Workflow Example

1 Introduction

This design document describes a simple workflow example that demonstrates the usage of UMLX to define a workflow application. The defined workflow is less important than the ability to define a useful workflow. Given this ability, other workflows may be easily defined.

Note that this example provides excellent opportunities for confusion of meta-levels, since transforms are defined that invoke transforms, and the individual transforms comprise workflows (transformation sequences) in order to define the overall workflow application.

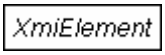
The workflow supports repeated interactive application of a selected library transform to an XMI model. The workflow therefore starts by 'prompting' for the file name containing the model, then enters the main recursion in which a list of available transforms is provided, and from which a chosen transform is selected. If no transform is selected, the recursion terminates, otherwise the selected transform is applied. Once the recursion terminates, the user is 'prompted' for a file name in which to save the model.

The workflow diagrams have much in common with UML activity diagrams, so could be trivially redrawn with considerable loss of semantic accuracy. This is left as an exercise for readers who need precise UML 1.x when we are discussing UML 2.x and QVT behaviours.

2 Information Models

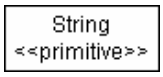
The Information model is partitioned into 4 'packages'; XMI, UML, UMLX, Files and Schema, of which the first 4 are externally defined in practice, but internally defined for the purposes of this example.

2.1 XMI



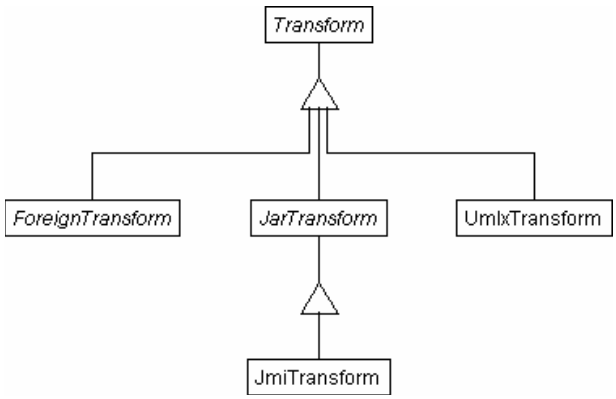
We are applying transforms to complete models so we need no detailed XMI structure, merely the abstract element from everything inherits.

2.2 UML



The only part of UML that is needed is a String value.

2.3 UMLX



The only part of the UMLX information model we need is the definition of invokable transforms, all inherited from the abstract `Transform` behaviour.

`ForeignTransform`s require a separate process to be invoked with communication via files or sockets. Derived transforms will need to establish these policies.

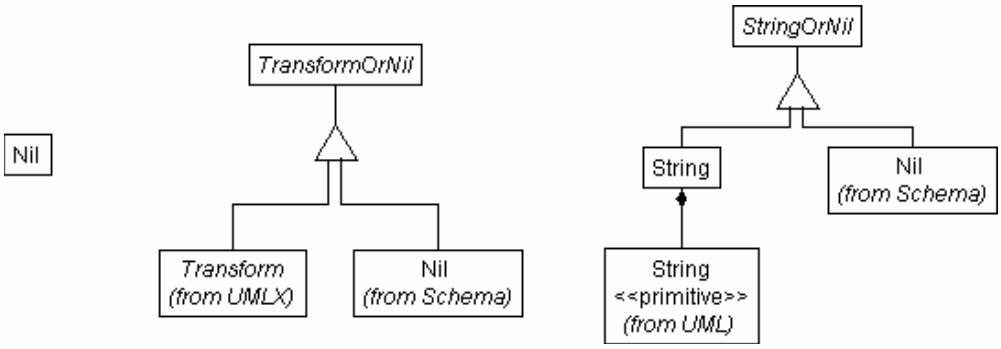
`JarTransform`s can be invoked using standard Java class path resolution. `JmiTransform` is a suggestion of one protocoil that may be sufficiently standard to need no further derivation.

`UmlxTransform`s are defined in UMLX and so may be invoked within the transforming execution environment.

2.4 Files

The detailed modelling of a `FileContext` describing the location of an opened file and the textual context of a `TextFile` is not defined yet. There is surely some standard.

2.5 Schema



The re-used information models are incorporated in the information model for the example.

`Nil` is a trivial void type. This might be part of UML if I studied it more carefully.

`String` is a class containing a UML `String` value.

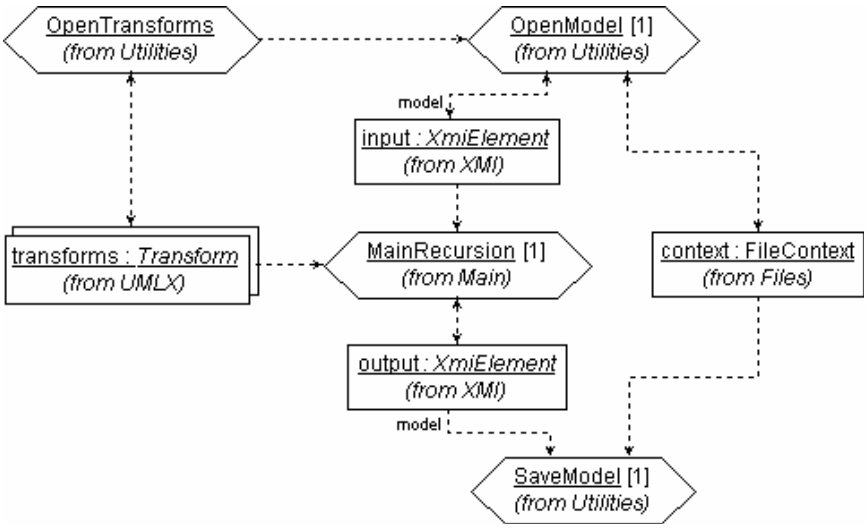
The abstract `StringOrNil` and `TransformOrNil` support successful/failed returns of a `String` or `Transform` from interactive transforms.

3 Transform Models

The Transformation model is partitioned into 4 'packages'; Main, Utilities, Interactive and <<BuiltIn>>, of which the Interactive at least should be a standard library. The top-level transformation is Main.Main, and like Java is just one of many possible starting points.

3.1 Main

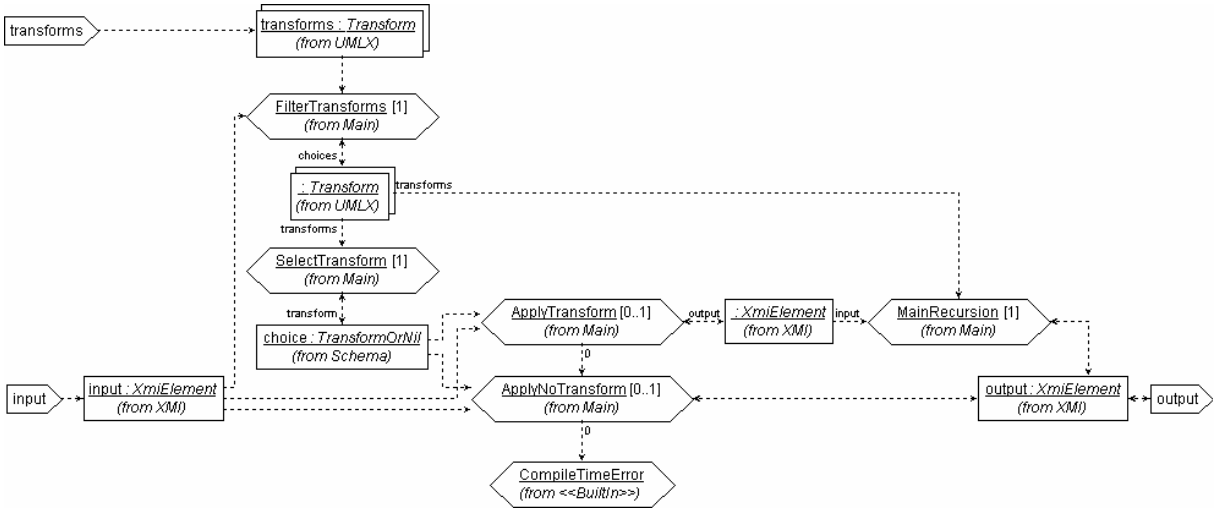
3.1.1 Main.Main



The main transform invokes `OpenTransforms` to identify the libraries of available transforms and then `OpenModel` to acquire the model which is then processed by the `MainRecursion`, whose result together with the context of the `OpenModel` is passed to the `SaveModel` prompter.

This transformation is unusual in having no input or output ports. It operates as information appears within the Open transforms and disappears within the Save.

3.1.2 Main.MainRecursion



The `MainRecursion` first invokes `FilterTransforms` to identify the subset of the `transforms` that are applicable to the `input`, and then invokes `SelectTransform` to obtain an interactive choice of none or one transform. This choice is then passed to `ApplyTransform` and `ApplyNoTransform`. If `ApplyTransform` is successful its output is passed to MainRecursion for further activity. If `ApplyTransform` is unsuccessful `ApplyNoTransform` provides the result and the recursion terminates.

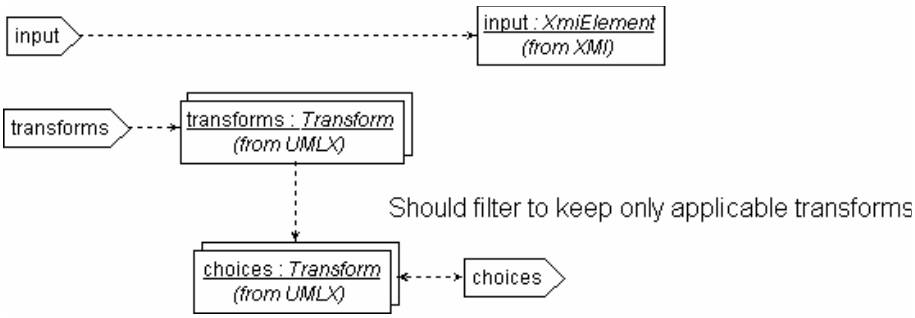
[The transformation cardinalities shown as `FindTransforms [1]` are not really necessary here, since everything can only execute once. The choice between `ApplyTransform` and `ApplyNoTransform` exactly corresponds to the `TransformOrNil` alternatives, so exactly one path matches. The 'else' arc from `ApplyTransform` to `ApplyNoTransform` is therefore

unnecessary. The sole benefit of the explicit rather than default cardinalities is to impose the intended behaviour so that run-time and preferably compile-time diagnostics can detect that `CompileTimeError` is something that does not occur by design, rather than something to be invoked when the impossible happens.

Matches have the unpleasant property that holes in the match coverage cause nothing to happen silently. `CompileTimeError` is my current idea on how to declare where holes are unintended and to be diagnosed.

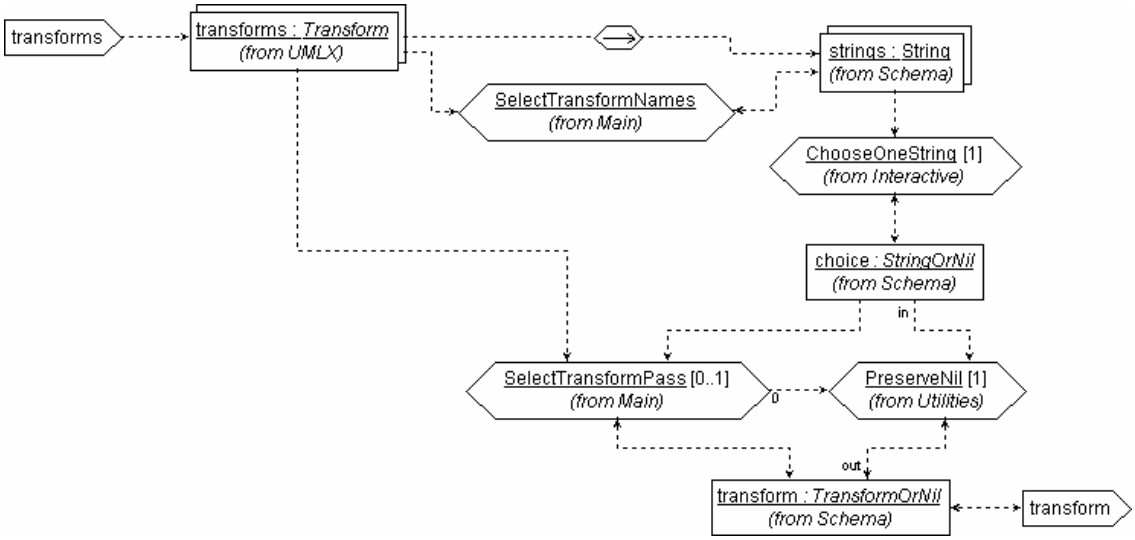
For instance, if a third derivation of `NilTransform` was added to `TransformOrNil`, the default cardinalities would fail to detect a match and so just do nothing. With the explicit cardinalities, by analogy with compilers that warn about missing enumeration cases in switch statements, we can diagnose any possibility of the `TransformOrNil` failing to match since we drop through to an explicit default.]

3.1.3 Main.FilterTransforms



For the time being all transforms are deemed applicable.

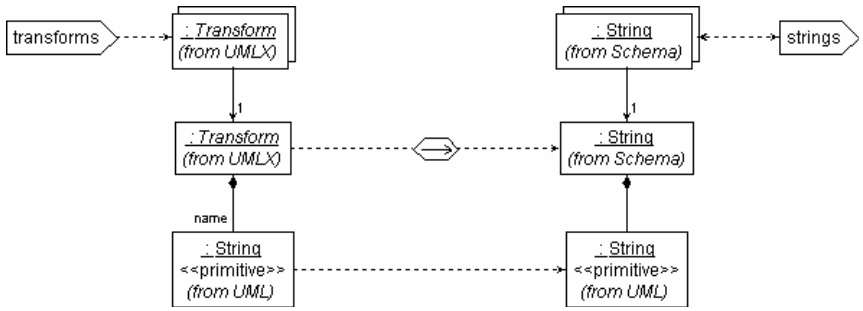
3.1.4 Main.SelectTransform



The collection of `transforms` is converted to a collection of `strings` for use by the `ChooseOneString` interactive transform. The chosen string or `Nil` is then passed to `SelectTransformPass` which identifies the chosen `transform` for return, or to `PreserveNil` which ensures a `Nil` return.

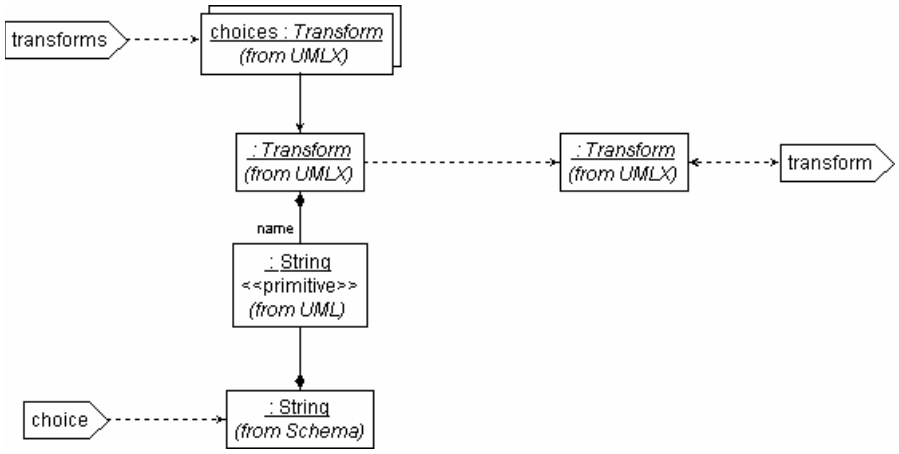
[A smarter chooser might accept a hierarchical model of transform names with description fields.]

3.1.5 Main.SelectTransformNames



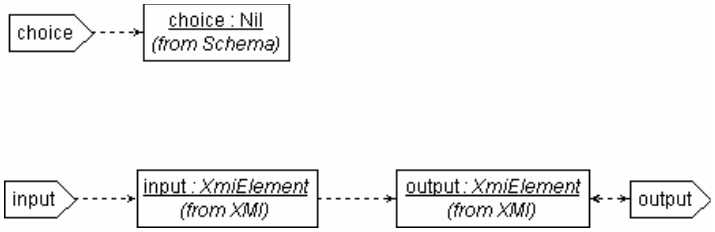
This transform matches once for each `Transform` and evolves a `String` entry in `strings` output for the transform `name`.

3.1.6 Main.SelectTransformPass



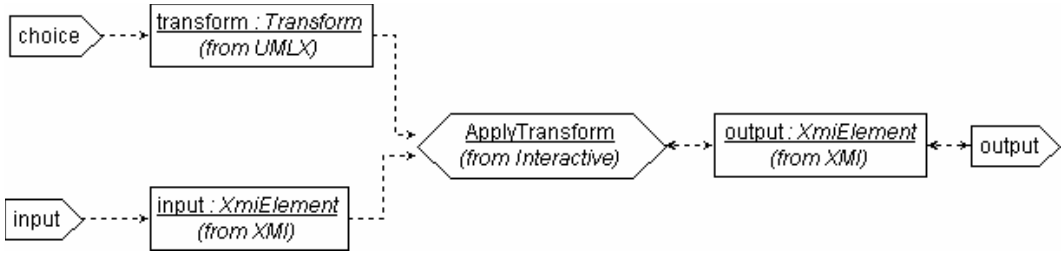
The returned `transform` is the one whose `name` matches the `choice`.

3.1.7 Main.ApplyNoTransform



When the `choice` is `Nil`, the `input` is preserved as the `output`.

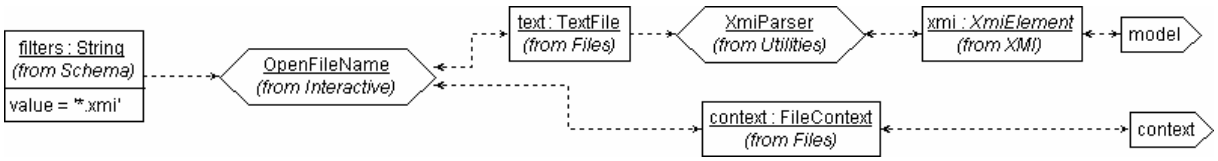
3.1.8 Main.ApplyTransform



When the `choice` is a `Transform`, it is applied to the `input`.

3.2 Utilities

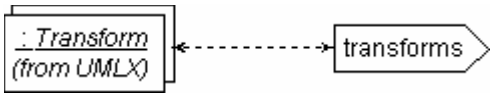
3.2.1 Utilities.OpenModel



The interactive `OpenFileName` transform is invoked to prompt for a file name, with assistance of a `filters` string to restrict names in a browser.

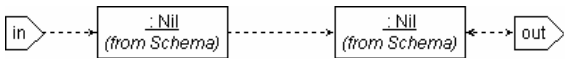
In principle `OpenFileName` should return a naked text file, which then gets parsed to XMI, however when using XSLT, reading and parsing are almost one operation, so this bit is left a bit vague for now.

3.2.2 Interactive.OpenTransforms

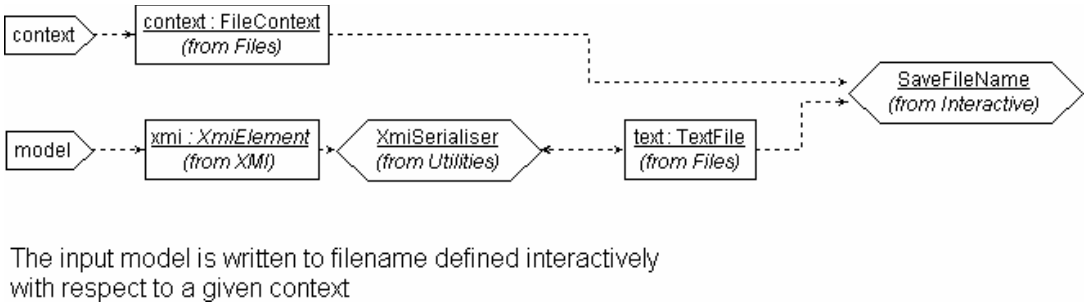


TBD, presumably using some form of TRANSFORM_PATH environment.

3.2.3 Utilities.PreserveNil



3.2.4 Utilities.SaveModel



Again a bit vague since XSLT does XML (and consequently XMI) serialisation almost for free.

3.2.5 Utilities.XmiParser

TBD, and probably in 'Java' rather than UMLX for quite a while.

3.2.6 Utilities.XmiSerialiser

TBD, and probably in 'Java' rather than UMLX for quite a while.

3.3 Interactive

The Interactive transforms represent built-in behaviours that interact with a user. The means of interaction, browser, pop-up or command line is unspecified, merely the source and target information models.

The names and behaviours here are just examples. It would be beneficial to align them to any standard workflow libraries or standards that already exist.

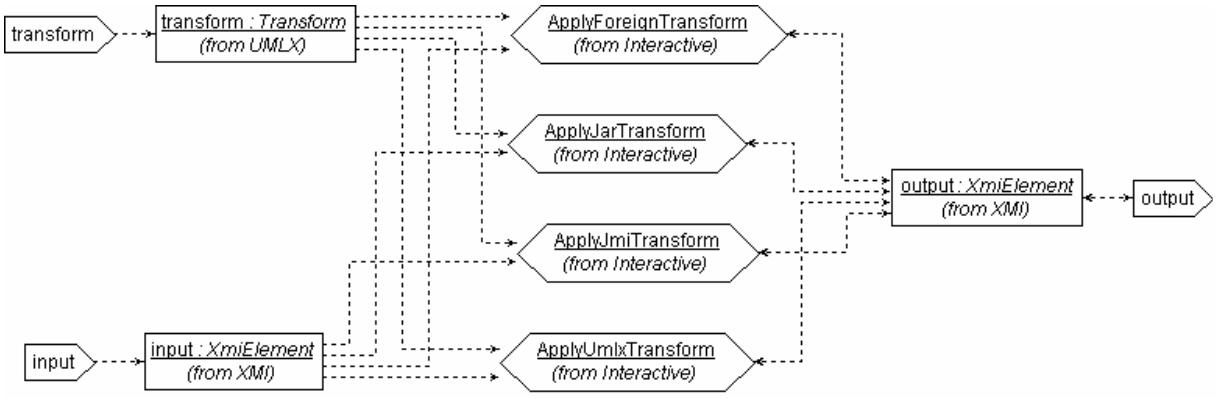
As built-in transforms, these behaviours should all be hand coded to establish interfaces between the transformation environment and a controlling GUI environment.

For use with XSLT these transforms will be public static Java functions taking a number of 'model' arguments and returning one 'model'.

3.3.1 Interactive.ChooseOneString

Return a particular String or Nil reflecting a user choice from a set of strings

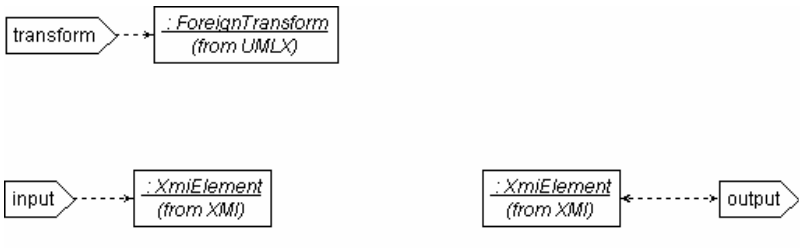
3.3.2 Interactive.ApplyTransform



Apply a transform to a model

This performs a polymorphic dispatch to the appropriate transformation technology.

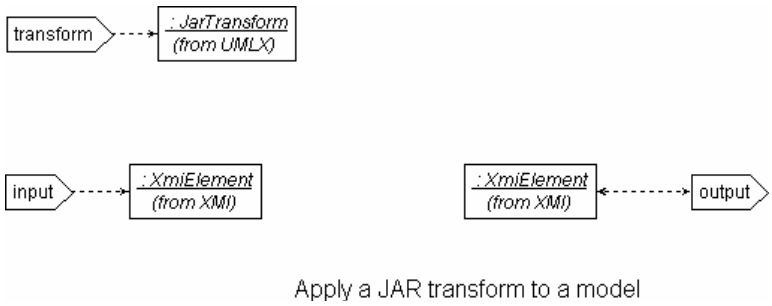
3.3.3 Interactive.ApplyForeignTransform



Apply a foreign transform to a model

For foreign transforms, realised by frozen code, interfaces will be necessary to transport the input and output models to the process activated to perform the transform, probably via files, with proprietary formats, before eventually forking off a process to execute the program that implements the transform.

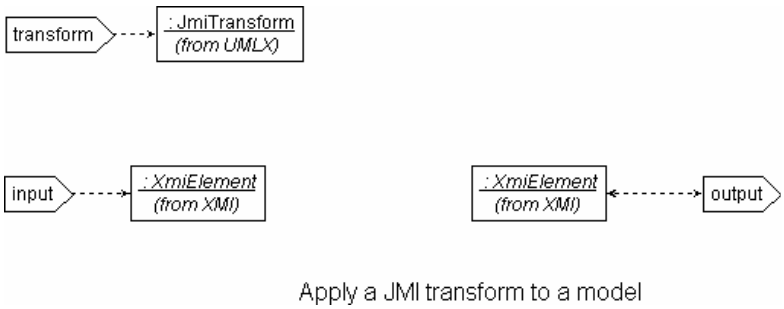
3.3.4 Interactive.ApplyJarTransform



Apply a JAR transform to a model

For external transforms, for which a suitable, typically Java, interface can be established, interfaces will just be required to transport the models within the current process.

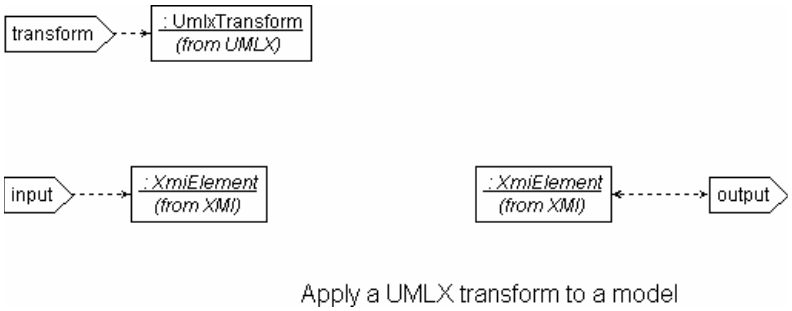
3.3.5 Interactive.ApplyJmiTransform



Apply a JMI transform to a model

For external transforms, for which a structer well-defined JMI interface has been established, interfaces will just be required to express the models as JMI within the current process.

3.3.6 Interactive.ApplyUmlxTransform



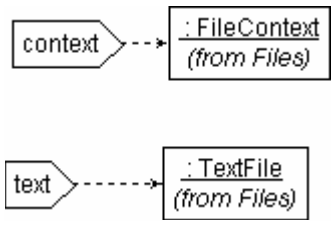
Apply a UMLX transform to a model

For internal transforms, for which a meta-model of the transformation is available, typically UMLX for now, no interfaces may be required at all, it is just necessary to pass inputs and outputs within the transformation process implementing the workflow application.

3.3.7 Interactve.OpenFileName

A text file is read with interactive assistance that may make use of a filters string to be more user-friendly about the offered selections. The context of the open is returned to enable a subsequent save to share the directory context of the opened file.

3.3.8 Interactive.SaveFileName



The text file is saved of with interactive assistance that may use a file context to be more user friendly about its defaults.

3.4 <<BuiltIn>>

3.4.1 <<BuiltIn>>.CompileTimeError

The compilation system should diagnose any context in which this transformation can be executed, and output a textual parameter as part of a diagnostic.