

# Local Type System, Version 1.0

## **URL**

<http://www.wiley.com/compbooks/poole/patterns/LocalTypeSystem.pdf>

## **Contributor**

John Poole (jpoole@alum.poly.edu)

## **Structural Classification**

Micro pattern.

## **Usage Category**

Structural, Typing.

## **Intent**

This pattern provides a simple, consistent way of organizing Data Types that have been defined locally within a given model.

## **Also Known As**

TBD

## **Motivation**

In situations where a number of Data Types are defined within a model for the purpose of typing the StructuralFeatures of Classifiers, the Data Type instances can be organized more effectively if they are culled together into a single, named TypeSystem. For convenience and uniformity of structure, this TypeSystem should then be placed within the most immediate Package whose Classifiers' instances utilize the TypeSystem's Data Type instances.

Note that, in general, the explicit modeling of Type Systems makes the use of modeled Data Types less ambiguous. For instance, if a given TypeSystem instance models a well known real-world type system, such as that of CORBA IDL or the Java programming language, there is a reference-able definition of those types that can readily be referred to. Type System modeling also facilitates meta data re-use. For example, a single model of a particular language's data types can be developed, published, and re-used by all models within a particular interchange environment. (In this case, though, the Local Type System pattern doesn't always apply, since Type Systems are shared, not always defined locally, but to the extent that a Type System might be defined within an interchange model, an attempt should be made to follow this pattern).

## **Applicability**

This micro pattern is used in the construction of any Domain pattern that requires local modeling of data types.

## **Projection**

The Local Type System pattern is based on a sub-graph of the CWM metamodel consisting of the metaclasses

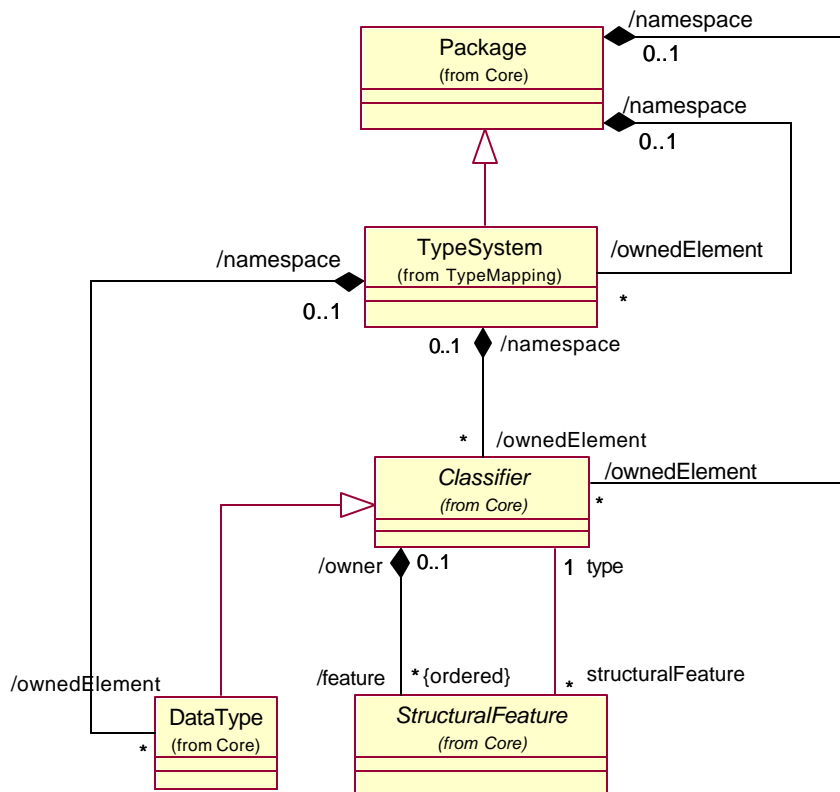
- `org.omg.cwm.objectmodel.core.Package`
- `org.omg.cwm.objectmodel.core.Classifier`

- org.omg.cwm.objectmodel.core.StructuralFeature
- org.omg.cwm.objectmodel.core.DataType
- org.omg.cwm.foundation.typemapping.TypeSystem

and the associations

- org.omg.cwm.objectmodel.core.ElementOwnership
- org.omg.cwm.objectmodel.core.ClassifierFeature
- org.omg.cwm.objectmodel.core.StructuralFeatureType

This sub-graph is illustrated in the diagram below:



**Figure 1: Local Type System Projection**

## Restriction

Restrictions on instances of the projection are as follows:

A *locally defined* Type System instance that provides Data Type instances to StructuralFeatures of Classifier instances defined within a given model is owned by the *nearest* or *most-immediate* Package instance in the recursive Package containment hierarchy of those Classifier instances.

This restriction is formally expressed by the following OCL constraint:

TBD: Provide an OCL constraint defining this restriction (a non-trivial constraint to formulate). It needs to assert that the TypeSystem instance is not imported, and not owned by some Package residing outside the recursive containment hierarchy of the Classifier instances that use it. This implies that the Type System

could not possibly be defined externally to the Classifiers' containment hierarchy (i.e., it could not be anything other than a locally defined Type System).

## **Usage**

When processing an imported XMI model, the importing process checks each Package instance in the model for the presence of any locally-defined TypeSystems. If such TypeSystem instances have been defined, each StructuralFeature of each Classifier instance within the Package is tested for typing by one of the local Type Systems' DataType instances. (Note that each such StructuralFeature must also be tested for typing by external DataTypes, if the import is expecting this). When writing an XMI file for export, the exporting process simply stores all locally defined TypeSystem instances in their most-immediate Package.

In essence, this pattern simply provides you with a convenient and well-known place to park your DataTypes. It does not otherwise affect or modify their meaning, nor how DataTypes and typed StructuralFeatures are to be interpreted.

This pattern does not apply to DataTypes and TypeSystems defined in *external* Packages; i.e., Packages that are not a part of the recursive containment hierarchy of some set of Classifier instances. Nor does it apply to Packages whose owned DataTypes or TypeSystems are imported into some Package of the recursive containment hierarchy of the Classifier instances.

## **Parameters**

TBD

## **Commentary**

None.

## **Consequences**

TBD

## **Known Uses**

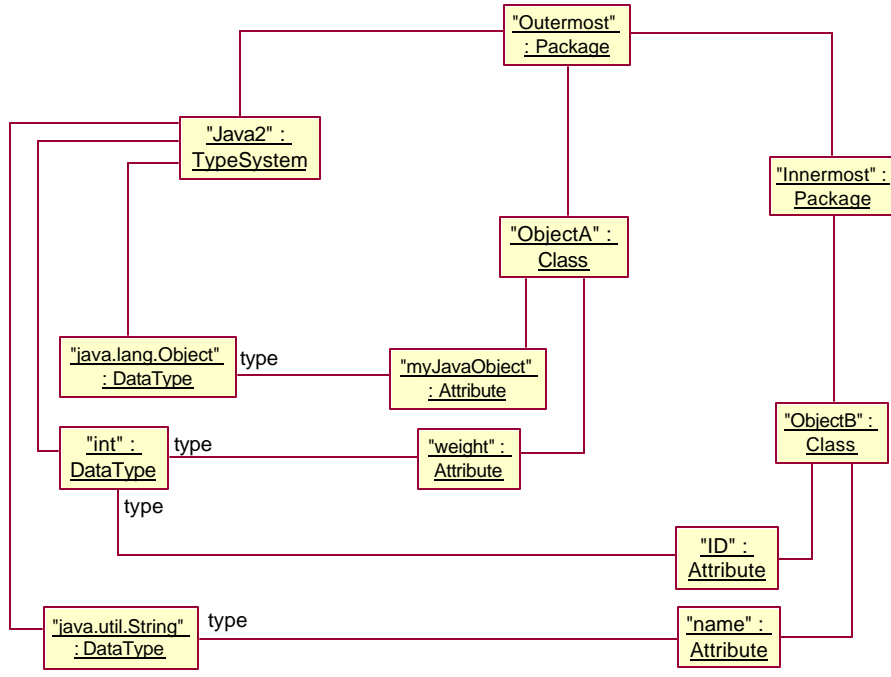
TBD

## **Related Patterns**

TBD

## **Sample Solution**

The instance diagram below illustrates an occurrence of the Local Type System pattern. In this example, a partial model of the Java 2 type system is defined, consisting of the types Object, int, and String. The Attributes of a number of CWM Classes use the Java types as their data types. These Classes are instantiated within two Package instances comprising the model. The Java 2 Type System instance is contained by the nearest Package in the containment hierarchy of the model.



**Figure 2: Local Type System Example: Instance Diagram**

TBD: Provide XMI fragment

**Figure 3: Local Type System Example: XMI Fragment**