IBM Software Group

EGL – Path to Standardization

Rational. software

e business on demand software

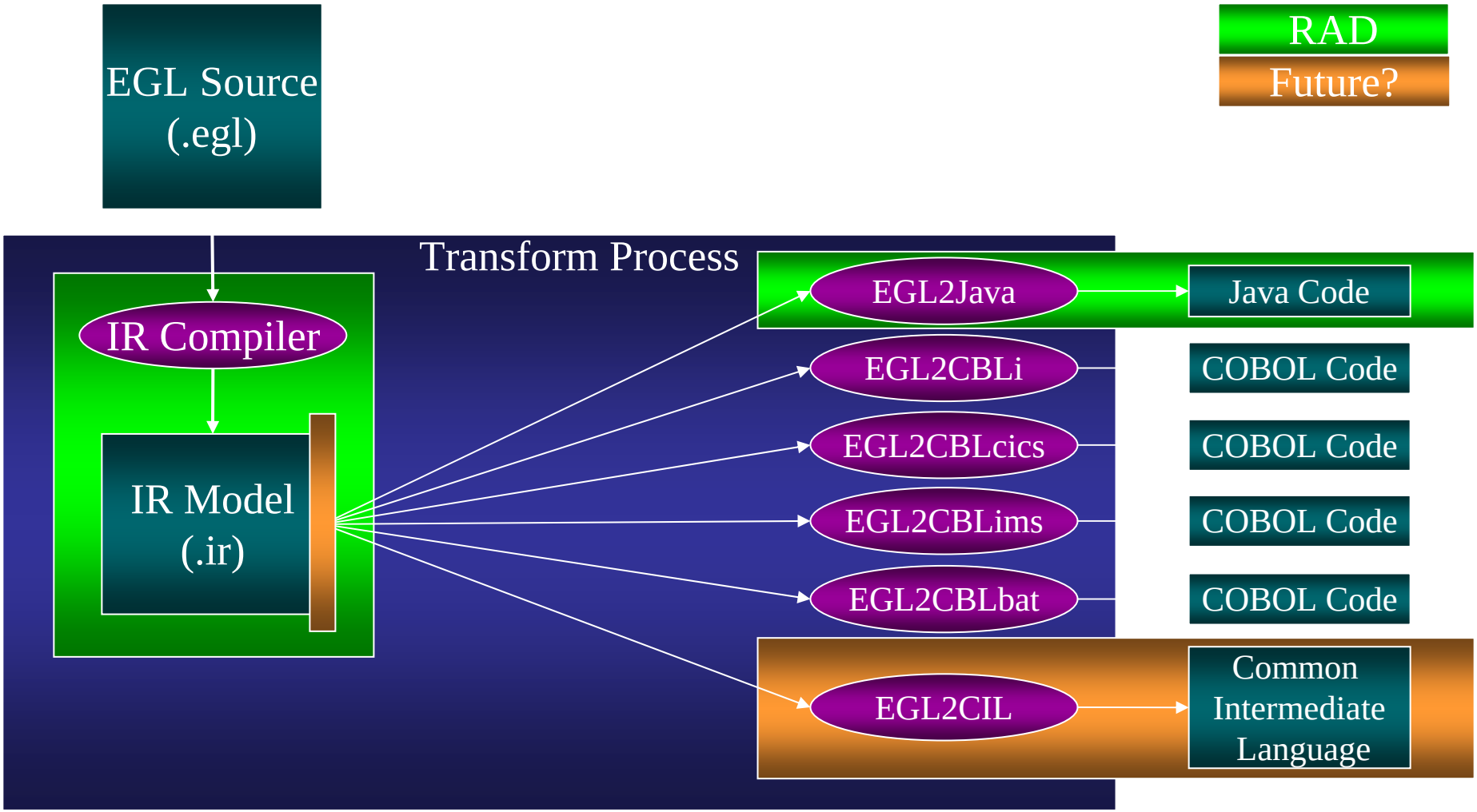
Agenda

- Purpose of an EGL standard
- EGL as a PIM for Transformation
- Discussion – next steps

General approach of EGL

- Provide a simple core language
- Provide a way to tag language elements with meta data
- Use these tags to represent complex semantics
  - Mapping Types to a Database
  - Binding of data to UI elements with validation and formatting
- Allows programmer to simply state semantics without forcing platform or middleware implementation choices
  - Same meta data can be applied in multiple contexts
- Transformation engine understands how to use meta data in mapping to a given runtime
  - Target language and platform leveraged to implement the defined semantics
- Conceptually similar to UML tags and stereotypes when used in transforming models into code

EGL Transformation Process



Purpose of an EGL Standard in ADM

- Transformation is a key aspect of “modernizing” existing applications
- Transformation from one PSM to another PSM is a many to many proposition
  - Difficult to manage – not cost effective
  - Difficult to get group involvement in any particular transformation
- Transformation to a language which is a PIM makes it a hub and spoke problem: PSM → PIM → PSM
  - Focuses transformations to and from a standard model
  - The PIM needs to be a full programming language
  - EGL is a PIM
- Many companies that do transformations invent their own languages to solve this same problem

PIM Requirements

- Needs to be a full programming language
  - Includes semantics as well as syntax
  - Future maintenance on transformed elements is done on the PIM
- Can easily represent common sources of legacy transformations
  - Semantic shift from legacy language to PIM should not be too great
  - Models common concepts in all PSMs – similar in spirit to lower level KDM concepts
    - Types, Actions, Packages, etc
- Can be annotated/profiled
  - Declarative metadata drives complex transformations and keeps the implementation of a given abstraction out of the source code.
- PIM can be extended/restricted
  - Not all model elements required to be implemented in transformations
  - PIM language itself used to define the model extensions
  - PIM defines standard points of extension

EGL as a PIM

- If EGL is to be a standard PIM then it must satisfy the basic PIM requirements
  - ✓ It is a programming language
  - ✓ It has concepts and syntax to deal with common sources of legacy transformation.
    - ✓ Data types and structures – typically where language interaction breaks
    - × No pointers or explicit memory management – a problem?
  - ✓ It has the concept of Stereotype and Annotation used to decorate declarations with metadata.
  - × The transformation engine is not extensible today
  - × EGL itself does not define the meta model

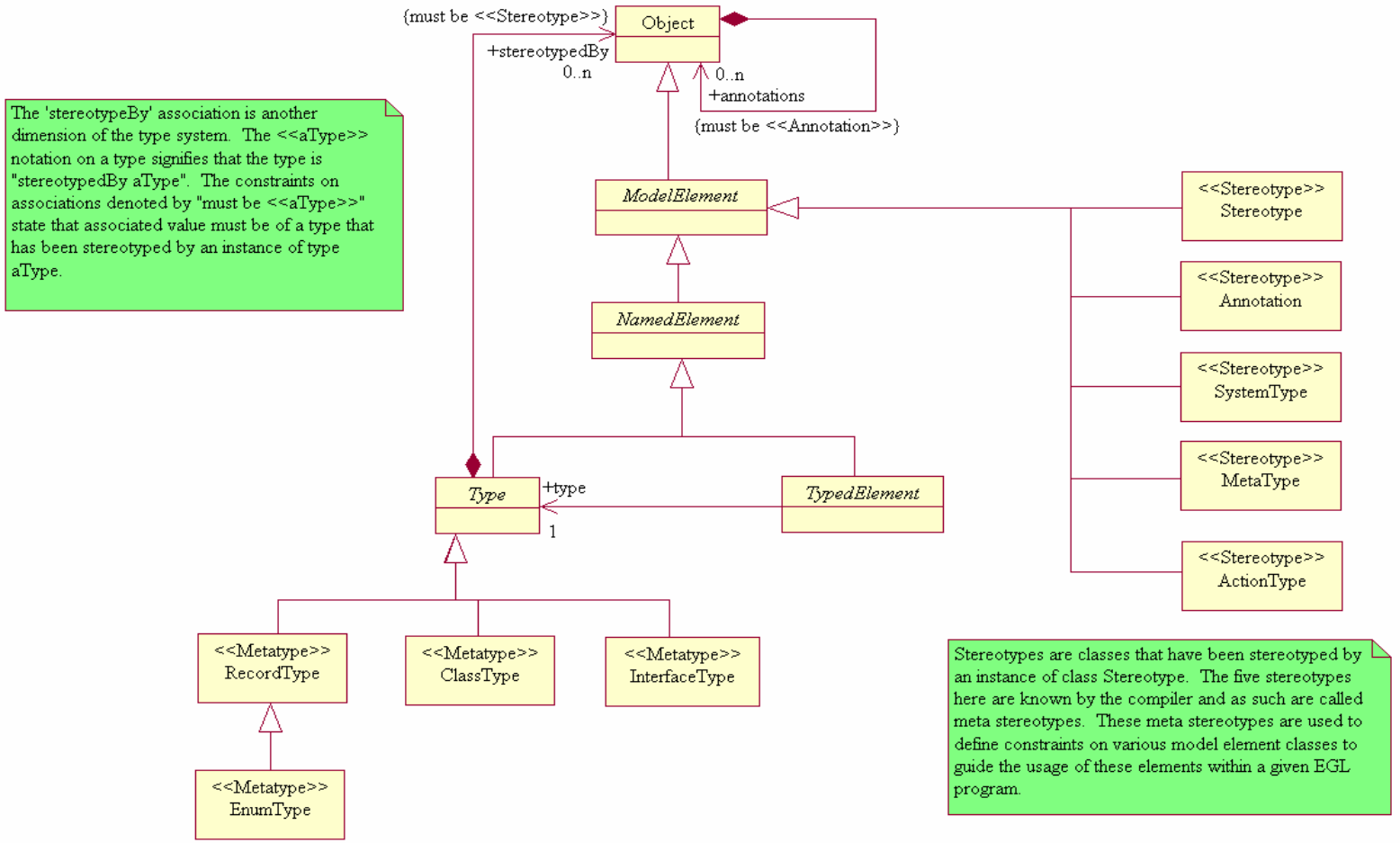
EGL as a PIM – Language Extensibility

- Original intent
  - Standardize EGL as is – includes standard way to add new stereotypes and annotations to affect transformations
  - However, this does not allow third parties that need OO concepts to use EGL as a PIM
- Current intent
  - Define *EGL kernel* as basis for "family" of languages
  - Core extension mechanism is Class and Stereotype
  - Meta Model defined reflectively by EGL itself using Classes and Stereotypes
  - The kernel meta model is based on Class but language extensions based on the kernel need not surface OO concepts
    - EGL as defined today would be such an example

Language Extensibility – Extension points

- Completely general extensibility cannot work
- Instead three points of extension are proposed
  - New *Classifiers* – new forms of user defined types
  - New Metadata types - Stereotypes and Annotations
  - New *Actions* – ACTION statement operands
- Syntax is not extensible
  - All extensions must be handled by closed syntax
  - Stereotypes are used to apply semantic information so that many compiler checks can be handled as semantic rather than syntactic checks

Language Extensibility – Type Meta Model



EGL Source that defines meta model

```
package egl.kernel;

// EGL Type Meta Model
class Object
    annotations Annotation[];
end

abstract class ModelElement
end

abstract class NamedElement extends ModelElement
    name string;
end

enum AccessModifier
    public = X"00000000";
    private = X"00000001";
    protected = X"00000002";
end
```

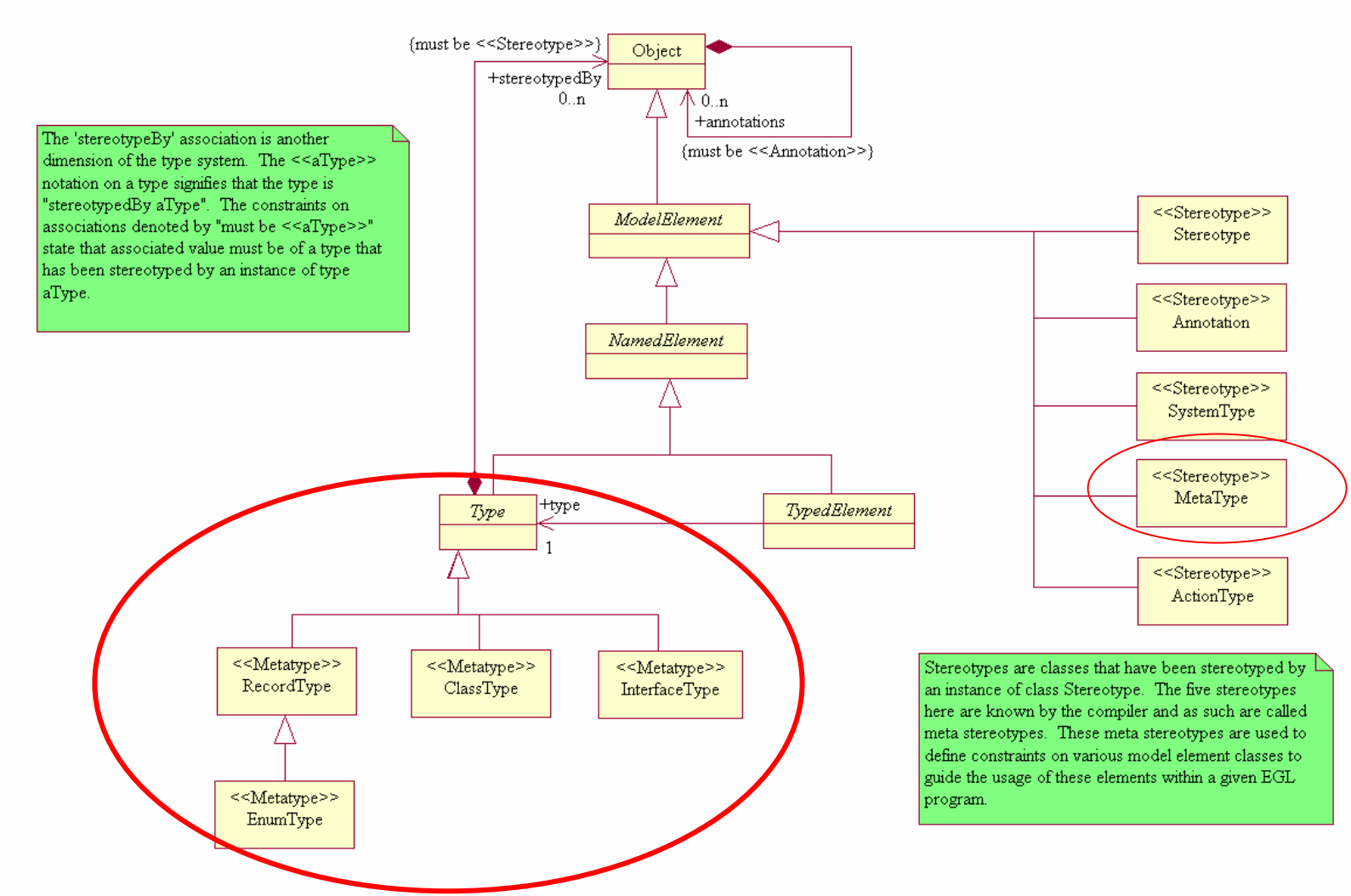
```
abstract class Classifier extends NamedElement
    accessModifier AccessModifier;
    stereotypedBy <<Stereotype>>;
    fields Field[];
end

class ClassType extends Classifier type Metatype {
    keyword = “class”;
    typeKind = TypeKind.Reference;
    memberKinds = [ MemberKind.All ];
    hasSuperType= yes;
    hasInterfaces = yes;
    requiresMain = no;
}
    classModifiers ClassModifier[];
    superType ClassType;
    interfaces InterfaceType[];
    constructors ConstructorMbr[];
    functions FunctionMbr[];
    operations OperationMbr[];
end
```

EGL Source for Meta Stereotypes

```
package egl.kernel;

// Meta Stereotype definitions
class Stereotype extends ModelElement type Stereotype {
    targets = [ ClassType.type ];
}
    targets Classifier[];
    memberAnnotations <<Annotation>>[];
    mutualExclusions <<Annotation>>[];
    associations <<Annotation>>[];
end

class Annotation extends ModelElement type Stereotype {
    targets = [ ClassType.type ];
}
    targets ModelElement[];
end
```

```
class Metatype extends ModelElement type Stereotype {
    targets = [ Classifier.type ];
}
    keyword string;
    typeKind TypeKind = TypeKind.Reference;
    isStaticType boolean = no;
    memberKinds MemberKind[] = [ MemberKinds.All ];
    hasSuperType boolean = no;
    hasInterfaces boolean = no;
    requiresMain boolean = no;
end

class SystemType extends ModelElement type Stereotype {
    targets = [ RecordType.type, ClassType.type ];
}
end
```

Language Extensibility – Classifier extension



Adding new Classifiers

- Languages based on EGL kernel must be free to choose relevant set of Classifiers
- EGL Kernel defines set of specific Classifiers which have very specific syntax
- Semantics for all Classifiers are governed by the *Metatype* stereotype.
- All new Classifiers are syntactically similar to ClassType but constrained by an instance of the Metatype stereotype

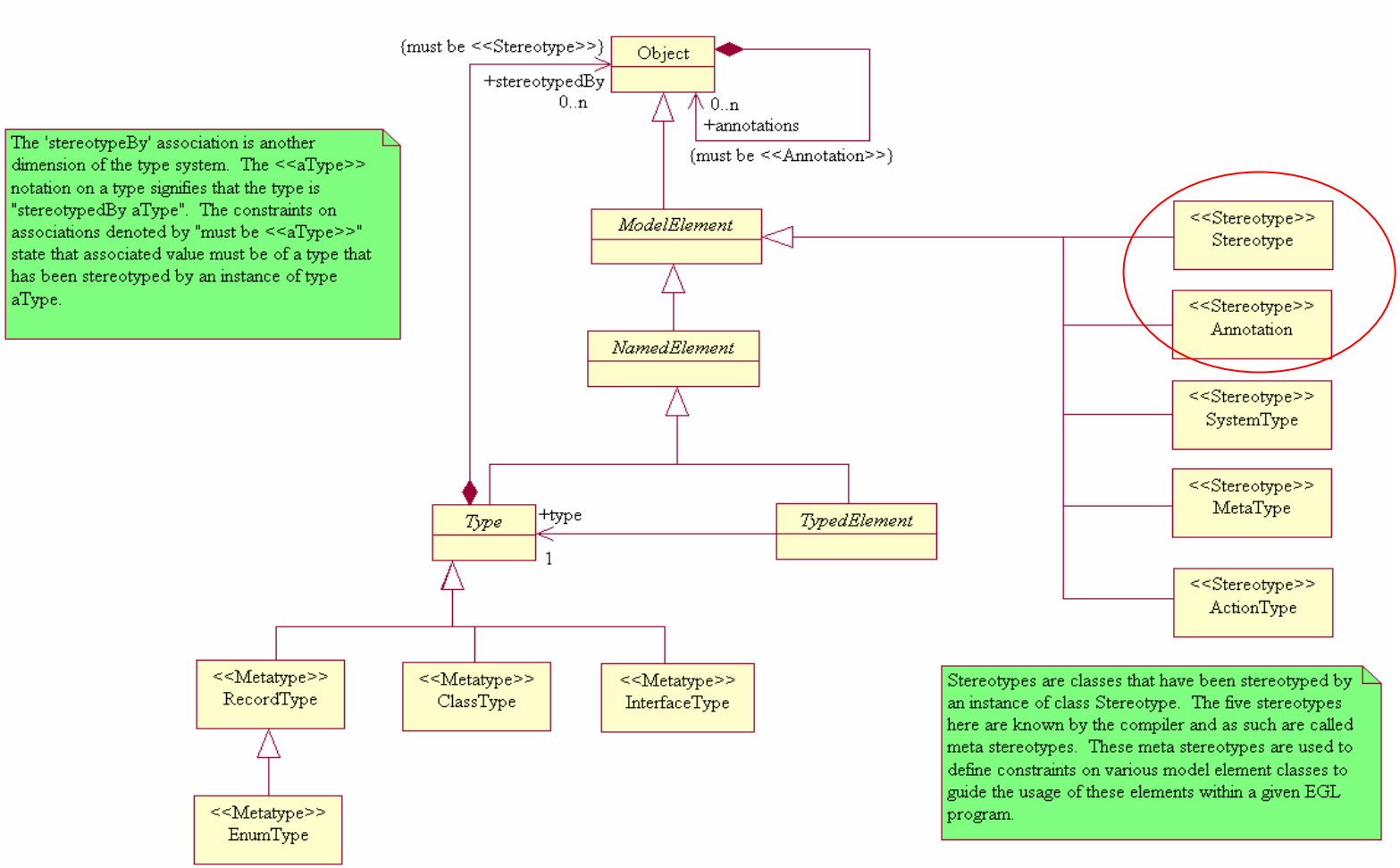
Classifier Extension Example – EGL Program type

```
package egl.core;

class ProgramType extends Classifier type Metatype {
    keyword = "program",
    memberKinds = [ MemberKinds.FieldMbr, MemberKinds.FunctionMbr ],
    isStaticType = yes,
    requiresMain = yes
}
    fields FieldMbr[];
    functions FunctionMbr[];
end
```

Registering Classifiers

- Languages based on kernel register to the compiler the set of Classifiers available.
- The ‘keyword’ values associated with the given classifier metatype information tell the compiler how to treat declarations of the given classifier
- This works because the basic syntax of all extended classifiers is the same except for the initial keyword
  - Metatype info used to semantically check the declaration

Language Extensibility – Meta data types

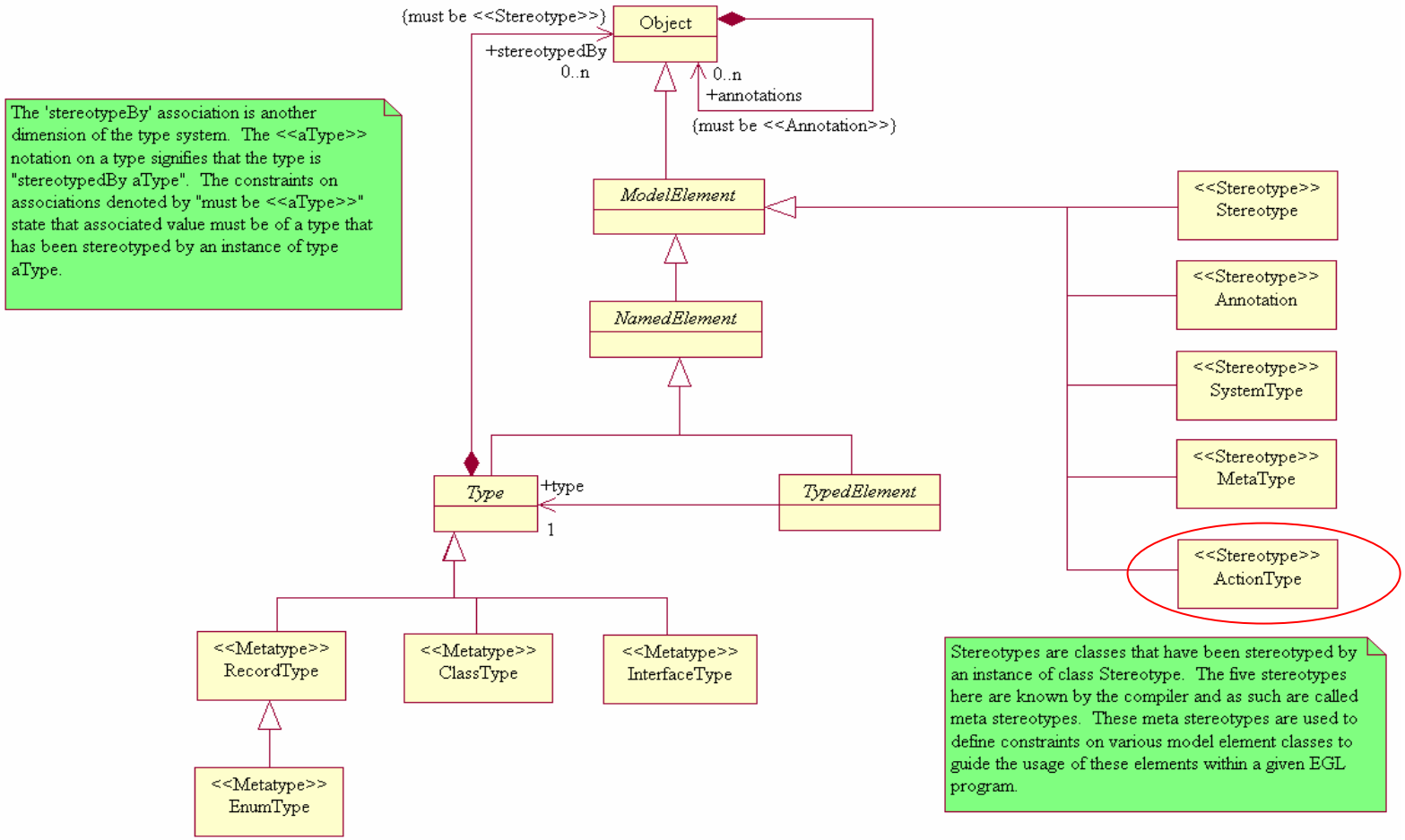


Example Stereotype – EGL SQLRecord

```
package egl.core.io.sql;

class SQLRecord type Stereotype {
    targets = [ egl.kernel.RecordType.type ],
    memberAnnotations = [ ColumnName.type, ..],
}

    tableNames String[][];
end

class ColumnName type Annotation {
    targets = [ egl.kernel.FieldMbr ]
}
    value string;
end
```

```
// Example usage

record Employee type SQLRecord {
    tableNames = [ [ "T1", "Employee" ] ]
}
    employeeNumber char(6) { @columnName{ "EMPNO" } };
    lastName string;
    firstName string { columnName = "FIRSTNME" };
.   ..
end
```

Language Extensibility - Action extension



Extending the set of Actions - TBD

- EGL Kernel defines a set of standard Actions
  - Add, Delete, Get, Replace, Open, Close, Converse
- Actions have abstract semantic which is made concrete through the use of stereotyped operands

```
get anEmployee;
```

- Extension is based on adding new stereotypes and adding transforms that understand the metadata
- Should the set of Actions be extensible?
  - Syntax issues more difficult to deal with

Discussion

- Relationship to KDM, GASTM
  - Standard transformations from EGL model
  - How are Stereotypes and Annotations expressed?
- What are the full requirements of a PIM in the context of ADM Transformation?