The MDA Foundation Model

1 Introduction

This document provides a formal model of the basic modeling concepts underlying the MDA approach to system development.

It covers the concepts *model*, *metamodel* and *transformation* and the relationships among them, together with some notions that are closely related to these concepts.

The OMG MDA Guide will formally define the Model Driven Architecture based on this Foundation Model. An MDA implementation consists of one or more applications of the pattern defined in the Foundation Model.

The MDA Foundation Model is a conceptual model that does not preclude any technology.

2 Models, Metamodels, and Systems

MDA is an approach to system development and interoperability that uses models to express and direct the course of understanding, requirements elicitation, design, construction, deployment, operation, maintenance, and modification.

At the core of the MDA are the concepts of *models* that represent *systems*, and of *metamodels* that define the languages in which the models are expressed. Figure 1 shows the relationships among these major concepts. All these concepts are defined in the following sub-sections.

2.1 System

A *system* is a collection of parts and relationships among these parts that may be organized to accomplish some purpose. [Note that this is a more general definition of *system* than used by other industry groups, e.g. INCOSE and IEEE.]

In MDA, the term *system* can refer to an information processing system but it is also applied more generally. Thus a *system* may include anything: a program in a computer, a system of programs, a single computer, a system of computers, a computer or system of computers embedded in some machine, a system of hardware, software, and people, an enterprise, a federation of enterprises, some combination of parts of different systems, a federation of systems, each under separate control, etc.

2.2 Environment

The interactions between a *system* and the *environment* or *environments* in which it is intended to work is an essential relationship in MDA.

The *environment* of a *system* includes everything external to the *system*. For example, the environment of a business system may include suppliers, customers, shareholders, regulatory bodies etc. The environment of a computer system may range from a group of business organizations to a piece of equipment.

The environment can be considered to be a system in its own right. Similarly, any system can be considered to be the environment for its subsystems, so the MDA principles can be applied recursively at any level.

2.3 Model

A model is a selective representation of some system whose form and content are chosen based on a specific set of concerns. The *model* is related to the system by an explicit or implicit mapping.

For example, it could be a UML model of some part of an application, or a business process model, or a simulation model.

The relationship between a model and the language in which it is expressed is captured in Figure 1 by the *conforms to* dependency between Model and Metamodel.

For example, in the case of a UML model of a set of classes of some part of an application, the language is the UML Metamodel for Class Diagrams.

A model may represent the software, hardware, environment, and other domain-specific aspects of the system. Such a model can include many kinds of expression; for example a model of a software system could include a UML class diagram, UML sequence diagrams, and images of the user interface, while a model of a physical system could include a representation of the hardware and physical environment, and a performance simulation. Some of these expressions are human readable, others exist in electronic media. For these expressions, there may exist several notations and formats. By a single *model* we mean the whole set of expressions constituting that model.

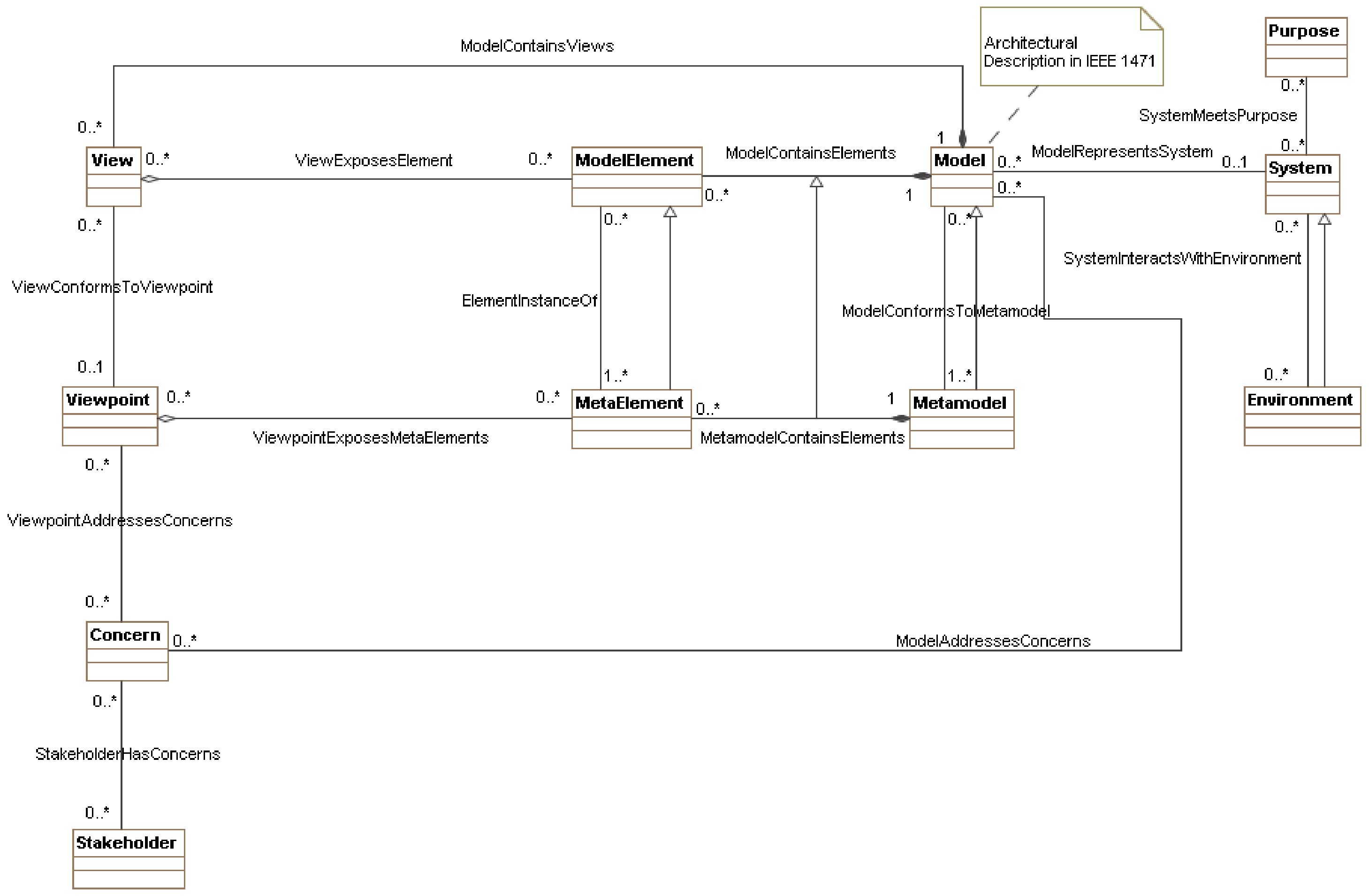

Figure 1 The relationships among models, metamodels, and systems

2.4 Metamodel

A *metamodel* specifies the abstract syntax of a *modeling language* and, as such, is a special kind of *model*. It can be understood as the specification of the set of all possible *models* expressed in that modeling language. Every model must conform to its metamodel(s).

2.5 Viewpoint and View

A *viewpoint* specifies a reusable set of criteria for construction, selection, and presentation of model elements, addressing particular stakeholder concerns.

A viewpoint may be partially expressed through the metamodel elements that it exposes, but may also include less structured expressions such as diagramming rules.

A *view* is a representation of a *system* that conforms to a *viewpoint*.

2.6 Artifact

An artifact is a piece of information that is used or produced by a system development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, binary executable files, tables in a database system, development deliverables, word-processing documents, and mail messages.

3 MDA Transformations

MDA Transformations accept input *elements* and produce output *elements*. Commonly used transformations include:

- model-to-model
- model-to-artifact (e.g. code, text, or document)
- artifact-to-model

The concept of *transformation* is broken down into some interrelated concepts: *transformation specification*, *transformation formal parameter* and *transformation record*. These concepts are defined in the following sub-sections.

3.1 Transformation Specification

A *transformation specification* defines how different *elements* will relate to each other. Typically, these elements belong to models or artifacts of the same system at different levels of refinement. When a change is applied to one of these elements, the transformation specification ensures that the change is applied to the related elements. A *directed transformation specification* is a transformation specification that determines how a set of output elements results from a set of input elements. A transformation specification is itself a model.

Figure 2 shows, at an abstract level, the internal structure of a model transformation specification. A transformation specification is made of a set of *transformation specification elements*, which determine how sets of elements relate to each other. Each such *TransformationSpecificationElement* determines how a group of output *elements* results from a group of input *elements*.

For example, in a rule-based implementation of *transformations*, the *transformation rules* constitute the *TransformationSpecificationElements.*

The *metamodels* specified by the *transformation formal parameters* determine the languages in which the input and output *models* are expressed.

The *transformation specification* is a generalized concept that covers both uni-directional and bi-directional transformations. The concept *directed transformation specification* is unidirectional, as shown in Figure 2. The roles *inParameter* and *outParameter* define subsets of the role *parameter*. A *directed transformation specification* defines the input and output side of the signature of the *transformation specification*.

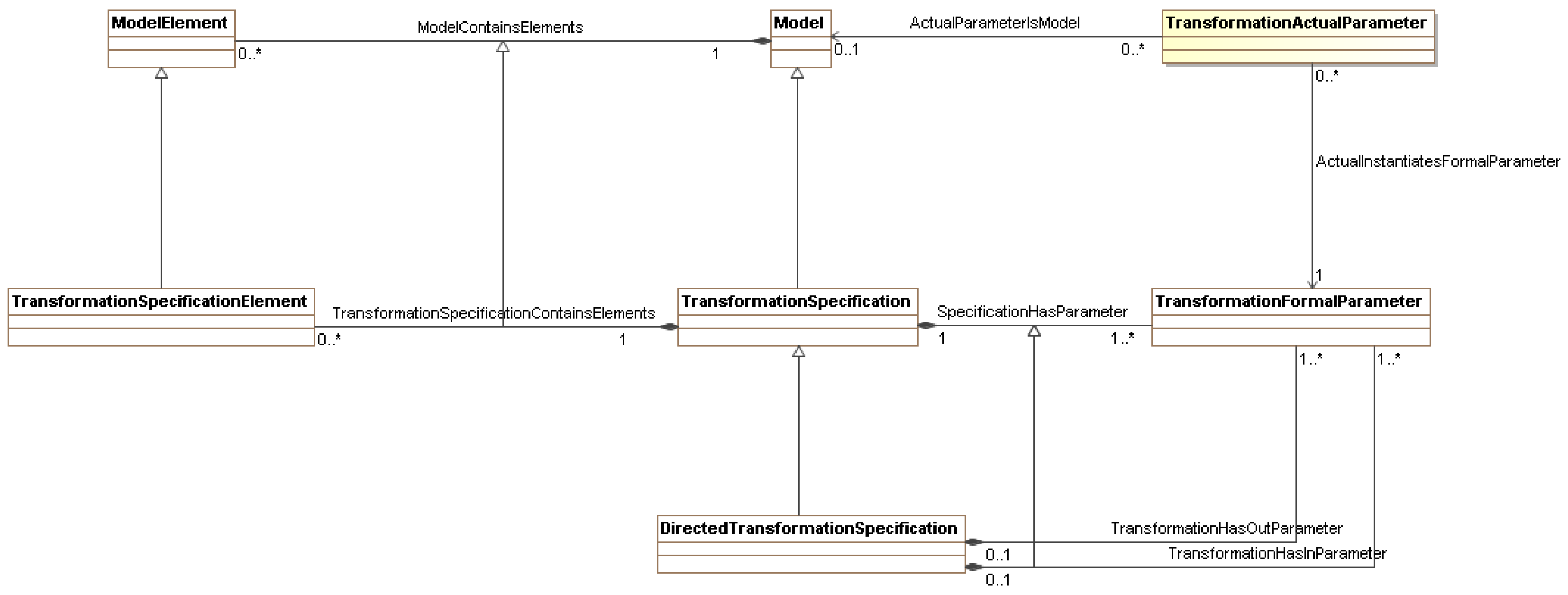
ModelElement
ModelContainsElements
0..*
1
Model
0..1
ActualParameterIsModel
0..*
TransformationActualParameter
0..*
ActualInstantiatesFormalParameter
1
TransformationSpecificationElement
TransformationSpecificationContainsElements
0..*
1
TransformationSpecification
SpecificationHasParameter
1
1..*
TransformationFormalParameter
1..*
1..*
DirectedTransformationSpecification
TransformationHasOutParameter
0..1
TransformationHasInParameter
0..1

Figure 2: Transformation Specifications and their relationships to models

3.2 Transformation Record

A transformation is executed based on a *transformation specification* and the *transformation specification elements* it contains. The transformation binds a set of input *models* and a set of output *models* to the model *transformation actual parameters* of the *transformation specification*.

A *transformationRecord*, as shown in Figure 3, keeps track of the transformation that occurred as a set of *Trace* elements each of which holds a record of the transformation of a group of model elements from the input models into a group of model elements in the output models, and the transformation specification element(s) that caused this. Note that the input and output elements are optional for a Trace because there may be transformation specification elements that match something in the input models but do not produce anything from it; or there may be model elements in the output models that are constant for the transformation and not dependent on anything in the input.

A *transformation record* has associations with the *transformation actual parameters* that were used as input and produced as output. A *transformation actual parameter* references a *transformation formal parameter* and a *model* that references, in turn, a *metamodel*, providing the modeling language specification for the *model*. The *models* used as the actual inputs and outputs of the transformation must comply with the *transformation formal parameters* of the *transformation specification*. Consequently, the modeling languages of these sets of *models* correspond to the modeling languages of the set of *transformation formal parameters* of the *transformation specification*.

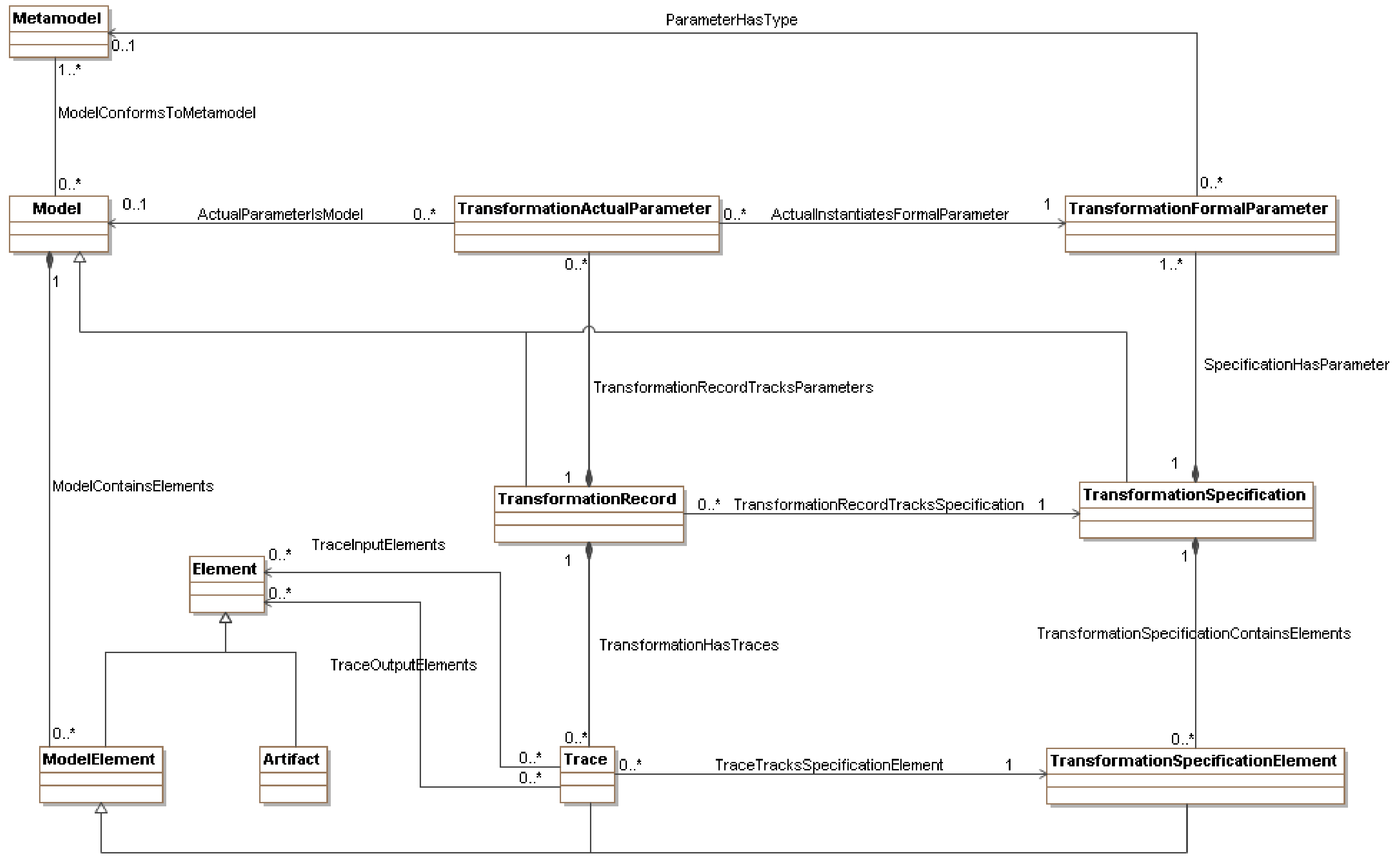
Metamodel
ParameterHasType
0..1
1..*
ModelConformsToMetamodel
0..*
Model
0..1
ActualParameterIsModel
0..*
TransformationActualParameter
0..*
ActualInstantiatesFormalParameter
1
TransformationFormalParameter
0..*
1
0..*
1..*
TransformationRecordTracksParameters
SpecificationHasParameter
ModelContainsElements
1
TransformationRecord
1
TransformationSpecification
0..*
TransformationRecordTracksSpecification
1
Element
0..*
TraceInputElements
0..*
1
1
TransformationHasTraces
TransformationSpecificationContainsElements
TraceOutputElements
0..*
ModelElement
Artifact
0..*
0..*
0..*
Trace
0..*
TraceTracksSpecificationElement
1
0..*
TransformationSpecificationElement

Figure 3: Transformation Records and their relationships to Transformation Specifications and model