also built a custom GAAST API for the Java 2 Micro Edition MIDP [11], using JavaDoc [12] tags as annotations; we use this custom GAAST API for supporting model-driven development of Java mobile applications [13].

of type enumeration, are used to decorate the methods of this web service.

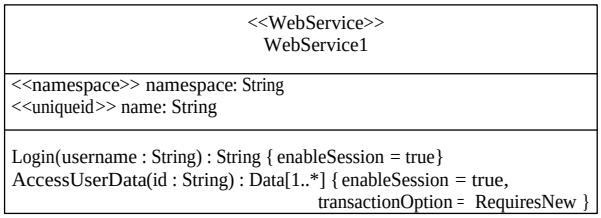


Fig. 1. Modeling using UML profiles

Compared to a corresponding more abstract model that does not contain any

on how the final software will be like. However, just selecting the language, still says nothing about issues such as sessions and transactions, which are to be implemented in the later stage of tag interpretation.

A specific transformer may combine all three stages: mapping types, mapping marks to language constructs, and interpreting the later, into a single pass. However, it makes sense to separate them, when the automation is not fully complete, for example, when modeling is used only for defining the high level architecture of an application. For example, in an EJB [6] application we prefer to model beans and their interactions, but it is easier to write and maintain the

```
1. webservice "WebService1"
2. {
3.  namespace namespace : "www.tu-darmstadt.de"
4.  uniquieid name : "Simple Service"
5.
6.  enableSession Login(username : String) : String
7.  enableSession transactionOption.RequiresNew
8.       AccessUserData(id : String) : Data[1..*]
9. }
```

(3)

would be mapped with pseudo-syntactic marking. The «WebService» stereotype is mapped to the predefined class WebService, from which WebService1 inherits. We use session

ify/transform some AST-like source- or binary-level representation of the annotated program. For instance, tags used in MOF/UML, or JavaDoc [12] attributes used in Java, can be interpreted, when we manipulate the model, respectively the source code. The .NET platform, on the other hand, also allows run-time interpretation of tags. Interpreting tags at run-time allows to postpone the decision about the semantics until execution time. However, it

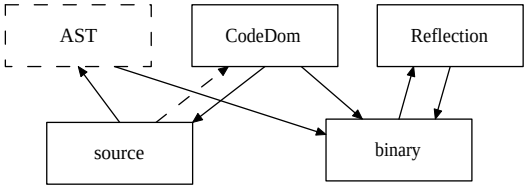
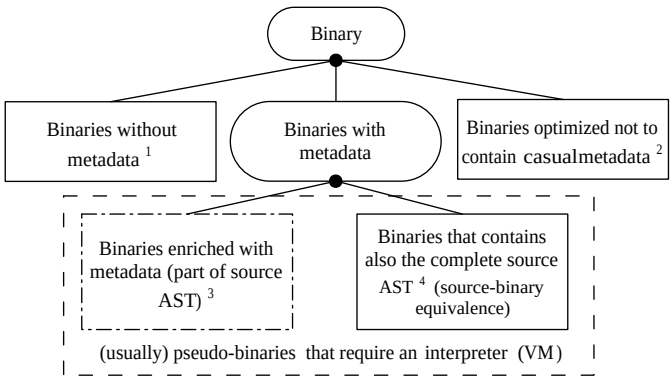


Fig. 6. .NET AST-Like Program Representations

stage of the compiler. Fig. 7 shows how this infrastructure can be used for interpreting attributes in .NET. Attributes are full-status types defined as classes; they are part of the type system and can also be marked with other attributes. Only a set of predefined attributes is interpreted by the .NET framework itself. User-defined attributes have no semantic meaning to .NET. The first action in using custom attributes in .NET consists in defining new attribute classes, if needed, and using them for decorating the code. In the next step, either a source level AST generated via some `CodeDom` implementation,

program. In .NET, this AST is not available to the programmer. We showed it in Fig. 6 to emphasize the similarity between the different .NET AST-like representations of the program (aimed at supporting program transformations) and the source AST built by the compiler: .NET transformation related ASTs



[1] E.g. DOS EXE files, contain "very few" metadata
[2] E.g. DOS BIN files, or specially encrypted EXE files have almost no "casual" metadata
[3] E.g. Java class files, or .NET assemblies, and at some extent COM TLB files
[4] E.g. any interpreted language, where the source is not compiled, but is reused every time

Fig. 8. GAAST Relation to Metadata

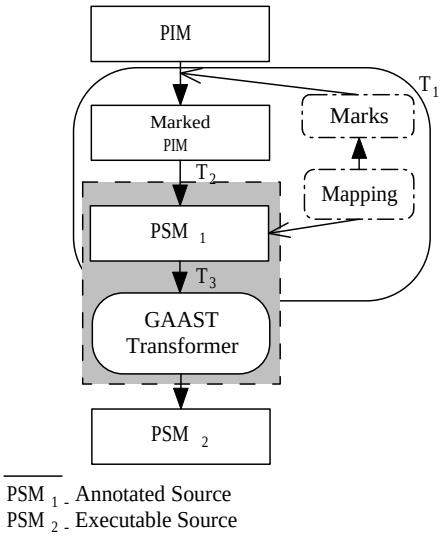


Fig. 10. MDA GAAST PIM-to-PSM Transformations

ping of the extended HUTN notation for the web service in C# language[10] which is used to represent a GAAST-like language with explicit support for tagging in the form of attributes.

```
1. [WebService(namespace="www.tu-darmstadt.de",
```

tecture. The annotated source can express the full design architecture better than pseudo-syntactic marking without having to process method names and without having to invent many method prefixes or su□xes which make it more di□cult to understand the code.

(2)

There are however some restrictions about applications of GAAST-like mapping. First, by our discussion up to now it seems that GAAST based mapping is easily. This however requires that the metamodel of the target GAAST language must map almost directly to the UML model being transformed. In our case a general purpose object-oriented (OO) language like C# [25], whose OO metamodel maps almost directly to the class based web-service model. The transformation process may be not so clean in other cases. However, if no

GAAST API is not our ultimate goal. Nevertheless, such an experiment is in-

[35] Byte Code Engineering Library (BCEL), http://jakarta.apache.org/bcel/.

[36] S. Chiba, Load-time Structural Reflection in Java, In Proc. of ECOOP '00, Springer Verlag, LNCS 1850 (2000) 313–336.

[37] G. Kiczales and J. Lamping and A. Menhdhekar and C. Maeda and C. Lopes