

[0174] A package descriptor describes a package, but does not implement that package. No additional events are added by the package descriptor.

[0175] A version descriptor of the present invention is a feature descriptor providing details about the number of modifications that have been made to its parent. Versions include a major release, minor release, build release, and revision release. The first public release is frequently referred to as 1.0.0.0. Very minor fixes increase the revision release. Minor changes that involved changes to several other models increase the build release and reset the revision release to zero. Small but significant changes increase the minor release and reset the build and revision release to zero. Very large changes increase the major release and reset the minor release, build release, and revision release to zero. The values allowed in a version are not always numeric, instead the allowed values are determined by the data descriptors. A version descriptor includes a major release attribute descriptor, a minor release attribute descriptor, a build release attribute descriptor, and a revision release attribute descriptor.

[0176] A version descriptor has a one-to-one association relationship with a major release attribute descriptor that provides details on the major release attribute. Generally, the major release is a number and cannot be less than zero. A version descriptor has a one-to-one association relationship with a minor release attribute descriptor that provides details on the minor release attribute. Generally, the minor release is a number and cannot be less than zero. A version descriptor has a one-to-one association relationship with a build release attribute descriptor that provides details on the build release attribute. Generally, the build release is a number and cannot be less than zero. A version descriptor has a one-to-one association relationship with a revision release attribute descriptor that provides details on the revision release attribute. Generally, the revision release is a number and cannot be less than zero.

[0177] A version descriptor describes a version, but does not implement that version. No additional events are added by the version descriptor.

[0178] A hint descriptor of the present invention is a descriptor that is a group of attributes adding details to a descriptor that are not adequately captured anywhere else. Hint descriptors describe the implementation of specific hints. A hint can be added to any descriptor. A hint can be used to create a descriptor implementation which has not yet been formally defined, or which is extremely domain specific. Hints can be useful to aid the user's understanding of the model or to store values used in the automatic generation of implementations from the descriptor as described in more detail below. Also notice that since hints are part of the modeling process, no access constraints exist. The modeling tool enforces access constraints.

[0179] A hint descriptor has a one-to-many aggregation relationship with attribute descriptors that describe the attributes hint instances hold. A hint descriptor describes a hint, but does not implement that hint. No additional events are added by the hint descriptor.

[0180] A role descriptor of the present invention is a descriptor serving as a mechanism for adding many related hints to a metamodel or feature descriptor in a logical

grouping. A role descriptor may have a relationship with hint descriptors and role descriptors.

[0181] A role descriptor has a zero-to-many aggregation relationship with hint descriptors that provide details about the hints contained by the role. A role descriptor has a zero-to-many aggregation relationship with other role descriptors that allows for the inclusion of all the hints from one or more other roles to participate as hints in this role.

[0182] A role descriptor describes a role, but does not implement that role. No additional events are added by the role descriptor.

[0183] In case one of the above-described descriptors does not fully meet the needs of the modeler, some implementations of modeling tools can include a meta-metarepository feature. The meta-metarepository contains the descriptors described above displayed as metamodels. The user of the modeling tool can alter these models, or construct a new model to create a new descriptor type. This type becomes part of the ontology of the modeling tool and may be used from that point on as if it were a native modeling descriptor.

[0184] By using metamodels to define models, different implementations can be created using different software languages. For example, a customer model may be implemented as a C++, Visual Basic, or a Java class. The current state of the industry relies heavily on hand coding to implement a model from a metamodel. Some pioneering companies use computer aided software engineering (CASE) tools or Unified Modeling Language (UML) tools to generate source code directly from the metamodel. This source code may be a largely complete object-oriented implementation of the model described in the metamodel. The Object Management Group (OMG) has advanced the goals of CASE by introducing a common metadata description language for writing metamodels in XML syntax. This XML Metadata interface (XMI) language allows different modeling and CASE tools to be used together to generate source code. The OMG has also defined a Metadata Object Facility (MOF) that defines a mapping process for object-oriented programming models. This mapping process may be used to translate a metamodel into another metamodel, another view of the same metamodel, or source code.

[0185] The present invention provides a new concept not covered by CASE tools, XMI, or the MOF. The new concept is that of a direct access layer between the metamodel and its implementation using a meta-implementation layer. This layer allows general tools that understand metamodels to access and manipulate the implementation of that model. Tools can interact with one or more models to use and integrate these models.

[0186] Unlike the MOF, the implementation of a metamodel does not have to be object-oriented. Unlike CASE and Executable UML, the implementation of a metamodel is not required to be the result of generating source code and compiling the result. The implementation is not the result of a long and complex mapping process to generate this source code as required by MOF. Instead, a platform is selected which contains one implementation for each of the model descriptor types. There is a one-to-one relationship between each model descriptor and each metamodel accessor and the model descriptors and accessors are organized in exactly the same way as the metamodels are organized.