotype_class_ref;
  readonly attribute TaggedValueClass tagged_value_class_ref;
```

```
      readonly attribute ARequiredTagStereotype
        a_required_tag_stereotype_class_ref;
      readonly attribute AStereotypeExtendedElement
        a_stereotype_extended_element_class_ref;
      readonly attribute AConstrainedStereotypeStereotypeConstraint
        a_constrained_stereotype_stereotype_constraint_class_ref;
      readonly attribute AModelElementTaggedValue
        a_model_element_tagged_value_class_ref;
    };
  }; // end of module ExtensionMechanisms

  interface FoundationPackageFactory {
    FoundationPackage create_foundation_package ()
      raises (Reflective::SemanticError);
  };

  interface FoundationPackage : Reflective::RefPackage {
    readonly attribute Core::CorePackage core_ref;
    readonly attribute DataTypes::DataTypesPackage data_types_ref;
    readonly attribute ExtensionMechanisms::ExtensionMechanismsPack-
age
      extension_mechanisms_ref;
  };
};
```

5.4.3 BehavioralElements

```
#include "Reflective.idl"
#include "Foundation.idl"

module BehavioralElements {
  typedef sequence<Foundation::Core::BehavioralFeature> BehavioralFea-
tureSet;
  typedef sequence<Foundation::Core::Parameter> ParameterSet;
  typedef sequence<Foundation::Core::Parameter> ParameterUList;
  typedef sequence<Foundation::Core::Feature> FeatureSet;
  typedef sequence<Foundation::Core::Classifier> ClassifierSet;
  typedef sequence<Foundation::Core::UmlAttribute> UmlAttributeSet;
  typedef sequence<Foundation::Core::ModelElement> ModelElementSet;
  interface BehavioralElementsPackage;

  module CommonBehavior {
    interface InstanceClass;
    interface Instance;
    typedef sequence<Instance> InstanceSet;
    typedef sequence<Instance> InstanceUList;
    interface SignalClass;
    interface Signal;
    typedef sequence<Signal> SignalSet;
typedef sequence<Signal> SignalUList;
interface CreateActionClass;
interface CreateAction;
typedef sequence<CreateAction> CreateActionSet;
typedef sequence<CreateAction> CreateActionUList;
interface DestroyActionClass;
interface DestroyAction;
typedef sequence<DestroyAction> DestroyActionUList;
interface UninterpretedActionClass;
interface UninterpretedAction;
typedef sequence<UninterpretedAction> UninterpretedActionUList;
interface ActionClass;
interface Action;
typedef sequence<Action> ActionSet;
typedef sequence<Action> ActionUList;
interface AttributeLinkClass;
interface AttributeLink;
typedef sequence<AttributeLink> AttributeLinkSet;
typedef sequence<AttributeLink> AttributeLinkUList;
interface LinkObjectClass;
interface LinkObject;
typedef sequence<LinkObject> LinkObjectUList;
interface UmlObjectClass;
interface UmlObject;
typedef sequence<UmlObject> UmlObjectUList;
interface DataValueClass;
interface DataValue;
typedef sequence<DataValue> DataValueUList;
interface CallActionClass;
interface CallAction;
typedef sequence<CallAction> CallActionSet;
typedef sequence<CallAction> CallActionUList;
interface SendActionClass;
interface SendAction;
typedef sequence<SendAction> SendActionSet;
typedef sequence<SendAction> SendActionUList;
interface ActionSequenceClass;
interface ActionSequence;
typedef sequence<ActionSequence> ActionSequenceUList;
interface ArgumentClass;
interface Argument;
typedef sequence<Argument> ArgumentUList;
interface ReceptionClass;
interface Reception;
typedef sequence<Reception> ReceptionSet;
typedef sequence<Reception> ReceptionUList;
interface LinkClass;
interface Link;
typedef sequence<Link> LinkSet;
typedef sequence<Link> LinkUList;
interface LinkEndClass;
interface LinkEnd;
typedef sequence<LinkEnd> LinkEndSet;
typedef sequence<LinkEnd> LinkEndUList;
interface CallClass;
interface Call;
typedef sequence<Call> CallUList;
interface ReturnActionClass;
interface ReturnAction;
typedef sequence<ReturnAction> ReturnActionUList;
interface TerminateActionClass;
interface TerminateAction;
typedef sequence<TerminateAction> TerminateActionUList;
interface StimulusClass;
interface Stimulus;
typedef sequence<Stimulus> StimulusSet;
typedef sequence<Stimulus> StimulusUList;
interface ActionInstanceClass;
interface ActionInstance;
typedef sequence<ActionInstance> ActionInstanceUList;
interface UmlExceptionClass;
interface UmlException;
typedef sequence<UmlException> UmlExceptionUList;
interface AssignmentActionClass;
interface AssignmentAction;
typedef sequence<AssignmentAction> AssignmentActionUList;
interface ComponentInstanceClass;
interface ComponentInstance;
typedef sequence<ComponentInstance> ComponentInstanceSet;
typedef sequence<ComponentInstance> ComponentInstanceUList;
interface NodeInstanceClass;
interface NodeInstance;
typedef sequence<NodeInstance> NodeInstanceUList;
interface CommonBehaviorPackage;

interface InstanceClass : Foundation::Core::ModelElementClass {
  readonly attribute InstanceUList all_of_type_instance;
  Instance create_an_instance (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Instance : InstanceClass, Foundation::Core::ModelElement {
  ClassifierSet classifier ()
    raises (Reflective::SemanticError);
  void set_classifier (in ClassifierSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
```

```
void add_classifier (in Foundation::Core::Classifier new_value)
  raises (Reflective::StructuralError);
void modify_classifier (
  in Foundation::Core::Classifier old_value,
  in Foundation::Core::Classifier new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_classifier (in Foundation::Core::Classifier old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
AttributeLinkSet attribute_link ()
  raises (Reflective::SemanticError);
void set_attribute_link (in AttributeLinkSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_attribute_link (in AttributeLink new_value)
  raises (Reflective::StructuralError);
void modify_attribute_link (
  in AttributeLink old_value,
  in AttributeLink new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_attribute_link (in AttributeLink old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
LinkEndSet link_end ()
  raises (Reflective::SemanticError);
void set_link_end (in LinkEndSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_link_end (in LinkEnd new_value)
  raises (Reflective::StructuralError);
void modify_link_end (
  in LinkEnd old_value,
  in LinkEnd new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_link_end (in LinkEnd old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
AttributeLinkSet slot ()
  raises (Reflective::SemanticError);
void set_slot (in AttributeLinkSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void unset_slot ()
  raises (Reflective::SemanticError);
void add_slot (in AttributeLink new_value)
  raises (Reflective::StructuralError);
void modify_slot (
  in AttributeLink old_value,
  in AttributeLink new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_slot (in AttributeLink old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
StimulusSet stimulus0 ()
  raises (Reflective::SemanticError);
void set_stimulus0 (in StimulusSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_stimulus0 (in CommonBehavior::Stimulus new_value)
  raises (Reflective::StructuralError);
void modify_stimulus0 (
  in CommonBehavior::Stimulus old_value,
  in CommonBehavior::Stimulus new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_stimulus0 (in CommonBehavior::Stimulus old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
StimulusSet stimulus1 ()
  raises (Reflective::SemanticError);
void set_stimulus1 (in StimulusSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
  void add_stimulus1 (in CommonBehavior::Stimulus new_value)
    raises (Reflective::StructuralError);
  void modify_stimulus1 (
    in CommonBehavior::Stimulus old_value,
    in CommonBehavior::Stimulus new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_stimulus1 (in CommonBehavior::Stimulus old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ComponentInstance component_instance ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_component_instance (in ComponentInstance new_value)
    raises (Reflective::SemanticError);
  void unset_component_instance ()
    raises (Reflective::SemanticError);
  StimulusSet stimulus2 ()
    raises (Reflective::SemanticError);
  void set_stimulus2 (in StimulusSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_stimulus2 (in CommonBehavior::Stimulus new_value)
    raises (Reflective::StructuralError);
  void modify_stimulus2 (
    in CommonBehavior::Stimulus old_value,
    in CommonBehavior::Stimulus new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_stimulus2 (in CommonBehavior::Stimulus old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Instance

interface SignalClass : Foundation::Core::ClassifierClass {
  readonly attribute SignalUList all_of_type_signal;
  Signal create_signal (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
      in boolean is_abstract)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
};

interface Signal : SignalClass, Foundation::Core::Classifier {
  ReceptionSet reception ()
    raises (Reflective::SemanticError);
  void set_reception (in ReceptionSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_reception ()
    raises (Reflective::SemanticError);
  void add_reception (in CommonBehavior::Reception new_value)
    raises (Reflective::StructuralError);
  void modify_reception (
    in CommonBehavior::Reception old_value,
    in CommonBehavior::Reception new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_reception (in CommonBehavior::Reception old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  ParameterUList parameter ()
    raises (Reflective::SemanticError);
  void set_parameter (in ParameterUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_parameter ()
    raises (Reflective::SemanticError);
  void add_parameter (in Foundation::Core::Parameter new_value)
    raises (Reflective::StructuralError);
  void add_parameter_before (
    in Foundation::Core::Parameter new_value,
    in Foundation::Core::Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameter (
    in Foundation::Core::Parameter old_value,
    in Foundation::Core::Parameter new_value)
    raises (
      Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_parameter (in Foundation::Core::Parameter old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    BehavioralFeatureSet uml_context ()
      raises (Reflective::SemanticError);
    void set_uml_context (in BehavioralFeatureSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_uml_context (in Foundation::Core::BehavioralFeature
new_value)
      raises (Reflective::StructuralError);
    void modify_uml_context (
      in Foundation::Core::BehavioralFeature old_value,
      in Foundation::Core::BehavioralFeature new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_uml_context (in Foundation::Core::BehavioralFeature
old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface Signal

  interface ActionClass : Foundation::Core::ModelElementClass {
    readonly attribute ActionUList all_of_type_action;
    Action create_action (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility,
      in Foundation::IterationExpression recurrence,
      in Foundation::ObjectSetExpression target,
      in boolean is_asynchronous,
      in Foundation::ActionExpression script)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Action : ActionClass, Foundation::Core::ModelElement {
    Foundation::IterationExpression recurrence ()
      raises (Reflective::SemanticError);
    void set_recurrence (in Foundation::IterationExpression new_value)
      raises (Reflective::SemanticError);
    Foundation::ObjectSetExpression target ()
  raises (Reflective::SemanticError);
void set_target (in Foundation::ObjectSetExpression new_value)
  raises (Reflective::SemanticError);
boolean is_asynchronous ()
  raises (Reflective::SemanticError);
void set_is_asynchronous (in boolean new_value)
  raises (Reflective::SemanticError);
Foundation::ActionExpression script ()
  raises (Reflective::SemanticError);
void set_script (in Foundation::ActionExpression new_value)
  raises (Reflective::SemanticError);
ArgumentUList actual_argument ()
  raises (Reflective::SemanticError);
void set_actual_argument (in ArgumentUList new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_actual_argument (in Argument new_value)
  raises (Reflective::StructuralError);
void add_actual_argument_before (
  in Argument new_value,
  in Argument before)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void modify_actual_argument (
  in Argument old_value,
  in Argument new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_actual_argument (in Argument old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ActionSequence action_sequence ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_action_sequence (in ActionSequence new_value)
  raises (Reflective::SemanticError);
void unset_action_sequence ()
  raises (Reflective::SemanticError);
StimulusSet stimulus ()
  raises (Reflective::SemanticError);
void set_stimulus (in StimulusSet new_value)
  raises (
    Reflective::StructuralError,
      Reflective::SemanticError);
  void add_stimulus (in CommonBehavior::Stimulus new_value)
    raises (Reflective::StructuralError);
  void modify_stimulus (
    in CommonBehavior::Stimulus old_value,
    in CommonBehavior::Stimulus new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_stimulus (in CommonBehavior::Stimulus old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Action

interface CreateActionClass : ActionClass {
  readonly attribute CreateActionUList all_of_type_create_action;
  CreateAction create_create_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface CreateAction : CreateActionClass, Action {
  Foundation::Core::Classifier instantiation ()
    raises (Reflective::SemanticError);
  void set_instantiation (in Foundation::Core::Classifier new_value)
    raises (Reflective::SemanticError);
}; // end of interface CreateAction

interface DestroyActionClass : ActionClass {
  readonly attribute DestroyActionUList all_of_type_destroy_action;
  DestroyAction create_destroy_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};
```

```
  interface DestroyAction : DestroyActionClass, Action {
  }; // end of interface DestroyAction

  interface UninterpretedActionClass : ActionClass {
    readonly attribute UninterpretedActionUList
all_of_type_uninterpreted_action;
    UninterpretedAction create_uninterpreted_action (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility,
      in Foundation::IterationExpression recurrence,
      in Foundation::ObjectSetExpression target,
      in boolean is_asynchronous,
      in Foundation::ActionExpression script)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface UninterpretedAction : UninterpretedActionClass, Action {
  }; // end of interface UninterpretedAction

  interface AttributeLinkClass : Foundation::Core::ModelElementClass {
    readonly attribute AttributeLinkUList all_of_type_attribute_link;
    AttributeLink create_attribute_link (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface AttributeLink : AttributeLinkClass, Foundation::Core::Mod-
elElement {
    Foundation::Core::UmlAttribute uml_attribute ()
      raises (Reflective::SemanticError);
    void set_uml_attribute (in Foundation::Core::UmlAttribute new_value)
      raises (Reflective::SemanticError);
    CommonBehavior::Instance uml_value ()
      raises (Reflective::SemanticError);
    void set_uml_value (in CommonBehavior::Instance new_value)
      raises (Reflective::SemanticError);
    CommonBehavior::Instance instance ()
      raises (Reflective::SemanticError);
    void set_instance (in CommonBehavior::Instance new_value)
      raises (Reflective::SemanticError);
  }; // end of interface AttributeLink

  interface UmlObjectClass : InstanceClass {
    readonly attribute UmlObjectUList all_of_type_uml_object;
    UmlObject create_uml_object (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlObject : UmlObjectClass, Instance {
}; // end of interface UmlObject

interface LinkClass : Foundation::Core::ModelElementClass {
  readonly attribute LinkUList all_of_type_link;
  Link create_link (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Link : LinkClass, Foundation::Core::ModelElement {
  Foundation::Core::Association association ()
    raises (Reflective::SemanticError);
  void set_association (in Foundation::Core::Association new_value)
    raises (Reflective::SemanticError);
  LinkEndSet connection ()
    raises (Reflective::SemanticError);
  void set_connection (in LinkEndSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_connection (in LinkEnd new_value)
    raises (Reflective::StructuralError);
  void modify_connection (
    in LinkEnd old_value,
    in LinkEnd new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_connection (in LinkEnd old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  StimulusSet stimulus ()
    raises (Reflective::SemanticError);
  void set_stimulus (in StimulusSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_stimulus (in CommonBehavior::Stimulus new_value)
    raises (Reflective::StructuralError);
  void modify_stimulus (
    in CommonBehavior::Stimulus old_value,
    in CommonBehavior::Stimulus new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_stimulus (in CommonBehavior::Stimulus old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Link

interface LinkObjectClass : UmlObjectClass, LinkClass {
  readonly attribute LinkObjectUList all_of_type_link_object;
  LinkObject create_link_object (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface LinkObject : LinkObjectClass, UmlObject, Link {
}; // end of interface LinkObject

interface DataValueClass : InstanceClass {
  readonly attribute DataValueUList all_of_type_data_value;
  DataValue create_data_value (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface DataValue : DataValueClass, Instance {
}; // end of interface DataValue

interface CallActionClass : ActionClass {
  readonly attribute CallActionUList all_of_type_call_action;
  CallAction create_call_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
```

```
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface CallAction : CallActionClass, Action {
      Foundation::Core::Operation operation ()
        raises (Reflective::SemanticError);
      void set_operation (in Foundation::Core::Operation new_value)
        raises (Reflective::SemanticError);
    }; // end of interface CallAction

    interface SendActionClass : ActionClass {
      readonly attribute SendActionUList all_of_type_send_action;
      SendAction create_send_action (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::IterationExpression recurrence,
        in Foundation::ObjectSetExpression target,
        in boolean is_asynchronous,
        in Foundation::ActionExpression script)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface SendAction : SendActionClass, Action {
    }; // end of interface SendAction

    interface ActionSequenceClass : CommonBehavior::ActionClass {
      readonly attribute ActionSequenceUList all_of_type_action_sequence;
      ActionSequence create_action_sequence (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::IterationExpression recurrence,
        in Foundation::ObjectSetExpression target,
        in boolean is_asynchronous,
        in Foundation::ActionExpression script)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface ActionSequence : ActionSequenceClass, CommonBehav-
ior::Action {
      ActionSet action ()
        raises (Reflective::SemanticError);
      void set_action (in ActionSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
```

```
  void add_action (in CommonBehavior::Action new_value)
    raises (Reflective::StructuralError);
  void modify_action (
    in CommonBehavior::Action old_value,
    in CommonBehavior::Action new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_action (in CommonBehavior::Action old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ActionSequence

interface ArgumentClass : Foundation::Core::ModelElementClass {
  readonly attribute ArgumentUList all_of_type_argument;
  Argument create_argument (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::Expression uml_value)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Argument : ArgumentClass, Foundation::Core::ModelElement {
  Foundation::Expression uml_value ()
    raises (Reflective::SemanticError);
  void set_uml_value (in Foundation::Expression new_value)
    raises (Reflective::SemanticError);
  CommonBehavior::Action action ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_action (in CommonBehavior::Action new_value)
    raises (Reflective::SemanticError);
  void unset_action ()
    raises (Reflective::SemanticError);
}; // end of interface Argument

interface ReceptionClass : Foundation::Core::BehavioralFeatureClass {
  readonly attribute ReceptionUList all_of_type_reception;
  Reception create_reception (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::ScopeKind owner_scope,
    in boolean is_query,
    in boolean is_polymorphic,
    in string specification)
```

```
            raises (
              Reflective::SemanticError,
              Reflective::ConstraintError);
    };

    interface Reception : ReceptionClass, Foundation::Core::BehavioralFea-
ture {
          boolean is_polymorphic ()
            raises (Reflective::SemanticError);
          void set_is_polymorphic (in boolean new_value)
            raises (Reflective::SemanticError);
          string specification ()
            raises (Reflective::SemanticError);
          void set_specification (in string new_value)
            raises (Reflective::SemanticError);
          CommonBehavior::Signal signal ()
            raises (Reflective::SemanticError);
          void set_signal (in CommonBehavior::Signal new_value)
            raises (Reflective::SemanticError);
    }; // end of interface Reception

    interface LinkEndClass : Foundation::Core::ModelElementClass {
          readonly attribute LinkEndUList all_of_type_link_end;
          LinkEnd create_link_end (
            in Foundation::Name name,
            in Foundation::VisibilityKind visibility)
            raises (
              Reflective::SemanticError,
              Reflective::ConstraintError);
    };

    interface LinkEnd : LinkEndClass, Foundation::Core::ModelElement {
          CommonBehavior::Instance instance ()
            raises (Reflective::SemanticError);
          void set_instance (in CommonBehavior::Instance new_value)
            raises (Reflective::SemanticError);
          CommonBehavior::Link link ()
            raises (Reflective::SemanticError);
          void set_link (in CommonBehavior::Link new_value)
            raises (Reflective::SemanticError);
          Foundation::Core::AssociationEnd association_end ()
            raises (Reflective::SemanticError);
          void set_association_end (in Foundation::Core::AssociationEnd
new_value)
            raises (Reflective::SemanticError);
    }; // end of interface LinkEnd

    interface CallClass : Reflective::RefObject {
          readonly attribute CallUList all_of_type_call;
          Call create_call ()
            raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Call : CallClass {
}; // end of interface Call

interface ReturnActionClass : ActionClass {
  readonly attribute ReturnActionUList all_of_type_return_action;
  ReturnAction create_return_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface ReturnAction : ReturnActionClass, Action {
}; // end of interface ReturnAction

interface TerminateActionClass : ActionClass {
  readonly attribute TerminateActionUList all_of_type_terminate_action;
  TerminateAction create_terminate_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface TerminateAction : TerminateActionClass, Action {
}; // end of interface TerminateAction

interface StimulusClass : Foundation::Core::ModelElementClass {
  readonly attribute StimulusUList all_of_type_stimulus;
  Stimulus create_stimulus (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};
```

```
interface Stimulus : StimulusClass, Foundation::Core::ModelElement {
  InstanceSet argument ()
    raises (Reflective::SemanticError);
  void set_argument (in InstanceSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_argument (in Instance new_value)
    raises (Reflective::StructuralError);
  void modify_argument (
    in Instance old_value,
    in Instance new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_argument (in Instance old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  Instance sender ()
    raises (Reflective::SemanticError);
  void set_sender (in Instance new_value)
    raises (Reflective::SemanticError);
  Instance receiver ()
    raises (Reflective::SemanticError);
  void set_receiver (in Instance new_value)
    raises (Reflective::SemanticError);
  Link communication_link ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_communication_link (in Link new_value)
    raises (Reflective::SemanticError);
  void unset_communication_link ()
    raises (Reflective::SemanticError);
  Action dispatch_action ()
    raises (Reflective::SemanticError);
  void set_dispatch_action (in Action new_value)
    raises (Reflective::SemanticError);
}; // end of interface Stimulus

interface ActionInstanceClass : Reflective::RefObject {
  readonly attribute ActionInstanceUList all_of_type_action_instance;
  ActionInstance create_action_instance ()
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};
```

```
interface ActionInstance : ActionInstanceClass {
}; // end of interface ActionInstance

interface UmlExceptionClass : SignalClass {
  readonly attribute UmlExceptionUList all_of_type_uml_exception;
  UmlException create_uml_exception (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in boolean is_root,
    in boolean is_leaf,
    in boolean is_abstract)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface UmlException : UmlExceptionClass, Signal {
}; // end of interface UmlException

interface AssignmentActionClass : ActionClass {
  readonly attribute AssignmentActionUList
all_of_type_assignment_action;
  AssignmentAction create_assignment_action (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::IterationExpression recurrence,
    in Foundation::ObjectSetExpression target,
    in boolean is_asynchronous,
    in Foundation::ActionExpression script)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface AssignmentAction : AssignmentActionClass, Action {
}; // end of interface AssignmentAction

interface ComponentInstanceClass : InstanceClass {
  readonly attribute ComponentInstanceUList
all_of_type_component_instance;
  ComponentInstance create_component_instance (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface ComponentInstance : ComponentInstanceClass, Instance {
  NodeInstance node_instance ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_node_instance (in NodeInstance new_value)
    raises (Reflective::SemanticError);
  void unset_node_instance ()
    raises (Reflective::SemanticError);
  InstanceSet resident ()
    raises (Reflective::SemanticError);
  void set_resident (in InstanceSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_resident (in Instance new_value)
    raises (Reflective::StructuralError);
  void modify_resident (
    in Instance old_value,
    in Instance new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_resident (in Instance old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ComponentInstance

interface NodeInstanceClass : InstanceClass {
  readonly attribute NodeInstanceUList all_of_type_node_instance;
  NodeInstance create_node_instance (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface NodeInstance : NodeInstanceClass, Instance {
  ComponentInstanceSet resident ()
    raises (Reflective::SemanticError);
  void set_resident (in ComponentInstanceSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_resident (in ComponentInstance new_value)
    raises (Reflective::StructuralError);
  void modify_resident (
    in ComponentInstance old_value,
    in ComponentInstance new_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  void remove_resident (in ComponentInstance old_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
}; // end of interface NodeInstance

struct AInstanceClassifierLink {
  Instance instance;
  Foundation::Core::Classifier classifier;
};
typedef sequence<AInstanceClassifierLink> AInstanceClassifierLinkSet;

interface AInstanceClassifier : Reflective::RefAssociation {
  AInstanceClassifierLinkSet all_a_instance_classifier_links();
  boolean exists (
    in Instance instance,
    in Foundation::Core::Classifier classifier);
  ClassifierSet with_instance (
    in Instance instance);
  InstanceSet with_classifier (
    in Foundation::Core::Classifier classifier);
  void add (
    in Instance instance,
    in Foundation::Core::Classifier classifier)
    raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
  void modify_instance (
    in Instance instance,
    in Foundation::Core::Classifier classifier,
    in Instance new_instance)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  void modify_classifier (
    in Instance instance,
    in Foundation::Core::Classifier classifier,
    in Foundation::Core::Classifier new_classifier)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  void remove (
    in Instance instance,
    in Foundation::Core::Classifier classifier)
    raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    }; // end of interface AInstanceClassifier

    struct AActualArgumentActionLink {
      Argument actual_argument;
      Action action;
    };
    typedef sequence<AActualArgumentActionLink> AActualArgumentAc-
tionLinkSet;

    interface AActualArgumentAction : Reflective::RefAssociation {
      AActualArgumentActionLinkSet all_a_actual_argument_action_links();
      boolean exists (
        in Argument actual_argument,
        in Action action);
      Action with_actual_argument (
        in Argument actual_argument);
      ArgumentUList with_action (
        in Action action);
      void add (
        in Argument actual_argument,
        in Action action)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_before_actual_argument (
        in Argument actual_argument,
        in Action action,
        in Argument before)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_actual_argument (
        in Argument actual_argument,
        in Action action,
        in Argument new_actual_argument)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_action (
        in Argument actual_argument,
        in Action action,
        in Action new_action)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    void remove (
      in Argument actual_argument,
      in Action action)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AActualArgumentAction

  struct ACreateActionInstantiationLink {
    CreateAction create_action;
    Foundation::Core::Classifier instantiation;
  };
  typedef sequence<ACreateActionInstantiationLink> ACreateActionIn-
stantiationLinkSet;

  interface ACreateActionInstantiation : Reflective::RefAssociation {
    ACreateActionInstantiationLinkSet
all_a_create_action_instantiation_links();
    boolean exists (
      in CreateAction create_action,
      in Foundation::Core::Classifier instantiation);
    Foundation::Core::Classifier with_create_action (
      in CreateAction create_action);
    CreateActionSet with_instantiation (
      in Foundation::Core::Classifier instantiation);
    void add (
      in CreateAction create_action,
      in Foundation::Core::Classifier instantiation)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_create_action (
      in CreateAction create_action,
      in Foundation::Core::Classifier instantiation,
      in CreateAction new_create_action)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_instantiation (
      in CreateAction create_action,
      in Foundation::Core::Classifier instantiation,
      in Foundation::Core::Classifier new_instantiation)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in CreateAction create_action,
      in Foundation::Core::Classifier instantiation)
```

```
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ACreateActionInstantiation

    struct AAttributeLinkAttributeLink {
      AttributeLink attribute_link;
      Foundation::Core::UmlAttribute uml_attribute;
    };
    typedef sequence<AAttributeLinkAttributeLink> AAttributeLinkAt-
tributeLinkSet;

    interface AAttributeLinkAttribute : Reflective::RefAssociation {
      AAttributeLinkAttributeLinkSet all_a_attribute_link_attribute_links();
      boolean exists (
        in AttributeLink attribute_link,
        in Foundation::Core::UmlAttribute uml_attribute);
      Foundation::Core::UmlAttribute with_attribute_link (
        in AttributeLink attribute_link);
      AttributeLinkSet with_uml_attribute (
        in Foundation::Core::UmlAttribute uml_attribute);
      void add (
        in AttributeLink attribute_link,
        in Foundation::Core::UmlAttribute uml_attribute)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_attribute_link (
        in AttributeLink attribute_link,
        in Foundation::Core::UmlAttribute uml_attribute,
        in AttributeLink new_attribute_link)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_uml_attribute (
        in AttributeLink attribute_link,
        in Foundation::Core::UmlAttribute uml_attribute,
        in Foundation::Core::UmlAttribute new_uml_attribute)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in AttributeLink attribute_link,
        in Foundation::Core::UmlAttribute uml_attribute)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
```

```
  }; // end of interface AAttributeLinkAttribute

  struct AAttributeLinkValueLink {
    AttributeLink attribute_link;
    Instance uml_value;
  };
  typedef sequence<AAttributeLinkValueLink> AAttributeLinkValueLink-
Set;

  interface AAttributeLinkValue : Reflective::RefAssociation {
    AAttributeLinkValueLinkSet all_a_attribute_link_value_links();
    boolean exists (
      in AttributeLink attribute_link,
      in Instance uml_value);
    Instance with_attribute_link (
      in AttributeLink attribute_link);
    AttributeLinkSet with_uml_value (
      in Instance uml_value);
    void add (
      in AttributeLink attribute_link,
      in Instance uml_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_attribute_link (
      in AttributeLink attribute_link,
      in Instance uml_value,
      in AttributeLink new_attribute_link)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_uml_value (
      in AttributeLink attribute_link,
      in Instance uml_value,
      in Instance new_uml_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in AttributeLink attribute_link,
      in Instance uml_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AAttributeLinkValue

  struct AInstanceLinkEndLink {
    Instance instance;
  LinkEnd link_end;
};
typedef sequence<AInstanceLinkEndLink> AInstanceLinkEndLinkSet;

interface AInstanceLinkEnd : Reflective::RefAssociation {
  AInstanceLinkEndLinkSet all_a_instance_link_end_links();
  boolean exists (
    in Instance instance,
    in LinkEnd link_end);
  LinkEndSet with_instance (
    in Instance instance);
  Instance with_link_end (
    in LinkEnd link_end);
  void add (
    in Instance instance,
    in LinkEnd link_end)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_instance (
    in Instance instance,
    in LinkEnd link_end,
    in Instance new_instance)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_link_end (
    in Instance instance,
    in LinkEnd link_end,
    in LinkEnd new_link_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Instance instance,
    in LinkEnd link_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AInstanceLinkEnd

struct ASignalReceptionLink {
  Signal signal;
  Reception reception;
};
typedef sequence<ASignalReceptionLink> ASignalReceptionLinkSet;

interface ASignalReception : Reflective::RefAssociation {
  ASignalReceptionLinkSet all_a_signal_reception_links();
  boolean exists (
    in Signal signal,
    in Reception reception);
  ReceptionSet with_signal (
    in Signal signal);
  Signal with_reception (
    in Reception reception);
  void add (
    in Signal signal,
    in Reception reception)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_signal (
    in Signal signal,
    in Reception reception,
    in Signal new_signal)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_reception (
    in Signal signal,
    in Reception reception,
    in Reception new_reception)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Signal signal,
    in Reception reception)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ASignalReception

struct ASignalParameterLink {
  Signal signal;
  Foundation::Core::Parameter parameter;
};
typedef sequence<ASignalParameterLink> ASignalParameterLinkSet;

interface ASignalParameter : Reflective::RefAssociation {
  ASignalParameterLinkSet all_a_signal_parameter_links();
  boolean exists (
    in Signal signal,
    in Foundation::Core::Parameter parameter);
  ParameterUList with_signal (
    in Signal signal);
  Signal with_parameter (
    in Foundation::Core::Parameter parameter);
  void add (
    in Signal signal,
    in Foundation::Core::Parameter parameter)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_parameter (
    in Signal signal,
    in Foundation::Core::Parameter parameter,
    in Foundation::Core::Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_signal (
    in Signal signal,
    in Foundation::Core::Parameter parameter,
    in Signal new_signal)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameter (
    in Signal signal,
    in Foundation::Core::Parameter parameter,
    in Foundation::Core::Parameter new_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Signal signal,
    in Foundation::Core::Parameter parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ASignalParameter

struct ASlotInstanceLink {
  AttributeLink slot;
  Instance instance;
};
typedef sequence<ASlotInstanceLink> ASlotInstanceLinkSet;

interface ASlotInstance : Reflective::RefAssociation {
  ASlotInstanceLinkSet all_a_slot_instance_links();
  boolean exists (
      in AttributeLink slot,
      in Instance instance);
    Instance with_slot (
      in AttributeLink slot);
    AttributeLinkSet with_instance (
      in Instance instance);
    void add (
      in AttributeLink slot,
      in Instance instance)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_slot (
      in AttributeLink slot,
      in Instance instance,
      in AttributeLink new_slot)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_instance (
      in AttributeLink slot,
      in Instance instance,
      in Instance new_instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in AttributeLink slot,
      in Instance instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ASlotInstance

  struct AArgumentStimulusLink {
    Instance argument;
    Stimulus stimulus;
  };
  typedef sequence<AArgumentStimulusLink> AArgumentStimulusLink-
Set;

  interface AArgumentStimulus : Reflective::RefAssociation {
    AArgumentStimulusLinkSet all_a_argument_stimulus_links();
    boolean exists (
      in Instance argument,
      in Stimulus stimulus);
    StimulusSet with_argument (
      in Instance argument);
    InstanceSet with_stimulus (
      in Stimulus stimulus);
    void add (
      in Instance argument,
      in Stimulus stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_argument (
      in Instance argument,
      in Stimulus stimulus,
      in Instance new_argument)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_stimulus (
      in Instance argument,
      in Stimulus stimulus,
      in Stimulus new_stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Instance argument,
      in Stimulus stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AArgumentStimulus

  struct AContextRaisedSignalLink {
    Foundation::Core::BehavioralFeature uml_context;
    Signal raised_signal;
  };
  typedef sequence<AContextRaisedSignalLink> AContextRaisedSignal-
LinkSet;

  interface AContextRaisedSignal : Reflective::RefAssociation {
    AContextRaisedSignalLinkSet all_a_context_raised_signal_links();
    boolean exists (
      in Foundation::Core::BehavioralFeature uml_context,
      in Signal raised_signal);
    SignalSet with_uml_context (
      in Foundation::Core::BehavioralFeature uml_context);
    BehavioralFeatureSet with_raised_signal (
      in Signal raised_signal);
    void add (
      in Foundation::Core::BehavioralFeature uml_context,
        in Signal raised_signal)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_uml_context (
        in Foundation::Core::BehavioralFeature uml_context,
        in Signal raised_signal,
        in Foundation::Core::BehavioralFeature new_uml_context)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_raised_signal (
        in Foundation::Core::BehavioralFeature uml_context,
        in Signal raised_signal,
        in Signal new_raised_signal)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in Foundation::Core::BehavioralFeature uml_context,
        in Signal raised_signal)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AContextRaisedSignal

    struct AAssociationLinkLink {
      Foundation::Core::Association association;
      Link link;
    };
    typedef sequence<AAssociationLinkLink> AAssociationLinkLinkSet;

    interface AAssociationLink : Reflective::RefAssociation {
      AAssociationLinkLinkSet all_a_association_link_links();
      boolean exists (
        in Foundation::Core::Association association,
        in Link link);
      LinkSet with_association (
        in Foundation::Core::Association association);
      Foundation::Core::Association with_link (
        in Link link);
      void add (
        in Foundation::Core::Association association,
        in Link link)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_association (
      in Foundation::Core::Association association,
      in Link link,
      in Foundation::Core::Association new_association)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_a_link (
      in Foundation::Core::Association association,
      in Link link,
      in Link new_link)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Foundation::Core::Association association,
      in Link link)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AAssociationLink

  struct ALinkConnectionLink {
    Link link;
    LinkEnd connection;
  };
  typedef sequence<ALinkConnectionLink> ALinkConnectionLinkSet;

  interface ALinkConnection : Reflective::RefAssociation {
    ALinkConnectionLinkSet all_a_link_connection_links();
    boolean exists (
      in Link link,
      in LinkEnd connection);
    LinkEndSet with_link (
      in Link link);
    Link with_connection (
      in LinkEnd connection);
    void add (
      in Link link,
      in LinkEnd connection)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_a_link (
      in Link link,
      in LinkEnd connection,
      in Link new_link)
      raises (
        Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
        void modify_connection (
            in Link link,
            in LinkEnd connection,
            in LinkEnd new_connection)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
        void remove (
            in Link link,
            in LinkEnd connection)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
    }; // end of interface ALinkConnection

    struct AAssociationEndLinkEndLink {
        Foundation::Core::AssociationEnd association_end;
        LinkEnd link_end;
    };
    typedef sequence<AAssociationEndLinkEndLink> AAssociation-
EndLinkEndLinkSet;

    interface AAssociationEndLinkEnd : Reflective::RefAssociation {
        AAssociationEndLinkEndLinkSet
all_a_association_end_link_end_links();
        boolean exists (
            in Foundation::Core::AssociationEnd association_end,
            in LinkEnd link_end);
        LinkEndSet with_association_end (
            in Foundation::Core::AssociationEnd association_end);
        Foundation::Core::AssociationEnd with_link_end (
            in LinkEnd link_end);
        void add (
            in Foundation::Core::AssociationEnd association_end,
            in LinkEnd link_end)
            raises (
                Reflective::StructuralError,
                Reflective::SemanticError);
        void modify_association_end (
            in Foundation::Core::AssociationEnd association_end,
            in LinkEnd link_end,
            in Foundation::Core::AssociationEnd new_association_end)
            raises (
                Reflective::StructuralError,
                Reflective::NotFound,
                Reflective::SemanticError);
        void modify_link_end (
    in Foundation::Core::AssociationEnd association_end,
    in LinkEnd link_end,
    in LinkEnd new_link_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Foundation::Core::AssociationEnd association_end,
    in LinkEnd link_end)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AAssociationEndLinkEnd

struct AStimulusSenderLink {
  Stimulus stimulus;
  Instance sender;
};
typedef sequence<AStimulusSenderLink> AStimulusSenderLinkSet;

interface AStimulusSender : Reflective::RefAssociation {
  AStimulusSenderLinkSet all_a_stimulus_sender_links();
  boolean exists (
    in Stimulus stimulus,
    in Instance sender);
  Instance with_stimulus (
    in Stimulus stimulus);
  StimulusSet with_sender (
    in Instance sender);
  void add (
    in Stimulus stimulus,
    in Instance sender)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_stimulus (
    in Stimulus stimulus,
    in Instance sender,
    in Stimulus new_stimulus)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_sender (
    in Stimulus stimulus,
    in Instance sender,
    in Instance new_sender)
    raises (
      Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void remove (
         in Stimulus stimulus,
         in Instance sender)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
   }; // end of interface AStimulusSender

   struct ACallActionOperationLink {
      CallAction call_action;
      Foundation::Core::Operation operation;
   };
   typedef sequence<ACallActionOperationLink> ACallActionOperation-
LinkSet;

   interface ACallActionOperation : Reflective::RefAssociation {
      ACallActionOperationLinkSet all_a_call_action_operation_links();
      boolean exists (
         in CallAction call_action,
         in Foundation::Core::Operation operation);
      Foundation::Core::Operation with_call_action (
         in CallAction call_action);
      CallActionSet with_operation (
         in Foundation::Core::Operation operation);
      void add (
         in CallAction call_action,
         in Foundation::Core::Operation operation)
         raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
      void modify_call_action (
         in CallAction call_action,
         in Foundation::Core::Operation operation,
         in CallAction new_call_action)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void modify_operation (
         in CallAction call_action,
         in Foundation::Core::Operation operation,
         in Foundation::Core::Operation new_operation)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void remove (
         in CallAction call_action,
      in Foundation::Core::Operation operation)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ACallActionOperation

  struct AActionSequenceActionLink {
    ActionSequence action_sequence;
    Action action;
  };
  typedef sequence<AActionSequenceActionLink> AActionSequenceAc-
tionLinkSet;

  interface AActionSequenceAction : Reflective::RefAssociation {
   AActionSequenceActionLinkSet all_a_action_sequence_action_links();
    boolean exists (
      in ActionSequence action_sequence,
      in Action action);
    ActionSet with_action_sequence (
      in ActionSequence action_sequence);
    ActionSequence with_action (
      in Action action);
    void add (
      in ActionSequence action_sequence,
      in Action action)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_action_sequence (
      in ActionSequence action_sequence,
      in Action action,
      in ActionSequence new_action_sequence)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_action (
      in ActionSequence action_sequence,
      in Action action,
      in Action new_action)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ActionSequence action_sequence,
      in Action action)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
      Reflective::SemanticError);
  }; // end of interface AActionSequenceAction

  struct AResidentNodeInstanceLink {
    ComponentInstance resident;
    NodeInstance node_instance;
  };
  typedef sequence<AResidentNodeInstanceLink> AResidentNodeInstan-
ceLinkSet;

  interface AResidentNodeInstance : Reflective::RefAssociation {
    AResidentNodeInstanceLinkSet all_a_resident_node_instance_links();
    boolean exists (
      in ComponentInstance resident,
      in NodeInstance node_instance);
    NodeInstance with_resident (
      in ComponentInstance resident);
    ComponentInstanceSet with_node_instance (
      in NodeInstance node_instance);
    void add (
      in ComponentInstance resident,
      in NodeInstance node_instance)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_resident (
      in ComponentInstance resident,
      in NodeInstance node_instance,
      in ComponentInstance new_resident)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_node_instance (
      in ComponentInstance resident,
      in NodeInstance node_instance,
      in NodeInstance new_node_instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ComponentInstance resident,
      in NodeInstance node_instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AResidentNodeInstance

  struct AResidentComponentInstanceLink {
    Instance resident;
    ComponentInstance component_instance;
  };
  typedef sequence<AResidentComponentInstanceLink> AResidentCom-
ponentInstanceLinkSet;

  interface AResidentComponentInstance : Reflective::RefAssociation {
    AResidentComponentInstanceLinkSet
all_a_resident_component_instance_links();
    boolean exists (
      in Instance resident,
      in ComponentInstance component_instance);
    ComponentInstance with_resident (
      in Instance resident);
    InstanceSet with_component_instance (
      in ComponentInstance component_instance);
    void add (
      in Instance resident,
      in ComponentInstance component_instance)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_resident (
      in Instance resident,
      in ComponentInstance component_instance,
      in Instance new_resident)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_component_instance (
      in Instance resident,
      in ComponentInstance component_instance,
      in ComponentInstance new_component_instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Instance resident,
      in ComponentInstance component_instance)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AResidentComponentInstance

  struct AReceiverStimulusLink {
    Instance receiver;
    Stimulus stimulus;
  };
```

```
typedef sequence<AReceiverStimulusLink> AReceiverStimulusLinkSet;

interface AReceiverStimulus : Reflective::RefAssociation {
  AReceiverStimulusLinkSet all_a_receiver_stimulus_links();
  boolean exists (
    in Instance receiver,
    in Stimulus stimulus);
  StimulusSet with_receiver (
    in Instance receiver);
  Instance with_stimulus (
    in Stimulus stimulus);
  void add (
    in Instance receiver,
    in Stimulus stimulus)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_receiver (
    in Instance receiver,
    in Stimulus stimulus,
    in Instance new_receiver)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_stimulus (
    in Instance receiver,
    in Stimulus stimulus,
    in Stimulus new_stimulus)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Instance receiver,
    in Stimulus stimulus)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AReceiverStimulus

struct AStimulusCommunicationLinkLink {
  Stimulus stimulus;
  Link communication_link;
};
typedef sequence<AStimulusCommunicationLinkLink> AStimulusCom-
municationLinkLinkSet;

interface AStimulusCommunicationLink : Reflective::RefAssociation {
  AStimulusCommunicationLinkLinkSet
```

```
all_a_stimulus_communication_link_links();
      boolean exists (
        in Stimulus stimulus,
        in Link communication_link);
      Link with_stimulus (
        in Stimulus stimulus);
      StimulusSet with_communication_link (
        in Link communication_link);
      void add (
        in Stimulus stimulus,
        in Link communication_link)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_stimulus (
        in Stimulus stimulus,
        in Link communication_link,
        in Stimulus new_stimulus)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_communication_link (
        in Stimulus stimulus,
        in Link communication_link,
        in Link new_communication_link)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in Stimulus stimulus,
        in Link communication_link)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AStimulusCommunicationLink

    struct ADispatchActionStimulusLink {
      Action dispatch_action;
      Stimulus stimulus;
    };
    typedef sequence<ADispatchActionStimulusLink> ADispatchAction-
StimulusLinkSet;

    interface ADispatchActionStimulus : Reflective::RefAssociation {
      ADispatchActionStimulusLinkSet
all_a_dispatch_action_stimulus_links();
      boolean exists (
        in Action dispatch_action,
      in Stimulus stimulus);
    StimulusSet with_dispatch_action (
      in Action dispatch_action);
    Action with_stimulus (
      in Stimulus stimulus);
    void add (
      in Action dispatch_action,
      in Stimulus stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_dispatch_action (
      in Action dispatch_action,
      in Stimulus stimulus,
      in Action new_dispatch_action)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_stimulus (
      in Action dispatch_action,
      in Stimulus stimulus,
      in Stimulus new_stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Action dispatch_action,
      in Stimulus stimulus)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ADispatchActionStimulus

  interface CommonBehaviorPackageFactory {
    CommonBehaviorPackage create_common_behavior_package ()
      raises (Reflective::SemanticError);
  };

  interface CommonBehaviorPackage : Reflective::RefPackage {
    readonly attribute InstanceClass instance_class_ref;
    readonly attribute SignalClass signal_class_ref;
    readonly attribute CreateActionClass create_action_class_ref;
    readonly attribute DestroyActionClass destroy_action_class_ref;
    readonly attribute UninterpretedActionClass
uninterpreted_action_class_ref;
    readonly attribute ActionClass action_class_ref;
    readonly attribute AttributeLinkClass attribute_link_class_ref;
    readonly attribute LinkObjectClass link_object_class_ref;
```

```
          readonly attribute UmlObjectClass uml_object_class_ref;
          readonly attribute DataValueClass data_value_class_ref;
          readonly attribute CallActionClass call_action_class_ref;
          readonly attribute SendActionClass send_action_class_ref;
          readonly attribute ActionSequenceClass action_sequence_class_ref;
          readonly attribute ArgumentClass argument_class_ref;
          readonly attribute ReceptionClass reception_class_ref;
          readonly attribute LinkClass link_class_ref;
          readonly attribute LinkEndClass link_end_class_ref;
          readonly attribute CallClass call_class_ref;
          readonly attribute ReturnActionClass return_action_class_ref;
          readonly attribute TerminateActionClass terminate_action_class_ref;
          readonly attribute StimulusClass stimulus_class_ref;
          readonly attribute ActionInstanceClass action_instance_class_ref;
          readonly attribute UmlExceptionClass uml_exception_class_ref;
          readonly attribute AssignmentActionClass
assignment_action_class_ref;
          readonly attribute ComponentInstanceClass
component_instance_class_ref;
          readonly attribute NodeInstanceClass node_instance_class_ref;
          readonly attribute AInstanceClassifier a_instance_classifier_class_ref;
          readonly attribute AActualArgumentAction
a_actual_argument_action_class_ref;
          readonly attribute ACreateActionInstantiation
a_create_action_instantiation_class_ref;
          readonly attribute AAttributeLinkAttribute
a_attribute_link_attribute_class_ref;
          readonly attribute AAttributeLinkValue
a_attribute_link_value_class_ref;
          readonly attribute AInstanceLinkEnd a_instance_link_end_class_ref;
          readonly attribute ASignalReception a_signal_reception_class_ref;
          readonly attribute ASignalParameter a_signal_parameter_class_ref;
          readonly attribute ASlotInstance a_slot_instance_class_ref;
          readonly attribute AArgumentStimulus
a_argument_stimulus_class_ref;
          readonly attribute AContextRaisedSignal
a_context_raised_signal_class_ref;
          readonly attribute AAssociationLink a_association_link_class_ref;
          readonly attribute ALinkConnection a_link_connection_class_ref;
          readonly attribute AAssociationEndLinkEnd
a_association_end_link_end_class_ref;
          readonly attribute AStimulusSender a_stimulus_sender_class_ref;
          readonly attribute ACallActionOperation
a_call_action_operation_class_ref;
          readonly attribute AActionSequenceAction
a_action_sequence_action_class_ref;
          readonly attribute AResidentNodeInstance
a_resident_node_instance_class_ref;
          readonly attribute AResidentComponentInstance
a_resident_component_instance_class_ref;
          readonly attribute AReceiverStimulus a_receiver_stimulus_class_ref;
```

```
         readonly attribute AStimulusCommunicationLink
a_stimulus_communication_link_class_ref;
         readonly attribute ADispatchActionStimulus
a_dispatch_action_stimulus_class_ref;
      };
   }; // end of module CommonBehavior

   module UseCases {
      interface UseCaseClass;
      interface UseCase;
      typedef sequence<UseCase> UseCaseUList;
      interface ActorClass;
      interface Actor;
      typedef sequence<Actor> ActorUList;
      interface UseCaseInstanceClass;
      interface UseCaseInstance;
      typedef sequence<UseCaseInstance> UseCaseInstanceUList;
      interface ExtendClass;
      interface Extend;
      typedef sequence<Extend> ExtendSet;
      typedef sequence<Extend> ExtendUList;
      interface IncludeClass;
      interface Include;
      typedef sequence<Include> IncludeSet;
      typedef sequence<Include> IncludeUList;
      interface ExtensionPointClass;
      interface ExtensionPoint;
      typedef sequence<ExtensionPoint> ExtensionPointSet;
      typedef sequence<ExtensionPoint> ExtensionPointUList;
      interface UseCasesPackage;

      interface UseCaseClass : Foundation::Core::ClassifierClass {
         readonly attribute UseCaseUList all_of_type_use_case;
         UseCase create_use_case (
            in Foundation::Name name,
            in Foundation::VisibilityKind visibility,
            in boolean is_root,
            in boolean is_leaf,
            in boolean is_abstract)
            raises (
               Reflective::SemanticError,
               Reflective::ConstraintError);
      };

      interface UseCase : UseCaseClass, Foundation::Core::Classifier {
         ExtendSet extension ()
            raises (Reflective::SemanticError);
         void set_extension (in ExtendSet new_value)
            raises (
               Reflective::StructuralError,
               Reflective::SemanticError);
```

```
void add_extension (in UseCases::Extend new_value)
  raises (Reflective::StructuralError);
void modify_extension (
  in UseCases::Extend old_value,
  in UseCases::Extend new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_extension (in UseCases::Extend old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ExtendSet extend ()
  raises (Reflective::SemanticError);
void set_extend (in ExtendSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_extend (in UseCases::Extend new_value)
  raises (Reflective::StructuralError);
void modify_extend (
  in UseCases::Extend old_value,
  in UseCases::Extend new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_extend (in UseCases::Extend old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
IncludeSet include ()
  raises (Reflective::SemanticError);
void set_include (in IncludeSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_include (in UseCases::Include new_value)
  raises (Reflective::StructuralError);
void modify_include (
  in UseCases::Include old_value,
  in UseCases::Include new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_include (in UseCases::Include old_value)
  raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  IncludeSet inclusion ()
    raises (Reflective::SemanticError);
  void set_inclusion (in IncludeSet new_value)
    raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
  void add_inclusion (in UseCases::Include new_value)
    raises (Reflective::StructuralError);
  void modify_inclusion (
    in UseCases::Include old_value,
    in UseCases::Include new_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  void remove_inclusion (in UseCases::Include old_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  ExtensionPointSet extension_point ()
    raises (Reflective::SemanticError);
  void set_extension_point (in ExtensionPointSet new_value)
    raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
  void add_extension_point (in ExtensionPoint new_value)
    raises (Reflective::StructuralError);
  void modify_extension_point (
    in ExtensionPoint old_value,
    in ExtensionPoint new_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  void remove_extension_point (in ExtensionPoint old_value)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
}; // end of interface UseCase

interface ActorClass : Foundation::Core::ClassifierClass {
  readonly attribute ActorUList all_of_type_actor;
  Actor create_actor (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in boolean is_root,
            in boolean is_leaf,
            in boolean is_abstract)
            raises (
              Reflective::SemanticError,
              Reflective::ConstraintError);
    };

    interface Actor : ActorClass, Foundation::Core::Classifier {
    }; // end of interface Actor

    interface UseCaseInstanceClass : CommonBehavior::InstanceClass {
      readonly attribute UseCaseInstanceUList
all_of_type_use_case_instance;
      UseCaseInstance create_use_case_instance (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface UseCaseInstance : UseCaseInstanceClass, CommonBehav-
ior::Instance {
    }; // end of interface UseCaseInstance

    interface ExtendClass : Foundation::Core::RelationshipClass {
      readonly attribute ExtendUList all_of_type_extend;
      Extend create_extend (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::BooleanExpression condition)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface Extend : ExtendClass, Foundation::Core::Relationship {
      Foundation::BooleanExpression condition ()
        raises (Reflective::SemanticError);
      void set_condition (in Foundation::BooleanExpression new_value)
        raises (Reflective::SemanticError);
      UseCase base ()
        raises (Reflective::SemanticError);
      void set_base (in UseCase new_value)
        raises (Reflective::SemanticError);
      UseCase extension ()
        raises (Reflective::SemanticError);
      void set_extension (in UseCase new_value)
        raises (Reflective::SemanticError);
      ExtensionPointUList extension_point ()
        raises (Reflective::SemanticError);
```

```
  void set_extension_point (in ExtensionPointUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_extension_point (in ExtensionPoint new_value)
    raises (Reflective::StructuralError);
  void add_extension_point_before (
    in ExtensionPoint new_value,
    in ExtensionPoint before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_extension_point (
    in ExtensionPoint old_value,
    in ExtensionPoint new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_extension_point (in ExtensionPoint old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Extend

interface IncludeClass : Foundation::Core::RelationshipClass {
  readonly attribute IncludeUList all_of_type_include;
  Include create_include (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Include : IncludeClass, Foundation::Core::Relationship {
  UseCase addition ()
    raises (Reflective::SemanticError);
  void set_addition (in UseCase new_value)
    raises (Reflective::SemanticError);
  UseCase base ()
    raises (Reflective::SemanticError);
  void set_base (in UseCase new_value)
    raises (Reflective::SemanticError);
}; // end of interface Include

interface ExtensionPointClass : Foundation::Core::ModelElementClass {
  readonly attribute ExtensionPointUList all_of_type_extension_point;
  ExtensionPoint create_extension_point (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::LocationReference location)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface ExtensionPoint : ExtensionPointClass, Foundation::Core::Mod-
elElement {
      Foundation::LocationReference location ()
        raises (Reflective::SemanticError);
      void set_location (in Foundation::LocationReference new_value)
        raises (Reflective::SemanticError);
      UseCase use_case ()
        raises (Reflective::SemanticError);
      void set_use_case (in UseCase new_value)
        raises (Reflective::SemanticError);
      ExtendSet extend ()
        raises (Reflective::SemanticError);
      void set_extend (in ExtendSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_extend (in UseCases::Extend new_value)
        raises (Reflective::StructuralError);
      void modify_extend (
        in UseCases::Extend old_value,
        in UseCases::Extend new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_extend (in UseCases::Extend old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ExtensionPoint

    struct ABaseExtensionLink {
      UseCase base;
      Extend extension;
    };
    typedef sequence<ABaseExtensionLink> ABaseExtensionLinkSet;

    interface ABaseExtension : Reflective::RefAssociation {
      ABaseExtensionLinkSet all_a_base_extension_links();
      boolean exists (
        in UseCase base,
        in Extend extension);
```

```
  ExtendSet with_base (
    in UseCase base);
  UseCase with_extension (
    in Extend extension);
  void add (
    in UseCase base,
    in Extend extension)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_base (
    in UseCase base,
    in Extend extension,
    in UseCase new_base)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_extension (
    in UseCase base,
    in Extend extension,
    in Extend new_extension)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in UseCase base,
    in Extend extension)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ABaseExtension

struct AExtensionExtendLink {
  UseCase extension;
  Extend extend;
};
typedef sequence<AExtensionExtendLink> AExtensionExtendLinkSet;

interface AExtensionExtend : Reflective::RefAssociation {
  AExtensionExtendLinkSet all_a_extension_extend_links();
  boolean exists (
    in UseCase extension,
    in Extend extend);
  ExtendSet with_extension (
    in UseCase extension);
  UseCase with_extend (
    in Extend extend);
  void add (
    in UseCase extension,
    in Extend extend)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_extension (
    in UseCase extension,
    in Extend extend,
    in UseCase new_extension)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_extend (
    in UseCase extension,
    in Extend extend,
    in Extend new_extend)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in UseCase extension,
    in Extend extend)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AExtensionExtend

struct AIncludeAdditionLink {
  Include include;
  UseCase addition;
};
typedef sequence<AIncludeAdditionLink> AIncludeAdditionLinkSet;

interface AIncludeAddition : Reflective::RefAssociation {
  AIncludeAdditionLinkSet all_a_include_addition_links();
  boolean exists (
    in Include include,
    in UseCase addition);
  UseCase with_include (
    in Include include);
  IncludeSet with_addition (
    in UseCase addition);
  void add (
    in Include include,
    in UseCase addition)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_include (
    in Include include,
    in UseCase addition,
    in Include new_include)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_addition (
    in Include include,
    in UseCase addition,
    in UseCase new_addition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Include include,
    in UseCase addition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AIncludeAddition

struct AInclusionBaseLink {
  Include inclusion;
  UseCase base;
};
typedef sequence<AInclusionBaseLink> AInclusionBaseLinkSet;

interface AInclusionBase : Reflective::RefAssociation {
  AInclusionBaseLinkSet all_a_inclusion_base_links();
  boolean exists (
    in Include inclusion,
    in UseCase base);
  UseCase with_inclusion (
    in Include inclusion);
  IncludeSet with_base (
    in UseCase base);
  void add (
    in Include inclusion,
    in UseCase base)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_inclusion (
    in Include inclusion,
    in UseCase base,
    in Include new_inclusion)
    raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_base (
      in Include inclusion,
      in UseCase base,
      in UseCase new_base)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Include inclusion,
      in UseCase base)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AInclusionBase

  struct AExtensionPointUseCaseLink {
    ExtensionPoint extension_point;
    UseCase use_case;
  };
  typedef sequence<AExtensionPointUseCaseLink> AExtensionPointUse-
CaseLinkSet;

  interface AExtensionPointUseCase : Reflective::RefAssociation {
    AExtensionPointUseCaseLinkSet
all_a_extension_point_use_case_links();
    boolean exists (
      in ExtensionPoint extension_point,
      in UseCase use_case);
    UseCase with_extension_point (
      in ExtensionPoint extension_point);
    ExtensionPointSet with_use_case (
      in UseCase use_case);
    void add (
      in ExtensionPoint extension_point,
      in UseCase use_case)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_extension_point (
      in ExtensionPoint extension_point,
      in UseCase use_case,
      in ExtensionPoint new_extension_point)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
        void modify_use_case (
          in ExtensionPoint extension_point,
          in UseCase use_case,
          in UseCase new_use_case)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove (
          in ExtensionPoint extension_point,
          in UseCase use_case)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      }; // end of interface AExtensionPointUseCase

      struct AExtensionPointExtendLink {
        ExtensionPoint extension_point;
        Extend extend;
      };
      typedef sequence<AExtensionPointExtendLink> AExtensionPointEx-
tendLinkSet;

      interface AExtensionPointExtend : Reflective::RefAssociation {
        AExtensionPointExtendLinkSet all_a_extension_point_extend_links();
        boolean exists (
          in ExtensionPoint extension_point,
          in Extend extend);
        ExtendSet with_extension_point (
          in ExtensionPoint extension_point);
        ExtensionPointUList with_extend (
          in Extend extend);
        void add (
          in ExtensionPoint extension_point,
          in Extend extend)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void add_before_extension_point (
          in ExtensionPoint extension_point,
          in Extend extend,
          in ExtensionPoint before)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void modify_extension_point (
          in ExtensionPoint extension_point,
          in Extend extend,
          in ExtensionPoint new_extension_point)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_extend (
        in ExtensionPoint extension_point,
        in Extend extend,
        in Extend new_extend)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in ExtensionPoint extension_point,
        in Extend extend)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AExtensionPointExtend

    interface UseCasesPackageFactory {
      UseCasesPackage create_use_cases_package ()
        raises (Reflective::SemanticError);
    };

    interface UseCasesPackage : Reflective::RefPackage {
      readonly attribute UseCaseClass use_case_class_ref;
      readonly attribute ActorClass actor_class_ref;
      readonly attribute UseCaseInstanceClass
use_case_instance_class_ref;
      readonly attribute ExtendClass extend_class_ref;
      readonly attribute IncludeClass include_class_ref;
      readonly attribute ExtensionPointClass extension_point_class_ref;
      readonly attribute ABaseExtension a_base_extension_class_ref;
      readonly attribute AExtensionExtend a_extension_extend_class_ref;
      readonly attribute AIncludeAddition a_include_addition_class_ref;
      readonly attribute AInclusionBase a_inclusion_base_class_ref;
      readonly attribute AExtensionPointUseCase
a_extension_point_use_case_class_ref;
      readonly attribute AExtensionPointExtend
a_extension_point_extend_class_ref;
    };
  }; // end of module UseCases

  module StateMachines {
    interface StateMachineClass;
    interface StateMachine;
    typedef sequence<StateMachine> StateMachineSet;
    typedef sequence<StateMachine> StateMachineUList;
    interface EventClass;
interface Event;
typedef sequence<Event> EventSet;
typedef sequence<Event> EventUList;
interface StateClass;
interface State;
typedef sequence<State> StateSet;
typedef sequence<State> StateUList;
interface TimeEventClass;
interface TimeEvent;
typedef sequence<TimeEvent> TimeEventUList;
interface CallEventClass;
interface CallEvent;
typedef sequence<CallEvent> CallEventSet;
typedef sequence<CallEvent> CallEventUList;
interface SignalEventClass;
interface SignalEvent;
typedef sequence<SignalEvent> SignalEventSet;
typedef sequence<SignalEvent> SignalEventUList;
interface TransitionClass;
interface Transition;
typedef sequence<Transition> TransitionSet;
typedef sequence<Transition> TransitionUList;
interface StateVertexClass;
interface StateVertex;
typedef sequence<StateVertex> StateVertexSet;
typedef sequence<StateVertex> StateVertexUList;
interface CompositeStateClass;
interface CompositeState;
typedef sequence<CompositeState> CompositeStateUList;
interface ChangeEventClass;
interface ChangeEvent;
typedef sequence<ChangeEvent> ChangeEventUList;
interface GuardClass;
interface Guard;
typedef sequence<Guard> GuardUList;
interface PseudostateClass;
interface Pseudostate;
typedef sequence<Pseudostate> PseudostateUList;
interface SimpleStateClass;
interface SimpleState;
typedef sequence<SimpleState> SimpleStateUList;
interface SubmachineStateClass;
interface SubmachineState;
typedef sequence<SubmachineState> SubmachineStateSet;
typedef sequence<SubmachineState> SubmachineStateUList;
interface SynchStateClass;
interface SynchState;
typedef sequence<SynchState> SynchStateUList;
interface StubStateClass;
interface StubState;
typedef sequence<StubState> StubStateUList;
```

```
interface FinalStateClass;
interface FinalState;
typedef sequence<FinalState> FinalStateUList;
interface StateMachinesPackage;

interface StateMachineClass : Foundation::Core::ModelElementClass {
  readonly attribute StateMachineUList all_of_type_state_machine;
  StateMachine create_state_machine (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface StateMachine : StateMachineClass, Foundation::Core::Mod-
elElement {
  TransitionSet transitions ()
    raises (Reflective::SemanticError);
  void set_transitions (in TransitionSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_transitions (in Transition new_value)
    raises (Reflective::StructuralError);
  void modify_transitions (
    in Transition old_value,
    in Transition new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_transitions (in Transition old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  SubmachineStateSet submachine_state ()
    raises (Reflective::SemanticError);
  void set_submachine_state (in SubmachineStateSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_submachine_state (in SubmachineState new_value)
    raises (Reflective::StructuralError);
  void modify_submachine_state (
    in SubmachineState old_value,
    in SubmachineState new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_submachine_state (in SubmachineState old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface StateMachine

interface EventClass : Foundation::Core::ModelElementClass {
  readonly attribute EventUList all_of_type_event;
};

interface Event : EventClass, Foundation::Core::ModelElement {
  ParameterUList parameters ()
    raises (Reflective::SemanticError);
  void set_parameters (in ParameterUList new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_parameters (in Foundation::Core::Parameter new_value)
    raises (Reflective::StructuralError);
  void add_parameters_before (
    in Foundation::Core::Parameter new_value,
    in Foundation::Core::Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameters (
    in Foundation::Core::Parameter old_value,
    in Foundation::Core::Parameter new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_parameters (in Foundation::Core::Parameter old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  StateSet state ()
    raises (Reflective::SemanticError);
  void set_state (in StateSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_state ()
    raises (Reflective::SemanticError);
  void add_state (in StateMachines::State new_value)
    raises (Reflective::StructuralError);
  void modify_state (
    in StateMachines::State old_value,
    in StateMachines::State new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_state (in StateMachines::State old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  TransitionSet transition ()
    raises (Reflective::SemanticError);
  void set_transition (in TransitionSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_transition (in StateMachines::Transition new_value)
    raises (Reflective::StructuralError);
  void modify_transition (
    in StateMachines::Transition old_value,
    in StateMachines::Transition new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_transition (in StateMachines::Transition old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface Event

interface StateVertexClass : Foundation::Core::ModelElementClass {
  readonly attribute StateVertexUList all_of_type_state_vertex;
};

interface StateVertex : StateVertexClass, Foundation::Core::ModelEle-
ment {
  CompositeState container ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_container (in CompositeState new_value)
    raises (Reflective::SemanticError);
  void unset_container ()
    raises (Reflective::SemanticError);
  TransitionSet outgoing ()
    raises (Reflective::SemanticError);
  void set_outgoing (in TransitionSet new_value)
    raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
    void add_outgoing (in Transition new_value)
        raises (Reflective::StructuralError);
    void modify_outgoing (
        in Transition old_value,
        in Transition new_value)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void remove_outgoing (in Transition old_value)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    TransitionSet incoming ()
        raises (Reflective::SemanticError);
    void set_incoming (in TransitionSet new_value)
        raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
    void add_incoming (in Transition new_value)
        raises (Reflective::StructuralError);
    void modify_incoming (
        in Transition old_value,
        in Transition new_value)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void remove_incoming (in Transition old_value)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
}; // end of interface StateVertex

interface StateClass : StateVertexClass {
    readonly attribute StateUList all_of_type_state;
    State create_state (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility)
        raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
};

interface State : StateClass, StateVertex {
    CommonBehavior::Action entry ()
        raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_entry (in CommonBehavior::Action new_value)
  raises (Reflective::SemanticError);
void unset_entry ()
  raises (Reflective::SemanticError);
CommonBehavior::Action exit ()
  raises (
    Reflective::NotSet,
    Reflective::SemanticError);
void set_exit (in CommonBehavior::Action new_value)
  raises (Reflective::SemanticError);
void unset_exit ()
  raises (Reflective::SemanticError);
EventSet deferrable_event ()
  raises (Reflective::SemanticError);
void set_deferrable_event (in EventSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void unset_deferrable_event ()
  raises (Reflective::SemanticError);
void add_deferrable_event (in Event new_value)
  raises (Reflective::StructuralError);
void modify_deferrable_event (
  in Event old_value,
  in Event new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_deferrable_event (in Event old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
TransitionSet internal_transition ()
  raises (Reflective::SemanticError);
void set_internal_transition (in TransitionSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_internal_transition (in Transition new_value)
  raises (Reflective::StructuralError);
void modify_internal_transition (
  in Transition old_value,
  in Transition new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
  void remove_internal_transition (in Transition old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  CommonBehavior::Action do_activity ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_do_activity (in CommonBehavior::Action new_value)
    raises (Reflective::SemanticError);
  void unset_do_activity ()
    raises (Reflective::SemanticError);
}; // end of interface State

interface TimeEventClass : EventClass {
  readonly attribute TimeEventUList all_of_type_time_event;
  TimeEvent create_time_event (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::TimeExpression when)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface TimeEvent : TimeEventClass, Event {
  Foundation::TimeExpression when ()
    raises (Reflective::SemanticError);
  void set_when (in Foundation::TimeExpression new_value)
    raises (Reflective::SemanticError);
}; // end of interface TimeEvent

interface CallEventClass : EventClass {
  readonly attribute CallEventUList all_of_type_call_event;
  CallEvent create_call_event (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface CallEvent : CallEventClass, Event {
  Foundation::Core::Operation operation ()
    raises (Reflective::SemanticError);
  void set_operation (in Foundation::Core::Operation new_value)
    raises (Reflective::SemanticError);
}; // end of interface CallEvent

interface SignalEventClass : EventClass {
  readonly attribute SignalEventUList all_of_type_signal_event;
  SignalEvent create_signal_event (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface SignalEvent : SignalEventClass, Event {
  CommonBehavior::Signal signal ()
    raises (Reflective::SemanticError);
  void set_signal (in CommonBehavior::Signal new_value)
    raises (Reflective::SemanticError);
}; // end of interface SignalEvent

interface TransitionClass : Foundation::Core::ModelElementClass {
  readonly attribute TransitionUList all_of_type_transition;
  Transition create_transition (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Transition : TransitionClass, Foundation::Core::ModelElement {
  StateMachines::Guard guard ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_guard (in StateMachines::Guard new_value)
    raises (Reflective::SemanticError);
  void unset_guard ()
    raises (Reflective::SemanticError);
  CommonBehavior::Action effect ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_effect (in CommonBehavior::Action new_value)
    raises (Reflective::SemanticError);
  void unset_effect ()
    raises (Reflective::SemanticError);
  StateMachines::State state ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_state (in StateMachines::State new_value)
    raises (Reflective::SemanticError);
  void unset_state ()
    raises (Reflective::SemanticError);
  Event trigger ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_trigger (in Event new_value)
    raises (Reflective::SemanticError);
  void unset_trigger ()
    raises (Reflective::SemanticError);
  StateMachine state_machine ()
    raises (
      Reflective::NotSet,
      Reflective::SemanticError);
  void set_state_machine (in StateMachine new_value)
    raises (Reflective::SemanticError);
  void unset_state_machine ()
    raises (Reflective::SemanticError);
  StateVertex source ()
    raises (Reflective::SemanticError);
  void set_source (in StateVertex new_value)
    raises (Reflective::SemanticError);
  StateVertex target ()
    raises (Reflective::SemanticError);
  void set_target (in StateVertex new_value)
    raises (Reflective::SemanticError);
}; // end of interface Transition

interface CompositeStateClass : StateClass {
  readonly attribute CompositeStateUList all_of_type_composite_state;
  CompositeState create_composite_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in boolean is_concurent)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface CompositeState : CompositeStateClass, State {
  boolean is_concurent ()
    raises (Reflective::SemanticError);
  void set_is_concurent (in boolean new_value)
    raises (Reflective::SemanticError);
  StateVertexSet subvertex ()
    raises (Reflective::SemanticError);
  void set_subvertex (in StateVertexSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_subvertex (in StateVertex new_value)
    raises (Reflective::StructuralError);
  void modify_subvertex (
        in StateVertex old_value,
        in StateVertex new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_subvertex (in StateVertex old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface CompositeState

    interface ChangeEventClass : EventClass {
      readonly attribute ChangeEventUList all_of_type_change_event;
      ChangeEvent create_change_event (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::BooleanExpression change_expression)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface ChangeEvent : ChangeEventClass, Event {
      Foundation::BooleanExpression change_expression ()
        raises (Reflective::SemanticError);
      void set_change_expression (in Foundation::BooleanExpression
new_value)
        raises (Reflective::SemanticError);
    }; // end of interface ChangeEvent

    interface GuardClass : Foundation::Core::ModelElementClass {
      readonly attribute GuardUList all_of_type_guard;
      Guard create_guard (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in Foundation::BooleanExpression expression)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface Guard : GuardClass, Foundation::Core::ModelElement {
      Foundation::BooleanExpression expression ()
        raises (Reflective::SemanticError);
      void set_expression (in Foundation::BooleanExpression new_value)
        raises (Reflective::SemanticError);
      StateMachines::Transition transition ()
        raises (Reflective::SemanticError);
      void set_transition (in StateMachines::Transition new_value)
    raises (Reflective::SemanticError);
}; // end of interface Guard

interface PseudostateClass : StateVertexClass {
  readonly attribute PseudostateUList all_of_type_pseudostate;
  Pseudostate create_pseudostate (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::PseudostateKind kind)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Pseudostate : PseudostateClass, StateVertex {
  Foundation::PseudostateKind kind ()
    raises (Reflective::SemanticError);
  void set_kind (in Foundation::PseudostateKind new_value)
    raises (Reflective::SemanticError);
}; // end of interface Pseudostate

interface SimpleStateClass : StateClass {
  readonly attribute SimpleStateUList all_of_type_simple_state;
  SimpleState create_simple_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface SimpleState : SimpleStateClass, State {
}; // end of interface SimpleState

interface SubmachineStateClass : CompositeStateClass {
  readonly attribute SubmachineStateUList
all_of_type_submachine_state;
  SubmachineState create_submachine_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in boolean is_concurent)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface SubmachineState : SubmachineStateClass, CompositeState {
  StateMachine submachine ()
    raises (Reflective::SemanticError);
  void set_submachine (in StateMachine new_value)
    raises (Reflective::SemanticError);
```

```
}; // end of interface SubmachineState

interface SynchStateClass : StateVertexClass {
  readonly attribute SynchStateUList all_of_type_synch_state;
  SynchState create_synch_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::UnlimitedInteger bound)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface SynchState : SynchStateClass, StateVertex {
  Foundation::UnlimitedInteger bound ()
    raises (Reflective::SemanticError);
  void set_bound (in Foundation::UnlimitedInteger new_value)
    raises (Reflective::SemanticError);
}; // end of interface SynchState

interface StubStateClass : StateVertexClass {
  readonly attribute StubStateUList all_of_type_stub_state;
  StubState create_stub_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility,
    in Foundation::Name reference_state)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface StubState : StubStateClass, StateVertex {
  Foundation::Name reference_state ()
    raises (Reflective::SemanticError);
  void set_reference_state (in Foundation::Name new_value)
    raises (Reflective::SemanticError);
}; // end of interface StubState

interface FinalStateClass : StateClass {
  readonly attribute FinalStateUList all_of_type_final_state;
  FinalState create_final_state (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface FinalState : FinalStateClass, State {
}; // end of interface FinalState
```

```
struct AStateEntryLink {
  State state;
  CommonBehavior::Action entry;
};
typedef sequence<AStateEntryLink> AStateEntryLinkSet;

interface AStateEntry : Reflective::RefAssociation {
  AStateEntryLinkSet all_a_state_entry_links();
  boolean exists (
    in State state,
    in CommonBehavior::Action entry);
  CommonBehavior::Action with_state (
    in State state);
  State with_entry (
    in CommonBehavior::Action entry);
  void add (
    in State state,
    in CommonBehavior::Action entry)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_state (
    in State state,
    in CommonBehavior::Action entry,
    in State new_state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_entry (
    in State state,
    in CommonBehavior::Action entry,
    in CommonBehavior::Action new_entry)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in State state,
    in CommonBehavior::Action entry)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AStateEntry

struct AStateExitLink {
  State state;
  CommonBehavior::Action exit;
};
typedef sequence<AStateExitLink> AStateExitLinkSet;
```

```
interface AStateExit : Reflective::RefAssociation {
  AStateExitLinkSet all_a_state_exit_links();
  boolean exists (
    in State state,
    in CommonBehavior::Action exit);
  CommonBehavior::Action with_state (
    in State state);
  State with_exit (
    in CommonBehavior::Action exit);
  void add (
    in State state,
    in CommonBehavior::Action exit)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_state (
    in State state,
    in CommonBehavior::Action exit,
    in State new_state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_exit (
    in State state,
    in CommonBehavior::Action exit,
    in CommonBehavior::Action new_exit)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in State state,
    in CommonBehavior::Action exit)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AStateExit

struct AEventParametersLink {
  Event event;
  Foundation::Core::Parameter parameters;
};
typedef sequence<AEventParametersLink> AEventParametersLinkSet;

interface AEventParameters : Reflective::RefAssociation {
  AEventParametersLinkSet all_a_event_parameters_links();
  boolean exists (
    in Event event,
    in Foundation::Core::Parameter parameters);
  ParameterUList with_event (
    in Event event);
  Event with_parameters (
    in Foundation::Core::Parameter parameters);
  void add (
    in Event event,
    in Foundation::Core::Parameter parameters)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_before_parameters (
    in Event event,
    in Foundation::Core::Parameter parameters,
    in Foundation::Core::Parameter before)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_event (
    in Event event,
    in Foundation::Core::Parameter parameters,
    in Event new_event)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_parameters (
    in Event event,
    in Foundation::Core::Parameter parameters,
    in Foundation::Core::Parameter new_parameters)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Event event,
    in Foundation::Core::Parameter parameters)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AEventParameters

struct AGuardTransitionLink {
  Guard guard;
  Transition transition;
};
typedef sequence<AGuardTransitionLink> AGuardTransitionLinkSet;

interface AGuardTransition : Reflective::RefAssociation {
  AGuardTransitionLinkSet all_a_guard_transition_links();
  boolean exists (
    in Guard guard,
    in Transition transition);
  Transition with_guard (
    in Guard guard);
  Guard with_transition (
    in Transition transition);
  void add (
    in Guard guard,
    in Transition transition)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_guard (
    in Guard guard,
    in Transition transition,
    in Guard new_guard)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_transition (
    in Guard guard,
    in Transition transition,
    in Transition new_transition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Guard guard,
    in Transition transition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AGuardTransition

struct ASignalSendActionLink {
  CommonBehavior::Signal signal;
  CommonBehavior::SendAction send_action;
};
typedef sequence<ASignalSendActionLink> ASignalSendActionLinkSet;

interface ASignalSendAction : Reflective::RefAssociation {
  ASignalSendActionLinkSet all_a_signal_send_action_links();
  boolean exists (
    in CommonBehavior::Signal signal,
    in CommonBehavior::SendAction send_action);
  CommonBehavior::SendActionSet with_signal (
    in CommonBehavior::Signal signal);
  CommonBehavior::Signal with_send_action (
    in CommonBehavior::SendAction send_action);
  void add (
    in CommonBehavior::Signal signal,
    in CommonBehavior::SendAction send_action)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_signal (
    in CommonBehavior::Signal signal,
    in CommonBehavior::SendAction send_action,
    in CommonBehavior::Signal new_signal)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_send_action (
    in CommonBehavior::Signal signal,
    in CommonBehavior::SendAction send_action,
    in CommonBehavior::SendAction new_send_action)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in CommonBehavior::Signal signal,
    in CommonBehavior::SendAction send_action)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ASignalSendAction

struct ACalledCallActionLink {
  Foundation::Core::Operation called;
  CommonBehavior::CallAction call_action;
};
typedef sequence<ACalledCallActionLink> ACalledCallActionLinkSet;

interface ACalledCallAction : Reflective::RefAssociation {
  ACalledCallActionLinkSet all_a_called_call_action_links();
  boolean exists (
    in Foundation::Core::Operation called,
    in CommonBehavior::CallAction call_action);
  CommonBehavior::CallActionSet with_called (
    in Foundation::Core::Operation called);
  Foundation::Core::Operation with_call_action (
    in CommonBehavior::CallAction call_action);
  void add (
    in Foundation::Core::Operation called,
    in CommonBehavior::CallAction call_action)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_called (
    in Foundation::Core::Operation called,
    in CommonBehavior::CallAction call_action,
    in Foundation::Core::Operation new_called)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_call_action (
    in Foundation::Core::Operation called,
    in CommonBehavior::CallAction call_action,
    in CommonBehavior::CallAction new_call_action)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Foundation::Core::Operation called,
    in CommonBehavior::CallAction call_action)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ACalledCallAction

struct ASignalOccurrenceLink {
  CommonBehavior::Signal signal;
  SignalEvent occurrence;
};
typedef sequence<ASignalOccurrenceLink> ASignalOccurrenceLinkSet;

interface ASignalOccurrence : Reflective::RefAssociation {
  ASignalOccurrenceLinkSet all_a_signal_occurrence_links();
  boolean exists (
    in CommonBehavior::Signal signal,
    in SignalEvent occurrence);
  SignalEventSet with_signal (
    in CommonBehavior::Signal signal);
  CommonBehavior::Signal with_occurrence (
    in SignalEvent occurrence);
  void add (
    in CommonBehavior::Signal signal,
    in SignalEvent occurrence)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_signal (
            in CommonBehavior::Signal signal,
            in SignalEvent occurrence,
            in CommonBehavior::Signal new_signal)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
          void modify_occurrence (
            in CommonBehavior::Signal signal,
            in SignalEvent occurrence,
            in SignalEvent new_occurrence)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
          void remove (
            in CommonBehavior::Signal signal,
            in SignalEvent occurrence)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
        }; // end of interface ASignalOccurrence

        struct AStateDeferrableEventLink {
          State state;
          Event deferrable_event;
        };
        typedef sequence<AStateDeferrableEventLink> AStateDeferra-
    bleEventLinkSet;

        interface AStateDeferrableEvent : Reflective::RefAssociation {
          AStateDeferrableEventLinkSet all_a_state_deferrable_event_links();
          boolean exists (
            in State state,
            in Event deferrable_event);
          EventSet with_state (
            in State state);
          StateSet with_deferrable_event (
            in Event deferrable_event);
          void add (
            in State state,
            in Event deferrable_event)
            raises (
              Reflective::StructuralError,
              Reflective::SemanticError);
          void modify_state (
            in State state,
            in Event deferrable_event,
            in State new_state)
            raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void modify_deferrable_event (
        in State state,
        in Event deferrable_event,
        in Event new_deferrable_event)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void remove (
        in State state,
        in Event deferrable_event)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
  }; // end of interface AStateDeferrableEvent

  struct AOccurrenceOperationLink {
    CallEvent occurrence;
    Foundation::Core::Operation operation;
  };
  typedef sequence<AOccurrenceOperationLink> AOccurrenceOperation-
LinkSet;

  interface AOccurrenceOperation : Reflective::RefAssociation {
    AOccurrenceOperationLinkSet all_a_occurrence_operation_links();
    boolean exists (
        in CallEvent occurrence,
        in Foundation::Core::Operation operation);
    Foundation::Core::Operation with_occurrence (
        in CallEvent occurrence);
    CallEventSet with_operation (
        in Foundation::Core::Operation operation);
    void add (
        in CallEvent occurrence,
        in Foundation::Core::Operation operation)
        raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
    void modify_occurrence (
        in CallEvent occurrence,
        in Foundation::Core::Operation operation,
        in CallEvent new_occurrence)
        raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
    void modify_operation (
        in CallEvent occurrence,
        in Foundation::Core::Operation operation,
        in Foundation::Core::Operation new_operation)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in CallEvent occurrence,
        in Foundation::Core::Operation operation)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AOccurrenceOperation

    struct AContainerSubvertexLink {
      CompositeState container;
      StateVertex subvertex;
    };
    typedef sequence<AContainerSubvertexLink> AContainerSubvertex-
LinkSet;

    interface AContainerSubvertex : Reflective::RefAssociation {
      AContainerSubvertexLinkSet all_a_container_subvertex_links();
      boolean exists (
        in CompositeState container,
        in StateVertex subvertex);
      StateVertexSet with_container (
        in CompositeState container);
      CompositeState with_subvertex (
        in StateVertex subvertex);
      void add (
        in CompositeState container,
        in StateVertex subvertex)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_container (
        in CompositeState container,
        in StateVertex subvertex,
        in CompositeState new_container)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_subvertex (
        in CompositeState container,
        in StateVertex subvertex,
        in StateVertex new_subvertex)
        raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in CompositeState container,
    in StateVertex subvertex)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AContainerSubvertex

struct ATransitionEffectLink {
  Transition transition;
  CommonBehavior::Action effect;
};
typedef sequence<ATransitionEffectLink> ATransitionEffectLinkSet;

interface ATransitionEffect : Reflective::RefAssociation {
  ATransitionEffectLinkSet all_a_transition_effect_links();
  boolean exists (
    in Transition transition,
    in CommonBehavior::Action effect);
  CommonBehavior::Action with_transition (
    in Transition transition);
  Transition with_effect (
    in CommonBehavior::Action effect);
  void add (
    in Transition transition,
    in CommonBehavior::Action effect)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_transition (
    in Transition transition,
    in CommonBehavior::Action effect,
    in Transition new_transition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_effect (
    in Transition transition,
    in CommonBehavior::Action effect,
    in CommonBehavior::Action new_effect)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Transition transition,
            in CommonBehavior::Action effect)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
        }; // end of interface ATransitionEffect

        struct AStateInternalTransitionLink {
          State state;
          Transition internal_transition;
        };
        typedef sequence<AStateInternalTransitionLink> AStateInternalTransi-
    tionLinkSet;

        interface AStateInternalTransition : Reflective::RefAssociation {
          AStateInternalTransitionLinkSet all_a_state_internal_transition_links();
          boolean exists (
            in State state,
            in Transition internal_transition);
          TransitionSet with_state (
            in State state);
          State with_internal_transition (
            in Transition internal_transition);
          void add (
            in State state,
            in Transition internal_transition)
            raises (
              Reflective::StructuralError,
              Reflective::SemanticError);
          void modify_state (
            in State state,
            in Transition internal_transition,
            in State new_state)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
          void modify_internal_transition (
            in State state,
            in Transition internal_transition,
            in Transition new_internal_transition)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
              Reflective::SemanticError);
          void remove (
            in State state,
            in Transition internal_transition)
            raises (
              Reflective::StructuralError,
              Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AStateInternalTransition

struct ATransitionTriggerLink {
  Transition transition;
  Event trigger;
};
typedef sequence<ATransitionTriggerLink> ATransitionTriggerLinkSet;

interface ATransitionTrigger : Reflective::RefAssociation {
  ATransitionTriggerLinkSet all_a_transition_trigger_links();
  boolean exists (
    in Transition transition,
    in Event trigger);
  Event with_transition (
    in Transition transition);
  TransitionSet with_trigger (
    in Event trigger);
  void add (
    in Transition transition,
    in Event trigger)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_transition (
    in Transition transition,
    in Event trigger,
    in Transition new_transition)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_trigger (
    in Transition transition,
    in Event trigger,
    in Event new_trigger)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Transition transition,
    in Event trigger)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ATransitionTrigger

struct AStateMachineTransitionsLink {
  StateMachine state_machine;
```

```
    Transition transitions;
  };
  typedef sequence<AStateMachineTransitionsLink> AStateMachineTran-
sitionsLinkSet;

  interface AStateMachineTransitions : Reflective::RefAssociation {
    AStateMachineTransitionsLinkSet
all_a_state_machine_transitions_links();
    boolean exists (
      in StateMachine state_machine,
      in Transition transitions);
    TransitionSet with_state_machine (
      in StateMachine state_machine);
    StateMachine with_transitions (
      in Transition transitions);
    void add (
      in StateMachine state_machine,
      in Transition transitions)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_state_machine (
      in StateMachine state_machine,
      in Transition transitions,
      in StateMachine new_state_machine)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_transitions (
      in StateMachine state_machine,
      in Transition transitions,
      in Transition new_transitions)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in StateMachine state_machine,
      in Transition transitions)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AStateMachineTransitions

  struct AOutgoingSourceLink {
    Transition outgoing;
    StateVertex source;
  };
  typedef sequence<AOutgoingSourceLink> AOutgoingSourceLinkSet;
```

```
interface AOutgoingSource : Reflective::RefAssociation {
  AOutgoingSourceLinkSet all_a_outgoing_source_links();
  boolean exists (
    in Transition outgoing,
    in StateVertex source);
  StateVertex with_outgoing (
    in Transition outgoing);
  TransitionSet with_source (
    in StateVertex source);
  void add (
    in Transition outgoing,
    in StateVertex source)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_outgoing (
    in Transition outgoing,
    in StateVertex source,
    in Transition new_outgoing)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_source (
    in Transition outgoing,
    in StateVertex source,
    in StateVertex new_source)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Transition outgoing,
    in StateVertex source)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AOutgoingSource

struct AIncomingTargetLink {
  Transition incoming;
  StateVertex target;
};
typedef sequence<AIncomingTargetLink> AIncomingTargetLinkSet;

interface AIncomingTarget : Reflective::RefAssociation {
  AIncomingTargetLinkSet all_a_incoming_target_links();
  boolean exists (
    in Transition incoming,
      in StateVertex target);
    StateVertex with_incoming (
      in Transition incoming);
    TransitionSet with_target (
      in StateVertex target);
    void add (
      in Transition incoming,
      in StateVertex target)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_incoming (
      in Transition incoming,
      in StateVertex target,
      in Transition new_incoming)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_target (
      in Transition incoming,
      in StateVertex target,
      in StateVertex new_target)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Transition incoming,
      in StateVertex target)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AIncomingTarget

  struct ASubmachineStateSubmachineLink {
    SubmachineState submachine_state;
    StateMachine submachine;
  };
  typedef sequence<ASubmachineStateSubmachineLink> ASubma-
chineStateSubmachineLinkSet;

  interface ASubmachineStateSubmachine : Reflective::RefAssociation {
    ASubmachineStateSubmachineLinkSet
all_a_submachine_state_submachine_links();
    boolean exists (
      in SubmachineState submachine_state,
      in StateMachine submachine);
    StateMachine with_submachine_state (
      in SubmachineState submachine_state);
  SubmachineStateSet with_submachine (
    in StateMachine submachine);
  void add (
    in SubmachineState submachine_state,
    in StateMachine submachine)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_submachine_state (
    in SubmachineState submachine_state,
    in StateMachine submachine,
    in SubmachineState new_submachine_state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_submachine (
    in SubmachineState submachine_state,
    in StateMachine submachine,
    in StateMachine new_submachine)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in SubmachineState submachine_state,
    in StateMachine submachine)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ASubmachineStateSubmachine

struct AStateDoActivityLink {
  State state;
  CommonBehavior::Action do_activity;
};
typedef sequence<AStateDoActivityLink> AStateDoActivityLinkSet;

interface AStateDoActivity : Reflective::RefAssociation {
  AStateDoActivityLinkSet all_a_state_do_activity_links();
  boolean exists (
    in State state,
    in CommonBehavior::Action do_activity);
  CommonBehavior::Action with_state (
    in State state);
  State with_do_activity (
    in CommonBehavior::Action do_activity);
  void add (
    in State state,
    in CommonBehavior::Action do_activity)
```

```
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_state (
        in State state,
        in CommonBehavior::Action do_activity,
        in State new_state)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_do_activity (
        in State state,
        in CommonBehavior::Action do_activity,
        in CommonBehavior::Action new_do_activity)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in State state,
        in CommonBehavior::Action do_activity)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AStateDoActivity

    interface StateMachinesPackageFactory {
      StateMachinesPackage create_state_machines_package ()
        raises (Reflective::SemanticError);
    };

    interface StateMachinesPackage : Reflective::RefPackage {
      readonly attribute StateMachineClass state_machine_class_ref;
      readonly attribute EventClass event_class_ref;
      readonly attribute StateClass state_class_ref;
      readonly attribute TimeEventClass time_event_class_ref;
      readonly attribute CallEventClass call_event_class_ref;
      readonly attribute SignalEventClass signal_event_class_ref;
      readonly attribute TransitionClass transition_class_ref;
      readonly attribute StateVertexClass state_vertex_class_ref;
      readonly attribute CompositeStateClass composite_state_class_ref;
      readonly attribute ChangeEventClass change_event_class_ref;
      readonly attribute GuardClass guard_class_ref;
      readonly attribute PseudostateClass pseudostate_class_ref;
      readonly attribute SimpleStateClass simple_state_class_ref;
      readonly attribute SubmachineStateClass
submachine_state_class_ref;
      readonly attribute SynchStateClass synch_state_class_ref;
      readonly attribute StubStateClass stub_state_class_ref;
```

```
        readonly attribute FinalStateClass final_state_class_ref;
        readonly attribute AStateEntry a_state_entry_class_ref;
        readonly attribute AStateExit a_state_exit_class_ref;
        readonly attribute AEventParameters a_event_parameters_class_ref;
        readonly attribute AGuardTransition a_guard_transition_class_ref;
        readonly attribute ASignalSendAction
a_signal_send_action_class_ref;
        readonly attribute ACalledCallAction a_called_call_action_class_ref;
        readonly attribute ASignalOccurrence a_signal_occurrence_class_ref;
        readonly attribute AStateDeferrableEvent
a_state_deferrable_event_class_ref;
        readonly attribute AOccurrenceOperation
a_occurrence_operation_class_ref;
        readonly attribute AContainerSubvertex
a_container_subvertex_class_ref;
        readonly attribute ATransitionEffect a_transition_effect_class_ref;
        readonly attribute AStateInternalTransition
a_state_internal_transition_class_ref;
        readonly attribute ATransitionTrigger a_transition_trigger_class_ref;
        readonly attribute AStateMachineTransitions
a_state_machine_transitions_class_ref;
        readonly attribute AOutgoingSource a_outgoing_source_class_ref;
        readonly attribute AIncomingTarget a_incoming_target_class_ref;
        readonly attribute ASubmachineStateSubmachine
a_submachine_state_submachine_class_ref;
        readonly attribute AStateDoActivity a_state_do_activity_class_ref;
      };
   }; // end of module StateMachines

   module Collaborations {
      interface CollaborationClass;
      interface Collaboration;
      typedef sequence<Collaboration> CollaborationSet;
      typedef sequence<Collaboration> CollaborationUList;
      interface ClassifierRoleClass;
      interface ClassifierRole;
      typedef sequence<ClassifierRole> ClassifierRoleSet;
      typedef sequence<ClassifierRole> ClassifierRoleUList;
      interface AssociationRoleClass;
      interface AssociationRole;
      typedef sequence<AssociationRole> AssociationRoleSet;
      typedef sequence<AssociationRole> AssociationRoleUList;
      interface AssociationEndRoleClass;
      interface AssociationEndRole;
      typedef sequence<AssociationEndRole> AssociationEndRoleSet;
      typedef sequence<AssociationEndRole> AssociationEndRoleUList;
      interface MessageClass;
      interface Message;
      typedef sequence<Message> MessageSet;
      typedef sequence<Message> MessageUList;
      interface InteractionClass;
```

```
    interface Interaction;
    typedef sequence<Interaction> InteractionSet;
    typedef sequence<Interaction> InteractionUList;
    interface CollaborationsPackage;

    interface CollaborationClass : Foundation::Core::NamespaceClass {
      readonly attribute CollaborationUList all_of_type_collaboration;
      Collaboration create_collaboration (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface Collaboration : CollaborationClass, Founda-
tion::Core::Namespace {
      InteractionSet interaction ()
        raises (Reflective::SemanticError);
      void set_interaction (in InteractionSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_interaction (in Collaborations::Interaction new_value)
        raises (Reflective::StructuralError);
      void modify_interaction (
        in Collaborations::Interaction old_value,
        in Collaborations::Interaction new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_interaction (in Collaborations::Interaction old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      ModelElementSet constraining_element ()
        raises (Reflective::SemanticError);
      void set_constraining_element (in ModelElementSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_constraining_element (in Foundation::Core::ModelElement
new_value)
        raises (Reflective::StructuralError);
      void modify_constraining_element (
        in Foundation::Core::ModelElement old_value,
        in Foundation::Core::ModelElement new_value)
        raises (
          Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_constraining_element (in Foundation::Core::ModelEle-
ment old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    Foundation::Core::Classifier represented_classifier ()
      raises (
        Reflective::NotSet,
        Reflective::SemanticError);
    void set_represented_classifier (in Foundation::Core::Classifier
new_value)
      raises (Reflective::SemanticError);
    void unset_represented_classifier ()
      raises (Reflective::SemanticError);
    Foundation::Core::Operation represented_operation ()
      raises (
        Reflective::NotSet,
        Reflective::SemanticError);
    void set_represented_operation (in Foundation::Core::Operation
new_value)
      raises (Reflective::SemanticError);
    void unset_represented_operation ()
      raises (Reflective::SemanticError);
  }; // end of interface Collaboration

  interface ClassifierRoleClass : Foundation::Core::ClassifierClass {
    readonly attribute ClassifierRoleUList all_of_type_classifier_role;
    ClassifierRole create_classifier_role (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility,
      in boolean is_root,
      in boolean is_leaf,
      in boolean is_abstract,
      in Foundation::Multiplicity multiplicity)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface ClassifierRole : ClassifierRoleClass, Foundation::Core::Classi-
fier {
    Foundation::Multiplicity multiplicity ()
      raises (Reflective::SemanticError);
    void set_multiplicity (in Foundation::Multiplicity new_value)
      raises (Reflective::SemanticError);
    Foundation::Core::Classifier base ()
      raises (Reflective::SemanticError);
    void set_base (in Foundation::Core::Classifier new_value)
```

```
      raises (Reflective::SemanticError);
    FeatureSet available_feature ()
      raises (Reflective::SemanticError);
    void set_available_feature (in FeatureSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_available_feature (in Foundation::Core::Feature new_value)
      raises (Reflective::StructuralError);
    void modify_available_feature (
      in Foundation::Core::Feature old_value,
      in Foundation::Core::Feature new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_available_feature (in Foundation::Core::Feature
old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    MessageSet message ()
      raises (Reflective::SemanticError);
    void set_message (in MessageSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_message (in Collaborations::Message new_value)
      raises (Reflective::StructuralError);
    void modify_message (
      in Collaborations::Message old_value,
      in Collaborations::Message new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_message (in Collaborations::Message old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    MessageSet message1 ()
      raises (Reflective::SemanticError);
    void set_message1 (in MessageSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_message1 (in Collaborations::Message new_value)
      raises (Reflective::StructuralError);
    void modify_message1 (
        in Collaborations::Message old_value,
        in Collaborations::Message new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_message1 (in Collaborations::Message old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      ModelElementSet available_contents ()
        raises (Reflective::SemanticError);
      void set_available_contents (in ModelElementSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_available_contents (in Foundation::Core::ModelElement
new_value)
        raises (Reflective::StructuralError);
      void modify_available_contents (
        in Foundation::Core::ModelElement old_value,
        in Foundation::Core::ModelElement new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_available_contents (in Foundation::Core::ModelElement
old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ClassifierRole

    interface AssociationRoleClass : Foundation::Core::AssociationClass {
      readonly attribute AssociationRoleUList all_of_type_association_role;
      AssociationRole create_association_role (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in boolean is_root,
        in boolean is_leaf,
        in boolean is_abstract,
        in Foundation::Multiplicity multiplicity)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface AssociationRole : AssociationRoleClass, Founda-
tion::Core::Association {
      Foundation::Multiplicity multiplicity ()
        raises (Reflective::SemanticError);
      void set_multiplicity (in Foundation::Multiplicity new_value)
        raises (Reflective::SemanticError);
      Foundation::Core::Association base ()
        raises (
          Reflective::NotSet,
          Reflective::SemanticError);
      void set_base (in Foundation::Core::Association new_value)
        raises (Reflective::SemanticError);
      void unset_base ()
        raises (Reflective::SemanticError);
      MessageSet message ()
        raises (Reflective::SemanticError);
      void set_message (in MessageSet new_value)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void add_message (in Collaborations::Message new_value)
        raises (Reflective::StructuralError);
      void modify_message (
        in Collaborations::Message old_value,
        in Collaborations::Message new_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove_message (in Collaborations::Message old_value)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface AssociationRole

    interface AssociationEndRoleClass : Foundation::Core::Association-
EndClass {
      readonly attribute AssociationEndRoleUList
all_of_type_association_end_role;
      AssociationEndRole create_association_end_role (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in boolean is_navigable,
        in Foundation::OrderingKind ordering,
        in Foundation::AggregationKind aggregation,
        in Foundation::ScopeKind target_scope,
        in Foundation::Multiplicity multiplicity,
        in Foundation::ChangeableKind changeability)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };
```

```
  interface AssociationEndRole : AssociationEndRoleClass, Founda-
tion::Core::AssociationEnd {
    Foundation::Core::AssociationEnd base ()
      raises (
        Reflective::NotSet,
        Reflective::SemanticError);
    void set_base (in Foundation::Core::AssociationEnd new_value)
      raises (Reflective::SemanticError);
    void unset_base ()
      raises (Reflective::SemanticError);
    UmlAttributeSet available_qualifier ()
      raises (Reflective::SemanticError);
    void set_available_qualifier (in UmlAttributeSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_available_qualifier (in Foundation::Core::UmlAttribute
new_value)
      raises (Reflective::StructuralError);
    void modify_available_qualifier (
      in Foundation::Core::UmlAttribute old_value,
      in Foundation::Core::UmlAttribute new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_available_qualifier (in Foundation::Core::UmlAttribute
old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AssociationEndRole

  interface MessageClass : Reflective::RefObject {
    readonly attribute Collaborations::MessageUList all_of_type_message;
    Collaborations::Message create_message ()
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Message : MessageClass {
    Collaborations::Interaction interaction ()
      raises (Reflective::SemanticError);
    void set_interaction (in Collaborations::Interaction new_value)
      raises (Reflective::SemanticError);
    Collaborations::Message activator ()
      raises (
        Reflective::NotSet,
    Reflective::SemanticError);
void set_activator (in Collaborations::Message new_value)
  raises (Reflective::SemanticError);
void unset_activator ()
  raises (Reflective::SemanticError);
MessageSet message ()
  raises (Reflective::SemanticError);
void set_message (in MessageSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_message (in Collaborations::Message new_value)
  raises (Reflective::StructuralError);
void modify_message (
  in Collaborations::Message old_value,
  in Collaborations::Message new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_message (in Collaborations::Message old_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
ClassifierRole sender ()
  raises (Reflective::SemanticError);
void set_sender (in ClassifierRole new_value)
  raises (Reflective::SemanticError);
ClassifierRole receiver ()
  raises (Reflective::SemanticError);
void set_receiver (in ClassifierRole new_value)
  raises (Reflective::SemanticError);
MessageSet message1 ()
  raises (Reflective::SemanticError);
void set_message1 (in MessageSet new_value)
  raises (
    Reflective::StructuralError,
    Reflective::SemanticError);
void add_message1 (in Collaborations::Message new_value)
  raises (Reflective::StructuralError);
void modify_message1 (
  in Collaborations::Message old_value,
  in Collaborations::Message new_value)
  raises (
    Reflective::StructuralError,
    Reflective::NotFound,
    Reflective::SemanticError);
void remove_message1 (in Collaborations::Message old_value)
  raises (
    Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    MessageSet predecessor ()
      raises (Reflective::SemanticError);
    void set_predecessor (in MessageSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_predecessor (in Collaborations::Message new_value)
      raises (Reflective::StructuralError);
    void modify_predecessor (
      in Collaborations::Message old_value,
      in Collaborations::Message new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_predecessor (in Collaborations::Message old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    AssociationRole communication_connection ()
      raises (
        Reflective::NotSet,
        Reflective::SemanticError);
    void set_communication_connection (in AssociationRole new_value)
      raises (Reflective::SemanticError);
    void unset_communication_connection ()
      raises (Reflective::SemanticError);
    CommonBehavior::Action action ()
      raises (Reflective::SemanticError);
    void set_action (in CommonBehavior::Action new_value)
      raises (Reflective::SemanticError);
  }; // end of interface Message

  interface InteractionClass : Foundation::Core::ModelElementClass {
    readonly attribute InteractionUList all_of_type_interaction;
    Interaction create_interaction (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Interaction : InteractionClass, Foundation::Core::ModelElement
{
    MessageSet message ()
      raises (Reflective::SemanticError);
    void set_message (in MessageSet new_value)
```

```
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_message (in Collaborations::Message new_value)
      raises (Reflective::StructuralError);
    void modify_message (
      in Collaborations::Message old_value,
      in Collaborations::Message new_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove_message (in Collaborations::Message old_value)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    Collaboration uml_context ()
      raises (Reflective::SemanticError);
    void set_uml_context (in Collaboration new_value)
      raises (Reflective::SemanticError);
  }; // end of interface Interaction

  struct AInteractionMessageLink {
    Interaction interaction;
    Message message;
  };
  typedef sequence<AInteractionMessageLink> AInteractionMessageLink-
Set;

  interface AInteractionMessage : Reflective::RefAssociation {
    AInteractionMessageLinkSet all_a_interaction_message_links();
    boolean exists (
      in Interaction interaction,
      in Message message);
    MessageSet with_interaction (
      in Interaction interaction);
    Interaction with_message (
      in Message message);
    void add (
      in Interaction interaction,
      in Message message)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_interaction (
      in Interaction interaction,
      in Message message,
      in Interaction new_interaction)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_message (
      in Interaction interaction,
      in Message message,
      in Message new_message)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Interaction interaction,
      in Message message)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AInteractionMessage

  struct AContextInteractionLink {
    Collaboration uml_context;
    Interaction interaction;
  };
  typedef sequence<AContextInteractionLink> AContextInteractionLink-
Set;

  interface AContextInteraction : Reflective::RefAssociation {
    AContextInteractionLinkSet all_a_context_interaction_links();
    boolean exists (
      in Collaboration uml_context,
      in Interaction interaction);
    InteractionSet with_uml_context (
      in Collaboration uml_context);
    Collaboration with_interaction (
      in Interaction interaction);
    void add (
      in Collaboration uml_context,
      in Interaction interaction)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_uml_context (
      in Collaboration uml_context,
      in Interaction interaction,
      in Collaboration new_uml_context)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_interaction (
      in Collaboration uml_context,
      in Interaction interaction,
      in Interaction new_interaction)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Collaboration uml_context,
      in Interaction interaction)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AContextInteraction

  struct AClassifierRoleBaseLink {
    ClassifierRole classifier_role;
    Foundation::Core::Classifier base;
  };
  typedef sequence<AClassifierRoleBaseLink> AClassifierRoleBaseLink-
Set;

  interface AClassifierRoleBase : Reflective::RefAssociation {
    AClassifierRoleBaseLinkSet all_a_classifier_role_base_links();
    boolean exists (
      in ClassifierRole classifier_role,
      in Foundation::Core::Classifier base);
    Foundation::Core::Classifier with_classifier_role (
      in ClassifierRole classifier_role);
    ClassifierRoleSet with_base (
      in Foundation::Core::Classifier base);
    void add (
      in ClassifierRole classifier_role,
      in Foundation::Core::Classifier base)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_classifier_role (
      in ClassifierRole classifier_role,
      in Foundation::Core::Classifier base,
      in ClassifierRole new_classifier_role)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_base (
      in ClassifierRole classifier_role,
      in Foundation::Core::Classifier base,
      in Foundation::Core::Classifier new_base)
      raises (
        Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void remove (
         in ClassifierRole classifier_role,
         in Foundation::Core::Classifier base)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
   }; // end of interface AClassifierRoleBase

   struct ABaseAssociationEndRoleLink {
      Foundation::Core::AssociationEnd base;
      AssociationEndRole association_end_role;
   };
   typedef sequence<ABaseAssociationEndRoleLink> ABaseAssociation-
EndRoleLinkSet;

   interface ABaseAssociationEndRole : Reflective::RefAssociation {
      ABaseAssociationEndRoleLinkSet
all_a_base_association_end_role_links();
      boolean exists (
         in Foundation::Core::AssociationEnd base,
         in AssociationEndRole association_end_role);
      AssociationEndRoleSet with_base (
         in Foundation::Core::AssociationEnd base);
      Foundation::Core::AssociationEnd with_association_end_role (
         in AssociationEndRole association_end_role);
      void add (
         in Foundation::Core::AssociationEnd base,
         in AssociationEndRole association_end_role)
         raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
      void modify_base (
         in Foundation::Core::AssociationEnd base,
         in AssociationEndRole association_end_role,
         in Foundation::Core::AssociationEnd new_base)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void modify_association_end_role (
         in Foundation::Core::AssociationEnd base,
         in AssociationEndRole association_end_role,
         in AssociationEndRole new_association_end_role)
         raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      void remove (
      in Foundation::Core::AssociationEnd base,
      in AssociationEndRole association_end_role)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ABaseAssociationEndRole

  struct ABaseAssociationRoleLink {
    Foundation::Core::Association base;
    AssociationRole association_role;
  };
  typedef sequence<ABaseAssociationRoleLink> ABaseAssociation-
RoleLinkSet;

  interface ABaseAssociationRole : Reflective::RefAssociation {
    ABaseAssociationRoleLinkSet all_a_base_association_role_links();
    boolean exists (
      in Foundation::Core::Association base,
      in AssociationRole association_role);
    AssociationRoleSet with_base (
      in Foundation::Core::Association base);
    Foundation::Core::Association with_association_role (
      in AssociationRole association_role);
    void add (
      in Foundation::Core::Association base,
      in AssociationRole association_role)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_base (
      in Foundation::Core::Association base,
      in AssociationRole association_role,
      in Foundation::Core::Association new_base)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_association_role (
      in Foundation::Core::Association base,
      in AssociationRole association_role,
      in AssociationRole new_association_role)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Foundation::Core::Association base,
      in AssociationRole association_role)
      raises (
        Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  }; // end of interface ABaseAssociationRole

  struct AClassifierRoleAvailableFeatureLink {
    ClassifierRole classifier_role;
    Foundation::Core::Feature available_feature;
  };
  typedef sequence<AClassifierRoleAvailableFeatureLink> AClassifier-
RoleAvailableFeatureLinkSet;

  interface AClassifierRoleAvailableFeature : Reflective::RefAssociation {
    AClassifierRoleAvailableFeatureLinkSet
all_a_classifier_role_available_feature_links();
    boolean exists (
      in ClassifierRole classifier_role,
      in Foundation::Core::Feature available_feature);
    FeatureSet with_classifier_role (
      in ClassifierRole classifier_role);
    ClassifierRoleSet with_available_feature (
      in Foundation::Core::Feature available_feature);
    void add (
      in ClassifierRole classifier_role,
      in Foundation::Core::Feature available_feature)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_classifier_role (
      in ClassifierRole classifier_role,
      in Foundation::Core::Feature available_feature,
      in ClassifierRole new_classifier_role)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_available_feature (
      in ClassifierRole classifier_role,
      in Foundation::Core::Feature available_feature,
      in Foundation::Core::Feature new_available_feature)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ClassifierRole classifier_role,
      in Foundation::Core::Feature available_feature)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AClassifierRoleAvailableFeature
  struct ACollaborationConstrainingElementLink {
    Collaboration collaboration;
    Foundation::Core::ModelElement constraining_element;
  };
  typedef sequence<ACollaborationConstrainingElementLink>
ACollaborationConstrainingElementLinkSet;

  interface ACollaborationConstrainingElement : Reflective::RefAssocia-
tion {
    ACollaborationConstrainingElementLinkSet
all_a_collaboration_constraining_element_links();
    boolean exists (
      in Collaboration collaboration,
      in Foundation::Core::ModelElement constraining_element);
    ModelElementSet with_collaboration (
      in Collaboration collaboration);
    CollaborationSet with_constraining_element (
      in Foundation::Core::ModelElement constraining_element);
    void add (
      in Collaboration collaboration,
      in Foundation::Core::ModelElement constraining_element)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_collaboration (
      in Collaboration collaboration,
      in Foundation::Core::ModelElement constraining_element,
      in Collaboration new_collaboration)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_constraining_element (
      in Collaboration collaboration,
      in Foundation::Core::ModelElement constraining_element,
      in Foundation::Core::ModelElement new_constraining_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Collaboration collaboration,
      in Foundation::Core::ModelElement constraining_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface ACollaborationConstrainingElement

  struct AMessageActivatorLink {
    Message message;
    Message activator;
  };
  typedef sequence<AMessageActivatorLink> AMessageActivatorLinkSet;

  interface AMessageActivator : Reflective::RefAssociation {
    AMessageActivatorLinkSet all_a_message_activator_links();
    boolean exists (
      in Message message,
      in Message activator);
    Message with_message (
      in Message message);
    MessageSet with_activator (
      in Message activator);
    void add (
      in Message message,
      in Message activator)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_message (
      in Message message,
      in Message activator,
      in Message new_message)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_activator (
      in Message message,
      in Message activator,
      in Message new_activator)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Message message,
      in Message activator)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AMessageActivator

  struct ACollaborationRepresentedClassifierLink {
    Collaboration collaboration;
    Foundation::Core::Classifier represented_classifier;
  };
  typedef sequence<ACollaborationRepresentedClassifierLink>
ACollaborationRepresentedClassifierLinkSet;
```

```
    interface ACollaborationRepresentedClassifier : Reflective::RefAssocia-
tion {
      ACollaborationRepresentedClassifierLinkSet
all_a_collaboration_represented_classifier_links();
      boolean exists (
        in Collaboration collaboration,
        in Foundation::Core::Classifier represented_classifier);
      Foundation::Core::Classifier with_collaboration (
        in Collaboration collaboration);
      CollaborationSet with_represented_classifier (
        in Foundation::Core::Classifier represented_classifier);
      void add (
        in Collaboration collaboration,
        in Foundation::Core::Classifier represented_classifier)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_collaboration (
        in Collaboration collaboration,
        in Foundation::Core::Classifier represented_classifier,
        in Collaboration new_collaboration)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_represented_classifier (
        in Collaboration collaboration,
        in Foundation::Core::Classifier represented_classifier,
        in Foundation::Core::Classifier new_represented_classifier)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in Collaboration collaboration,
        in Foundation::Core::Classifier represented_classifier)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ACollaborationRepresentedClassifier

    struct ACollaborationRepresentedOperationLink {
      Collaboration collaboration;
      Foundation::Core::Operation represented_operation;
    };
    typedef sequence<ACollaborationRepresentedOperationLink>
ACollaborationRepresentedOperationLinkSet;

    interface ACollaborationRepresentedOperation : Reflective::RefAssocia-
```

```
tion {
      ACollaborationRepresentedOperationLinkSet
all_a_collaboration_represented_operation_links();
      boolean exists (
        in Collaboration collaboration,
        in Foundation::Core::Operation represented_operation);
      Foundation::Core::Operation with_collaboration (
        in Collaboration collaboration);
      CollaborationSet with_represented_operation (
        in Foundation::Core::Operation represented_operation);
      void add (
        in Collaboration collaboration,
        in Foundation::Core::Operation represented_operation)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_collaboration (
        in Collaboration collaboration,
        in Foundation::Core::Operation represented_operation,
        in Collaboration new_collaboration)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_represented_operation (
        in Collaboration collaboration,
        in Foundation::Core::Operation represented_operation,
        in Foundation::Core::Operation new_represented_operation)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in Collaboration collaboration,
        in Foundation::Core::Operation represented_operation)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ACollaborationRepresentedOperation

    struct AMessageSenderLink {
      Message message;
      ClassifierRole sender;
    };
    typedef sequence<AMessageSenderLink> AMessageSenderLinkSet;

    interface AMessageSender : Reflective::RefAssociation {
      AMessageSenderLinkSet all_a_message_sender_links();
      boolean exists (
        in Message message,
    in ClassifierRole sender);
  ClassifierRole with_message (
    in Message message);
  MessageSet with_sender (
    in ClassifierRole sender);
  void add (
    in Message message,
    in ClassifierRole sender)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_message (
    in Message message,
    in ClassifierRole sender,
    in Message new_message)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_sender (
    in Message message,
    in ClassifierRole sender,
    in ClassifierRole new_sender)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Message message,
    in ClassifierRole sender)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AMessageSender

struct AReceiverMessageLink {
  ClassifierRole receiver;
  Message message;
};
typedef sequence<AReceiverMessageLink> AReceiverMessageLinkSet;

interface AReceiverMessage : Reflective::RefAssociation {
  AReceiverMessageLinkSet all_a_receiver_message_links();
  boolean exists (
    in ClassifierRole receiver,
    in Message message);
  MessageSet with_receiver (
    in ClassifierRole receiver);
  ClassifierRole with_message (
    in Message message);
    void add (
      in ClassifierRole receiver,
      in Message message)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_receiver (
      in ClassifierRole receiver,
      in Message message,
      in ClassifierRole new_receiver)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_message (
      in ClassifierRole receiver,
      in Message message,
      in Message new_message)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ClassifierRole receiver,
      in Message message)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AReceiverMessage

  struct APredecessorMessageLink {
    Message predecessor;
    Message message;
  };
  typedef sequence<APredecessorMessageLink> APredecessorMes-
sageLinkSet;

  interface APredecessorMessage : Reflective::RefAssociation {
    APredecessorMessageLinkSet all_a_predecessor_message_links();
    boolean exists (
      in Message predecessor,
      in Message message);
    MessageSet with_predecessor (
      in Message predecessor);
    MessageSet with_message (
      in Message message);
    void add (
      in Message predecessor,
      in Message message)
      raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void modify_predecessor (
          in Message predecessor,
          in Message message,
          in Message new_predecessor)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void modify_message (
          in Message predecessor,
          in Message message,
          in Message new_message)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove (
          in Message predecessor,
          in Message message)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      }; // end of interface APredecessorMessage

      struct AMessageCommunicationConnectionLink {
        Message message;
        AssociationRole communication_connection;
      };
      typedef sequence<AMessageCommunicationConnectionLink>
  AMessageCommunicationConnectionLinkSet;

      interface AMessageCommunicationConnection : Reflective::RefAssocia-
  tion {
        AMessageCommunicationConnectionLinkSet
  all_a_message_communication_connection_links();
        boolean exists (
          in Message message,
          in AssociationRole communication_connection);
        AssociationRole with_message (
          in Message message);
        MessageSet with_communication_connection (
          in AssociationRole communication_connection);
        void add (
          in Message message,
          in AssociationRole communication_connection)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
```

```
   void modify_message (
     in Message message,
     in AssociationRole communication_connection,
     in Message new_message)
     raises (
       Reflective::StructuralError,
       Reflective::NotFound,
       Reflective::SemanticError);
   void modify_communication_connection (
     in Message message,
     in AssociationRole communication_connection,
     in AssociationRole new_communication_connection)
     raises (
       Reflective::StructuralError,
       Reflective::NotFound,
       Reflective::SemanticError);
   void remove (
     in Message message,
     in AssociationRole communication_connection)
     raises (
       Reflective::StructuralError,
       Reflective::NotFound,
       Reflective::SemanticError);
 }; // end of interface AMessageCommunicationConnection

 struct AClassifierRoleAvailableContentsLink {
   ClassifierRole classifier_role;
   Foundation::Core::ModelElement available_contents;
 };
 typedef sequence<AClassifierRoleAvailableContentsLink> AClassifier-
RoleAvailableContentsLinkSet;

 interface AClassifierRoleAvailableContents : Reflective::RefAssociation
{
   AClassifierRoleAvailableContentsLinkSet
all_a_classifier_role_available_contents_links();
   boolean exists (
     in ClassifierRole classifier_role,
     in Foundation::Core::ModelElement available_contents);
   ModelElementSet with_classifier_role (
     in ClassifierRole classifier_role);
   ClassifierRoleSet with_available_contents (
     in Foundation::Core::ModelElement available_contents);
   void add (
     in ClassifierRole classifier_role,
     in Foundation::Core::ModelElement available_contents)
     raises (
       Reflective::StructuralError,
       Reflective::SemanticError);
   void modify_classifier_role (
     in ClassifierRole classifier_role,
    in Foundation::Core::ModelElement available_contents,
    in ClassifierRole new_classifier_role)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_available_contents (
    in ClassifierRole classifier_role,
    in Foundation::Core::ModelElement available_contents,
    in Foundation::Core::ModelElement new_available_contents)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in ClassifierRole classifier_role,
    in Foundation::Core::ModelElement available_contents)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AClassifierRoleAvailableContents

struct AActionMessageLink {
  CommonBehavior::Action action;
  Message message;
};
typedef sequence<AActionMessageLink> AActionMessageLinkSet;

interface AActionMessage : Reflective::RefAssociation {
  AActionMessageLinkSet all_a_action_message_links();
  boolean exists (
    in CommonBehavior::Action action,
    in Message message);
  MessageSet with_action (
    in CommonBehavior::Action action);
  CommonBehavior::Action with_message (
    in Message message);
  void add (
    in CommonBehavior::Action action,
    in Message message)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_action (
    in CommonBehavior::Action action,
    in Message message,
    in CommonBehavior::Action new_action)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
            Reflective::SemanticError);
        void modify_message (
          in CommonBehavior::Action action,
          in Message message,
          in Message new_message)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove (
          in CommonBehavior::Action action,
          in Message message)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      }; // end of interface AActionMessage

      struct AAssociationEndRoleAvailableQualifierLink {
        AssociationEndRole association_end_role;
        Foundation::Core::UmlAttribute available_qualifier;
      };
      typedef sequence<AAssociationEndRoleAvailableQualifierLink>
AAssociationEndRoleAvailableQualifierLinkSet;

      interface AAssociationEndRoleAvailableQualifier : Reflective::RefAsso-
ciation {
        AAssociationEndRoleAvailableQualifierLinkSet
all_a_association_end_role_available_qualifier_links();
        boolean exists (
          in AssociationEndRole association_end_role,
          in Foundation::Core::UmlAttribute available_qualifier);
        UmlAttributeSet with_association_end_role (
          in AssociationEndRole association_end_role);
        AssociationEndRoleSet with_available_qualifier (
          in Foundation::Core::UmlAttribute available_qualifier);
        void add (
          in AssociationEndRole association_end_role,
          in Foundation::Core::UmlAttribute available_qualifier)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void modify_association_end_role (
          in AssociationEndRole association_end_role,
          in Foundation::Core::UmlAttribute available_qualifier,
          in AssociationEndRole new_association_end_role)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void modify_available_qualifier (
      in AssociationEndRole association_end_role,
      in Foundation::Core::UmlAttribute available_qualifier,
      in Foundation::Core::UmlAttribute new_available_qualifier)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in AssociationEndRole association_end_role,
      in Foundation::Core::UmlAttribute available_qualifier)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AAssociationEndRoleAvailableQualifier

  interface CollaborationsPackageFactory {
    CollaborationsPackage create_collaborations_package ()
      raises (Reflective::SemanticError);
  };

  interface CollaborationsPackage : Reflective::RefPackage {
    readonly attribute CollaborationClass collaboration_class_ref;
    readonly attribute ClassifierRoleClass classifier_role_class_ref;
    readonly attribute AssociationRoleClass association_role_class_ref;
    readonly attribute AssociationEndRoleClass
association_end_role_class_ref;
    readonly attribute MessageClass message_class_ref;
    readonly attribute InteractionClass interaction_class_ref;
    readonly attribute AInteractionMessage
a_interaction_message_class_ref;
    readonly attribute AContextInteraction
a_context_interaction_class_ref;
    readonly attribute AClassifierRoleBase
a_classifier_role_base_class_ref;
    readonly attribute ABaseAssociationEndRole
a_base_association_end_role_class_ref;
    readonly attribute ABaseAssociationRole
a_base_association_role_class_ref;
    readonly attribute AClassifierRoleAvailableFeature
a_classifier_role_available_feature_class_ref;
    readonly attribute ACollaborationConstrainingElement
a_collaboration_constraining_element_class_ref;
    readonly attribute AMessageActivator a_message_activator_class_ref;
    readonly attribute ACollaborationRepresentedClassifier
a_collaboration_represented_classifier_class_ref;
    readonly attribute ACollaborationRepresentedOperation
a_collaboration_represented_operation_class_ref;
    readonly attribute AMessageSender a_message_sender_class_ref;
    readonly attribute AReceiverMessage a_receiver_message_class_ref;
    readonly attribute APredecessorMessage
```

```
a_predecessor_message_class_ref;
      readonly attribute AMessageCommunicationConnection
a_message_communication_connection_class_ref;
      readonly attribute AClassifierRoleAvailableContents
a_classifier_role_available_contents_class_ref;
      readonly attribute AActionMessage a_action_message_class_ref;
      readonly attribute AAssociationEndRoleAvailableQualifier
a_association_end_role_available_qualifier_class_ref;
    };
  }; // end of module Collaborations

  module ActivityGraphs {
    interface ActivityGraphClass;
    interface ActivityGraph;
    typedef sequence<ActivityGraph> ActivityGraphUList;
    interface PartitionClass;
    interface Partition;
    typedef sequence<Partition> PartitionSet;
    typedef sequence<Partition> PartitionUList;
    interface SubactivityStateClass;
    interface SubactivityState;
    typedef sequence<SubactivityState> SubactivityStateUList;
    interface CallStateClass;
    interface CallState;
    typedef sequence<CallState> CallStateUList;
    interface ObjectFlowStateClass;
    interface ObjectFlowState;
    typedef sequence<ObjectFlowState> ObjectFlowStateSet;
    typedef sequence<ObjectFlowState> ObjectFlowStateUList;
    interface ClassifierInStateClass;
    interface ClassifierInState;
    typedef sequence<ClassifierInState> ClassifierInStateSet;
    typedef sequence<ClassifierInState> ClassifierInStateUList;
    interface ActionStateClass;
    interface ActionState;
    typedef sequence<ActionState> ActionStateUList;
    interface ActivityGraphsPackage;

    interface ActivityGraphClass : StateMachines::StateMachineClass {
      readonly attribute ActivityGraphUList all_of_type_activity_graph;
      ActivityGraph create_activity_graph (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface ActivityGraph : ActivityGraphClass, StateMachines::StateMa-
chine {
      PartitionSet partition ()
    raises (Reflective::SemanticError);
  void set_partition (in PartitionSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void unset_partition ()
    raises (Reflective::SemanticError);
  void add_partition (in ActivityGraphs::Partition new_value)
    raises (Reflective::StructuralError);
  void modify_partition (
    in ActivityGraphs::Partition old_value,
    in ActivityGraphs::Partition new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_partition (in ActivityGraphs::Partition old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ActivityGraph

interface PartitionClass : Foundation::Core::ModelElementClass {
  readonly attribute PartitionUList all_of_type_partition;
  Partition create_partition (
    in Foundation::Name name,
    in Foundation::VisibilityKind visibility)
    raises (
      Reflective::SemanticError,
      Reflective::ConstraintError);
};

interface Partition : PartitionClass, Foundation::Core::ModelElement {
  ModelElementSet contents ()
    raises (Reflective::SemanticError);
  void set_contents (in ModelElementSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_contents (in Foundation::Core::ModelElement new_value)
    raises (Reflective::StructuralError);
  void modify_contents (
    in Foundation::Core::ModelElement old_value,
    in Foundation::Core::ModelElement new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_contents (in Foundation::Core::ModelElement old_value)
    raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      ActivityGraph activity_graph ()
        raises (Reflective::SemanticError);
      void set_activity_graph (in ActivityGraph new_value)
        raises (Reflective::SemanticError);
    }; // end of interface Partition

    interface SubactivityStateClass : StateMachines::SubmachineStateClass
{
      readonly attribute SubactivityStateUList all_of_type_subactivity_state;
      SubactivityState create_subactivity_state (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in boolean is_concurent,
        in boolean is_dynamic,
        in Foundation::ArgListsExpression dynamic_arguments)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface SubactivityState : SubactivityStateClass, StateMachines::Sub-
machineState {
      boolean is_dynamic ()
        raises (Reflective::SemanticError);
      void set_is_dynamic (in boolean new_value)
        raises (Reflective::SemanticError);
      Foundation::ArgListsExpression dynamic_arguments ()
        raises (Reflective::SemanticError);
      void set_dynamic_arguments (in Foundation::ArgListsExpression
new_value)
        raises (Reflective::SemanticError);
    }; // end of interface SubactivityState

    interface ActionStateClass : StateMachines::SimpleStateClass {
      readonly attribute ActionStateUList all_of_type_action_state;
      ActionState create_action_state (
        in Foundation::Name name,
        in Foundation::VisibilityKind visibility,
        in boolean is_dynamic,
        in Foundation::ArgListsExpression dynamic_arguments)
        raises (
          Reflective::SemanticError,
          Reflective::ConstraintError);
    };

    interface ActionState : ActionStateClass, StateMachines::SimpleState {
      boolean is_dynamic ()
        raises (Reflective::SemanticError);
        void set_is_dynamic (in boolean new_value)
          raises (Reflective::SemanticError);
        Foundation::ArgListsExpression dynamic_arguments ()
          raises (Reflective::SemanticError);
        void set_dynamic_arguments (in Foundation::ArgListsExpression
new_value)
          raises (Reflective::SemanticError);
      }; // end of interface ActionState

      interface CallStateClass : ActionStateClass {
        readonly attribute CallStateUList all_of_type_call_state;
        CallState create_call_state (
          in Foundation::Name name,
          in Foundation::VisibilityKind visibility,
          in boolean is_dynamic,
          in Foundation::ArgListsExpression dynamic_arguments)
          raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
      };

      interface CallState : CallStateClass, ActionState {
      }; // end of interface CallState

      interface ObjectFlowStateClass : StateMachines::SimpleStateClass {
       readonly attribute ObjectFlowStateUList all_of_type_object_flow_state;
        ObjectFlowState create_object_flow_state (
          in Foundation::Name name,
          in Foundation::VisibilityKind visibility,
          in boolean is_synch)
          raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
      };

      interface ObjectFlowState : ObjectFlowStateClass, StateMachines::Sim-
pleState {
        boolean is_synch ()
          raises (Reflective::SemanticError);
        void set_is_synch (in boolean new_value)
          raises (Reflective::SemanticError);
        ClassifierInState type_state ()
          raises (Reflective::SemanticError);
        void set_type_state (in ClassifierInState new_value)
          raises (Reflective::SemanticError);
        ParameterSet parameter ()
          raises (Reflective::SemanticError);
        void set_parameter (in ParameterSet new_value)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void add_parameter (in Foundation::Core::Parameter new_value)
          raises (Reflective::StructuralError);
        void modify_parameter (
          in Foundation::Core::Parameter old_value,
          in Foundation::Core::Parameter new_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove_parameter (in Foundation::Core::Parameter old_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      }; // end of interface ObjectFlowState

      interface ClassifierInStateClass : Foundation::Core::ClassifierClass {
        readonly attribute ClassifierInStateUList
all_of_type_classifier_in_state;
        ClassifierInState create_classifier_in_state (
          in Foundation::Name name,
          in Foundation::VisibilityKind visibility,
          in boolean is_root,
          in boolean is_leaf,
          in boolean is_abstract)
          raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
      };

      interface ClassifierInState : ClassifierInStateClass, Founda-
tion::Core::Classifier {
        Foundation::Core::Classifier type ()
          raises (Reflective::SemanticError);
        void set_type (in Foundation::Core::Classifier new_value)
          raises (Reflective::SemanticError);
        ObjectFlowStateSet object_flow_state ()
          raises (Reflective::SemanticError);
        void set_object_flow_state (in ObjectFlowStateSet new_value)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void add_object_flow_state (in ObjectFlowState new_value)
          raises (Reflective::StructuralError);
        void modify_object_flow_state (
          in ObjectFlowState old_value,
          in ObjectFlowState new_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
```

```
  void remove_object_flow_state (in ObjectFlowState old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  StateMachines::StateSet in_state ()
    raises (Reflective::SemanticError);
  void set_in_state (in StateMachines::StateSet new_value)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void add_in_state (in StateMachines::State new_value)
    raises (Reflective::StructuralError);
  void modify_in_state (
    in StateMachines::State old_value,
    in StateMachines::State new_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove_in_state (in StateMachines::State old_value)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ClassifierInState

struct ABehaviorContextLink {
  StateMachines::StateMachine behavior;
  Foundation::Core::ModelElement uml_context;
};
typedef sequence<ABehaviorContextLink> ABehaviorContextLinkSet;

interface ABehaviorContext : Reflective::RefAssociation {
  ABehaviorContextLinkSet all_a_behavior_context_links();
  boolean exists (
    in StateMachines::StateMachine behavior,
    in Foundation::Core::ModelElement uml_context);
  Foundation::Core::ModelElement with_behavior (
    in StateMachines::StateMachine behavior);
  StateMachines::StateMachineSet with_uml_context (
    in Foundation::Core::ModelElement uml_context);
  void add (
    in StateMachines::StateMachine behavior,
    in Foundation::Core::ModelElement uml_context)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_behavior (
    in StateMachines::StateMachine behavior,
    in Foundation::Core::ModelElement uml_context,
    in StateMachines::StateMachine new_behavior)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_uml_context (
    in StateMachines::StateMachine behavior,
    in Foundation::Core::ModelElement uml_context,
    in Foundation::Core::ModelElement new_uml_context)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in StateMachines::StateMachine behavior,
    in Foundation::Core::ModelElement uml_context)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ABehaviorContext

struct AContentsPartitionLink {
  Foundation::Core::ModelElement contents;
  Partition partition;
};
typedef sequence<AContentsPartitionLink> AContentsPartitionLinkSet;

interface AContentsPartition : Reflective::RefAssociation {
  AContentsPartitionLinkSet all_a_contents_partition_links();
  boolean exists (
    in Foundation::Core::ModelElement contents,
    in Partition partition);
  PartitionSet with_contents (
    in Foundation::Core::ModelElement contents);
  ModelElementSet with_partition (
    in Partition partition);
  void add (
    in Foundation::Core::ModelElement contents,
    in Partition partition)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_contents (
    in Foundation::Core::ModelElement contents,
    in Partition partition,
    in Foundation::Core::ModelElement new_contents)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
    void modify_partition (
      in Foundation::Core::ModelElement contents,
      in Partition partition,
      in Partition new_partition)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Foundation::Core::ModelElement contents,
      in Partition partition)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AContentsPartition

  struct AActivityGraphPartitionLink {
    ActivityGraph activity_graph;
    Partition partition;
  };
  typedef sequence<AActivityGraphPartitionLink> AActivityGraphParti-
tionLinkSet;

  interface AActivityGraphPartition : Reflective::RefAssociation {
    AActivityGraphPartitionLinkSet all_a_activity_graph_partition_links();
    boolean exists (
      in ActivityGraph activity_graph,
      in Partition partition);
    PartitionSet with_activity_graph (
      in ActivityGraph activity_graph);
    ActivityGraph with_partition (
      in Partition partition);
    void add (
      in ActivityGraph activity_graph,
      in Partition partition)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_activity_graph (
      in ActivityGraph activity_graph,
      in Partition partition,
      in ActivityGraph new_activity_graph)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_partition (
      in ActivityGraph activity_graph,
      in Partition partition,
      in Partition new_partition)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ActivityGraph activity_graph,
      in Partition partition)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AActivityGraphPartition

  struct ATypeClassifierInStateLink {
    Foundation::Core::Classifier type;
    ClassifierInState classifier_in_state;
  };
  typedef sequence<ATypeClassifierInStateLink> ATypeClassifierIn-
StateLinkSet;

  interface ATypeClassifierInState : Reflective::RefAssociation {
    ATypeClassifierInStateLinkSet all_a_type_classifier_in_state_links();
    boolean exists (
      in Foundation::Core::Classifier type,
      in ClassifierInState classifier_in_state);
    ClassifierInStateSet with_type (
      in Foundation::Core::Classifier type);
    Foundation::Core::Classifier with_classifier_in_state (
      in ClassifierInState classifier_in_state);
    void add (
      in Foundation::Core::Classifier type,
      in ClassifierInState classifier_in_state)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_type (
      in Foundation::Core::Classifier type,
      in ClassifierInState classifier_in_state,
      in Foundation::Core::Classifier new_type)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_classifier_in_state (
      in Foundation::Core::Classifier type,
      in ClassifierInState classifier_in_state,
      in ClassifierInState new_classifier_in_state)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
      void remove (
        in Foundation::Core::Classifier type,
        in ClassifierInState classifier_in_state)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
    }; // end of interface ATypeClassifierInState

    struct ATypeStateObjectFlowStateLink {
      ClassifierInState type_state;
      ObjectFlowState object_flow_state;
    };
    typedef sequence<ATypeStateObjectFlowStateLink> ATypeStateObject-
FlowStateLinkSet;

    interface ATypeStateObjectFlowState : Reflective::RefAssociation {
      ATypeStateObjectFlowStateLinkSet
all_a_type_state_object_flow_state_links();
      boolean exists (
        in ClassifierInState type_state,
        in ObjectFlowState object_flow_state);
      ObjectFlowStateSet with_type_state (
        in ClassifierInState type_state);
      ClassifierInState with_object_flow_state (
        in ObjectFlowState object_flow_state);
      void add (
        in ClassifierInState type_state,
        in ObjectFlowState object_flow_state)
        raises (
          Reflective::StructuralError,
          Reflective::SemanticError);
      void modify_type_state (
        in ClassifierInState type_state,
        in ObjectFlowState object_flow_state,
        in ClassifierInState new_type_state)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void modify_object_flow_state (
        in ClassifierInState type_state,
        in ObjectFlowState object_flow_state,
        in ObjectFlowState new_object_flow_state)
        raises (
          Reflective::StructuralError,
          Reflective::NotFound,
          Reflective::SemanticError);
      void remove (
        in ClassifierInState type_state,
        in ObjectFlowState object_flow_state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ATypeStateObjectFlowState

struct AParameterStateLink {
  Foundation::Core::Parameter parameter;
  ObjectFlowState state;
};
typedef sequence<AParameterStateLink> AParameterStateLinkSet;

interface AParameterState : Reflective::RefAssociation {
  AParameterStateLinkSet all_a_parameter_state_links();
  boolean exists (
    in Foundation::Core::Parameter parameter,
    in ObjectFlowState state);
  ObjectFlowStateSet with_parameter (
    in Foundation::Core::Parameter parameter);
  ParameterSet with_state (
    in ObjectFlowState state);
  void add (
    in Foundation::Core::Parameter parameter,
    in ObjectFlowState state)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_parameter (
    in Foundation::Core::Parameter parameter,
    in ObjectFlowState state,
    in Foundation::Core::Parameter new_parameter)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_state (
    in Foundation::Core::Parameter parameter,
    in ObjectFlowState state,
    in ObjectFlowState new_state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in Foundation::Core::Parameter parameter,
    in ObjectFlowState state)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface AParameterState
```

```
struct ATopStateMachineLink {
  StateMachines::State top;
  StateMachines::StateMachine state_machine;
};
typedef sequence<ATopStateMachineLink> ATopStateMachineLinkSet;

interface ATopStateMachine : Reflective::RefAssociation {
  ATopStateMachineLinkSet all_a_top_state_machine_links();
  boolean exists (
    in StateMachines::State top,
    in StateMachines::StateMachine state_machine);
  StateMachines::StateMachine with_top (
    in StateMachines::State top);
  StateMachines::State with_state_machine (
    in StateMachines::StateMachine state_machine);
  void add (
    in StateMachines::State top,
    in StateMachines::StateMachine state_machine)
    raises (
      Reflective::StructuralError,
      Reflective::SemanticError);
  void modify_top (
    in StateMachines::State top,
    in StateMachines::StateMachine state_machine,
    in StateMachines::State new_top)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void modify_state_machine (
    in StateMachines::State top,
    in StateMachines::StateMachine state_machine,
    in StateMachines::StateMachine new_state_machine)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
  void remove (
    in StateMachines::State top,
    in StateMachines::StateMachine state_machine)
    raises (
      Reflective::StructuralError,
      Reflective::NotFound,
      Reflective::SemanticError);
}; // end of interface ATopStateMachine

struct AClassifierInStateInStateLink {
  ClassifierInState classifier_in_state;
  StateMachines::State in_state;
};
```

```
  typedef sequence<AClassifierInStateInStateLink> AClassifierInStateIn-
StateLinkSet;

  interface AClassifierInStateInState : Reflective::RefAssociation {
    AClassifierInStateInStateLinkSet
all_a_classifier_in_state_in_state_links();
    boolean exists (
      in ClassifierInState classifier_in_state,
      in StateMachines::State in_state);
    StateMachines::StateSet with_classifier_in_state (
      in ClassifierInState classifier_in_state);
    ClassifierInStateSet with_in_state (
      in StateMachines::State in_state);
    void add (
      in ClassifierInState classifier_in_state,
      in StateMachines::State in_state)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_classifier_in_state (
      in ClassifierInState classifier_in_state,
      in StateMachines::State in_state,
      in ClassifierInState new_classifier_in_state)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_in_state (
      in ClassifierInState classifier_in_state,
      in StateMachines::State in_state,
      in StateMachines::State new_in_state)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ClassifierInState classifier_in_state,
      in StateMachines::State in_state)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AClassifierInStateInState

  interface ActivityGraphsPackageFactory {
    ActivityGraphsPackage create_activity_graphs_package ()
      raises (Reflective::SemanticError);
  };

  interface ActivityGraphsPackage : Reflective::RefPackage {
    readonly attribute ActivityGraphClass activity_graph_class_ref;
```

```
      readonly attribute PartitionClass partition_class_ref;
      readonly attribute SubactivityStateClass subactivity_state_class_ref;
      readonly attribute CallStateClass call_state_class_ref;
      readonly attribute ObjectFlowStateClass object_flow_state_class_ref;
      readonly attribute ClassifierInStateClass classifier_in_state_class_ref;
      readonly attribute ActionStateClass action_state_class_ref;
      readonly attribute ABehaviorContext a_behavior_context_class_ref;
      readonly attribute AContentsPartition a_contents_partition_class_ref;
      readonly attribute AActivityGraphPartition
a_activity_graph_partition_class_ref;
      readonly attribute ATypeClassifierInState
a_type_classifier_in_state_class_ref;
      readonly attribute ATypeStateObjectFlowState
a_type_state_object_flow_state_class_ref;
      readonly attribute AParameterState a_parameter_state_class_ref;
      readonly attribute ATopStateMachine a_top_state_machine_class_ref;
      readonly attribute AClassifierInStateInState
a_classifier_in_state_in_state_class_ref;
    };
  }; // end of module ActivityGraphs

  interface BehavioralElementsPackageFactory {
    BehavioralElementsPackage create_behavioral_elements_package ()
      raises (Reflective::SemanticError);
  };

  interface BehavioralElementsPackage : Reflective::RefPackage {
    readonly attribute CommonBehavior::CommonBehaviorPackage
common_behavior_ref;
    readonly attribute UseCases::UseCasesPackage use_cases_ref;
    readonly attribute StateMachines::StateMachinesPackage
state_machines_ref;
    readonly attribute Collaborations::CollaborationsPackage
collaborations_ref;
    readonly attribute ActivityGraphs::ActivityGraphsPackage
activity_graphs_ref;
  };
};
```

5.4.4 UMLModelManagement

```
#include "Reflective.idl"
#include "Foundation.idl"

module ModelManagement {
  typedef sequence<Foundation::Core::ModelElement> ModelElementSet;
  interface ModelClass;
  interface Model;
  typedef sequence<Model> ModelUList;
  interface PackageClass;
  interface Package;
  typedef sequence<Package> PackageSet;
  typedef sequence<Package> PackageUList;
  interface SubsystemClass;
  interface Subsystem;
  typedef sequence<Subsystem> SubsystemUList;
  interface ElementImportClass;
  interface ElementImport;
  typedef sequence<ElementImport> ElementImportSet;
  typedef sequence<ElementImport> ElementImportUList;
  interface ModelManagementPackage;

  interface PackageClass : Foundation::Core::GeneralizableElementClass,
Foundation::Core::NamespaceClass {
    readonly attribute PackageUList all_of_type_package;
    Package create_package (
      in Foundation::Name name,
      in Foundation::VisibilityKind visibility,
      in boolean is_root,
      in boolean is_leaf,
      in boolean is_abstract)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Package : PackageClass, Foundation::Core::GeneralizableEle-
ment, Foundation::Core::Namespace {
    ModelElementSet imported_element ()
      raises (Reflective::SemanticError);
    void set_imported_element (in ModelElementSet new_value)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void add_imported_element (in Foundation::Core::ModelElement
new_value)
      raises (Reflective::StructuralError);
    void modify_imported_element (
      in Foundation::Core::ModelElement old_value,
      in Foundation::Core::ModelElement new_value)
      raises (
        Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove_imported_element (in Foundation::Core::ModelElement
    old_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        ElementImportSet element_import ()
          raises (Reflective::SemanticError);
        void set_element_import (in ElementImportSet new_value)
          raises (
            Reflective::StructuralError,
            Reflective::SemanticError);
        void add_element_import (in ElementImport new_value)
          raises (Reflective::StructuralError);
        void modify_element_import (
          in ElementImport old_value,
          in ElementImport new_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
        void remove_element_import (in ElementImport old_value)
          raises (
            Reflective::StructuralError,
            Reflective::NotFound,
            Reflective::SemanticError);
      }; // end of interface Package

      interface ModelClass : PackageClass {
        readonly attribute ModelUList all_of_type_model;
        Model create_model (
          in Foundation::Name name,
          in Foundation::VisibilityKind visibility,
          in boolean is_root,
          in boolean is_leaf,
          in boolean is_abstract)
          raises (
            Reflective::SemanticError,
            Reflective::ConstraintError);
      };

      interface Model : ModelClass, Package {
      }; // end of interface Model

      interface SubsystemClass : PackageClass, Foundation::Core::Classi-
    fierClass {
        readonly attribute SubsystemUList all_of_type_subsystem;
        Subsystem create_subsystem (
          in Foundation::Name name,
      in Foundation::VisibilityKind visibility,
      in boolean is_root,
      in boolean is_leaf,
      in boolean is_abstract,
      in boolean is_instantiable)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface Subsystem : SubsystemClass, Package, Foundation::Core::Clas-
sifier {
    boolean is_instantiable ()
      raises (Reflective::SemanticError);
    void set_is_instantiable (in boolean new_value)
      raises (Reflective::SemanticError);
  }; // end of interface Subsystem

  interface ElementImportClass : Reflective::RefObject {
    readonly attribute ElementImportUList all_of_type_element_import;
    ElementImport create_element_import (
      in Foundation::VisibilityKind visibility,
      in Foundation::Name alias)
      raises (
        Reflective::SemanticError,
        Reflective::ConstraintError);
  };

  interface ElementImport : ElementImportClass {
    Foundation::VisibilityKind visibility ()
      raises (Reflective::SemanticError);
    void set_visibility (in Foundation::VisibilityKind new_value)
      raises (Reflective::SemanticError);
    Foundation::Name alias ()
      raises (Reflective::SemanticError);
    void set_alias (in Foundation::Name new_value)
      raises (Reflective::SemanticError);
    Foundation::Core::ModelElement model_element ()
      raises (Reflective::SemanticError);
    void set_model_element (in Foundation::Core::ModelElement
new_value)
      raises (Reflective::SemanticError);
    ModelManagement::Package package ()
      raises (Reflective::SemanticError);
    void set_package (in ModelManagement::Package new_value)
      raises (Reflective::SemanticError);
  }; // end of interface ElementImport

  struct APackageImportedElementLink {
    Package package;
    Foundation::Core::ModelElement imported_element;
  };
  typedef sequence<APackageImportedElementLink> APackageImport-
edElementLinkSet;

  interface APackageImportedElement : Reflective::RefAssociation {
    APackageImportedElementLinkSet
all_a_package_imported_element_links();
    boolean exists (
      in Package package,
      in Foundation::Core::ModelElement imported_element);
    ModelElementSet with_package (
      in Package package);
    PackageSet with_imported_element (
      in Foundation::Core::ModelElement imported_element);
    void add (
      in Package package,
      in Foundation::Core::ModelElement imported_element)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_package (
      in Package package,
      in Foundation::Core::ModelElement imported_element,
      in Package new_package)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_imported_element (
      in Package package,
      in Foundation::Core::ModelElement imported_element,
      in Foundation::Core::ModelElement new_imported_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Package package,
      in Foundation::Core::ModelElement imported_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface APackageImportedElement

  struct AElementImportModelElementLink {
    ElementImport element_import;
    Foundation::Core::ModelElement model_element;
  };
  typedef sequence<AElementImportModelElementLink> AElementImport-
ModelElementLinkSet;
```

```
  interface AElementImportModelElement : Reflective::RefAssociation {
    AElementImportModelElementLinkSet
all_a_element_import_model_element_links();
    boolean exists (
      in ElementImport element_import,
      in Foundation::Core::ModelElement model_element);
    Foundation::Core::ModelElement with_element_import (
      in ElementImport element_import);
    ElementImportSet with_model_element (
      in Foundation::Core::ModelElement model_element);
    void add (
      in ElementImport element_import,
      in Foundation::Core::ModelElement model_element)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_element_import (
      in ElementImport element_import,
      in Foundation::Core::ModelElement model_element,
      in ElementImport new_element_import)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_model_element (
      in ElementImport element_import,
      in Foundation::Core::ModelElement model_element,
      in Foundation::Core::ModelElement new_model_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in ElementImport element_import,
      in Foundation::Core::ModelElement model_element)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface AElementImportModelElement

  struct APackageElementImportLink {
    Package package;
    ElementImport element_import;
  };
  typedef sequence<APackageElementImportLink> APackageElementIm-
portLinkSet;

  interface APackageElementImport : Reflective::RefAssociation {
    APackageElementImportLinkSet all_a_package_element_import_links();
    boolean exists (
      in Package package,
      in ElementImport element_import);
    ElementImportSet with_package (
      in Package package);
    Package with_element_import (
      in ElementImport element_import);
    void add (
      in Package package,
      in ElementImport element_import)
      raises (
        Reflective::StructuralError,
        Reflective::SemanticError);
    void modify_package (
      in Package package,
      in ElementImport element_import,
      in Package new_package)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void modify_element_import (
      in Package package,
      in ElementImport element_import,
      in ElementImport new_element_import)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
    void remove (
      in Package package,
      in ElementImport element_import)
      raises (
        Reflective::StructuralError,
        Reflective::NotFound,
        Reflective::SemanticError);
  }; // end of interface APackageElementImport

  interface ModelManagementPackageFactory {
    ModelManagementPackage create_model_management_package ()
      raises (Reflective::SemanticError);
  };

  interface ModelManagementPackage : Reflective::RefPackage {
    readonly attribute ModelClass model_class_ref;
    readonly attribute PackageClass package_class_ref;
    readonly attribute SubsystemClass subsystem_class_ref;
    readonly attribute ElementImportClass element_import_class_ref;
    readonly attribute APackageImportedElement
a_package_imported_element_class_ref;
    readonly attribute AElementImportModelElement
```

```
a_element_import_model_element_class_ref;
    readonly attribute APackageElementImport
a_package_element_import_class_ref;
  };
};
```

5 OA&D CORBAfacility InterfaceDefinition