is equivalent to the `Class` metaclass in MOF. `Property` from MOF is, however, split in two metaclasses in Ecore: `EReference` for links between `EClass`es and `EAttribute` for properties which are typed by primitive datatypes such as `Integer` or `String`. The `containment` attribute in `EReference` specifies composition links in Ecore. MOF uses the `isComposite` attribute for this. Other attributes have been factorized and placed into abstract metaclasses. For example, `upper` and `lower`, which defines multiplicities of links, are extracted into the abstract class `ETypedElement`.

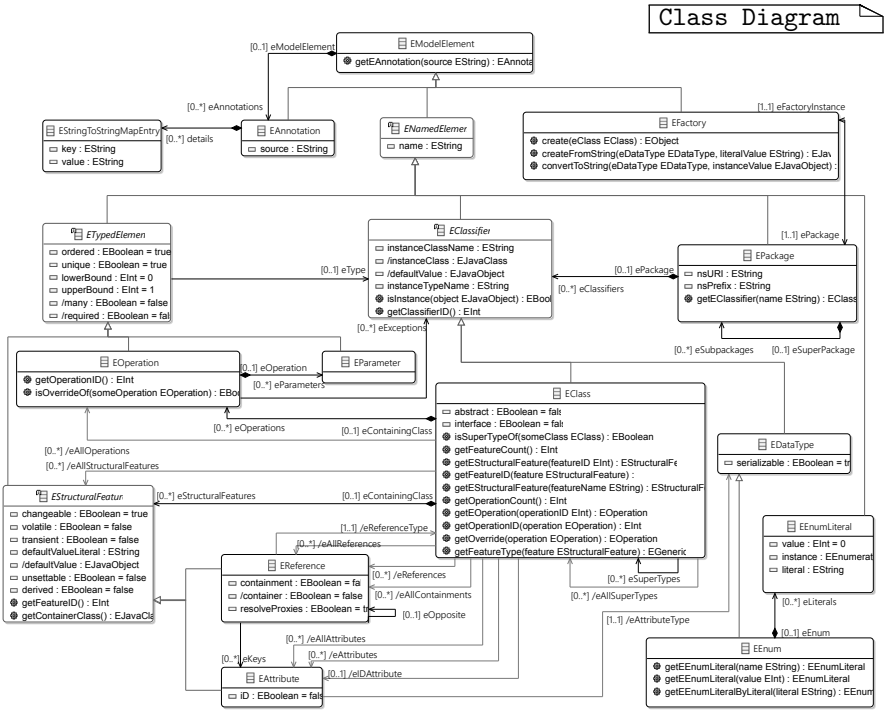

Figure 3.6 Ecore metamodel main concepts.

For avoiding confusion, not only with MOF, but also with the underlying Java implementation, the designers of Ecore have prefixed all the name of the concepts with an `E`. This is convenient when writing transformations and also helps to identify on which level (M1, etc.) a class resides.

EMF provides a number of tools that allow to specify metamodels and then to instantiate models conforming to them. The instantiation

vendors.

One important aspect of traditional practice that is perhaps threatened by automating quantity take-off in Estimator and similar software is the identification of errors by the quantity surveyor/estimator. The ability to browse the model through the geometric and textual panels provides an alternative method for identifying errors. Additionally, this supports filtering of outputs by element, by material, by type, and by story.

15.6.2.1 The Intelligent Building Model Language

Because of the imposing size of the IFC language, the DesignView platform (and, by implication, Automated Estimator) uses a simplified language for representing design models, called the Intelligent Building Model (IntBM). IFC models are converted into the IntBM language through an import wizard. An extract of the IntBM metamodel is shown in Figure 15.2. As can be seen, this language includes only about 30 classes, as opposed to more than 600 in IFC—a reduction aimed at simplifying the development of design analysis tools, and achieved through a few significant language design decisions.

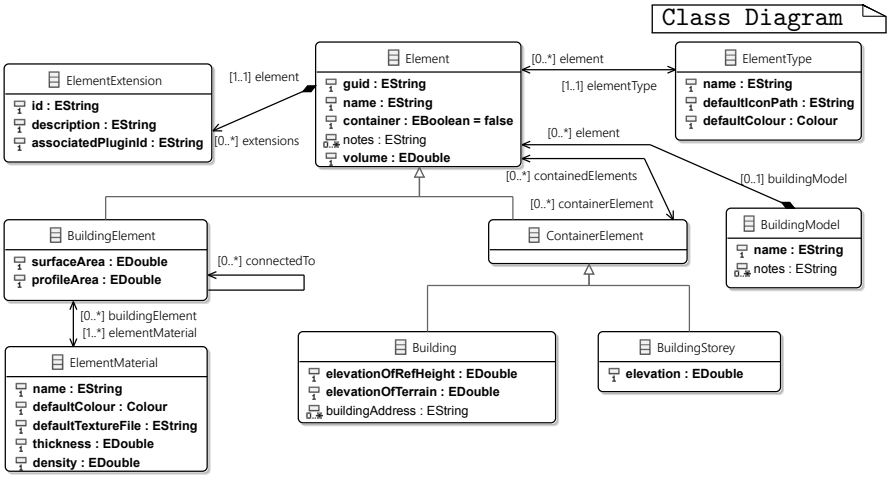

Figure 15.2 The intelligent building model metamodel.

The first economy is that a lot of elements are not considered during import. Discipline-specific information such as structural moment models, property sets containing metadata for lifecycle or performance

element analysis of the model, coloring of the 3D model is typically done by element-type in order to distinguish, e.g., walls from beams from columns from slabs, etc. The hierarchy and 3D viewers use two levels of selection sharing in order to facilitate inspection of the models. Clicking on an element or a set of elements in either view will highlight these elements in the other view. Dragging a container element or a set of elements from the hierarchy view to the 3D view will restrict the 3D visualization to just those elements that are dragged. This is particularly useful for inspecting a single floor.

15.6.2.2 The Bill of Quantities Language

The language used for describing bills of quantities is based on an analysis of the existing documents used by quantity surveyors. The result is the metamodel shown in Figure 15.3.

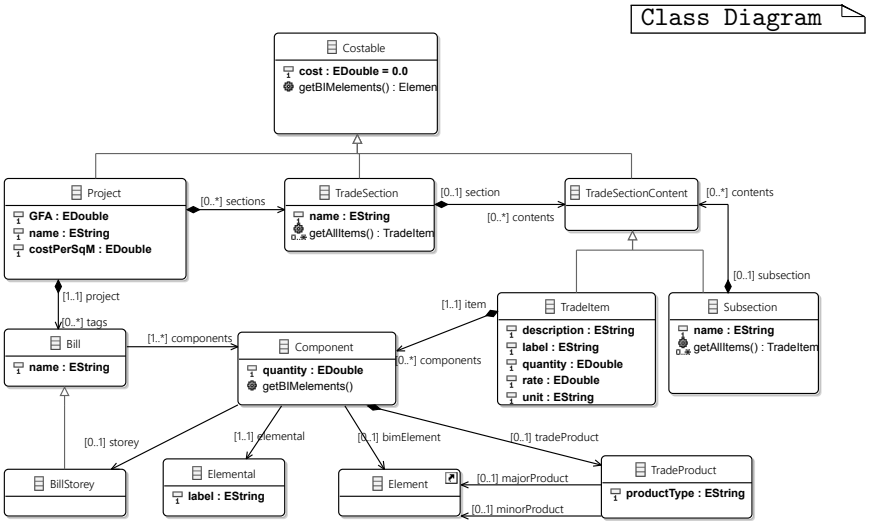

Figure 15.3 The Bill of Quantities metamodel.

The BoQ is broken down into a series of trade sections, each of which can in turn have a hierarchical structure within them. At the bottom of these hierarchies are `TradeItems`, each of which has a description, a quantity, a unit of measurement for the quantity, and a cost per unit. These trade sections and the trade items within them are not hard-coded in the metamodel; they are created by the take-off

implementation of its engine within Automated Estimator. We also discuss the mechanism for storing the trace information between the building model and the generated BoQ, and the facilities for inspecting/debugging the generated bill.

15.6.3.1 Take-Off Rules

Figure 15.4 shows the metamodel of the rule-based Take-Off Rules language used within Automated Estimator. There are two parts to this language—the structural part and the expression part. The high-level structures in the take-off language are based on those from the Bill of Quantities metamodel. The `RuleModule`, `Subsection` and `TakeoffRule` concepts correspond to, and result in the instantiation/population of the `TradeSection`, `Subsection`, and `TradeItem`/`Component` concepts, respectively, from the Bill of Quantities metamodel. Unlike bills of quantity, however, Take-Off Rule modules are stored with one `RuleModule` per file—the collection of rule modules is not modeled.

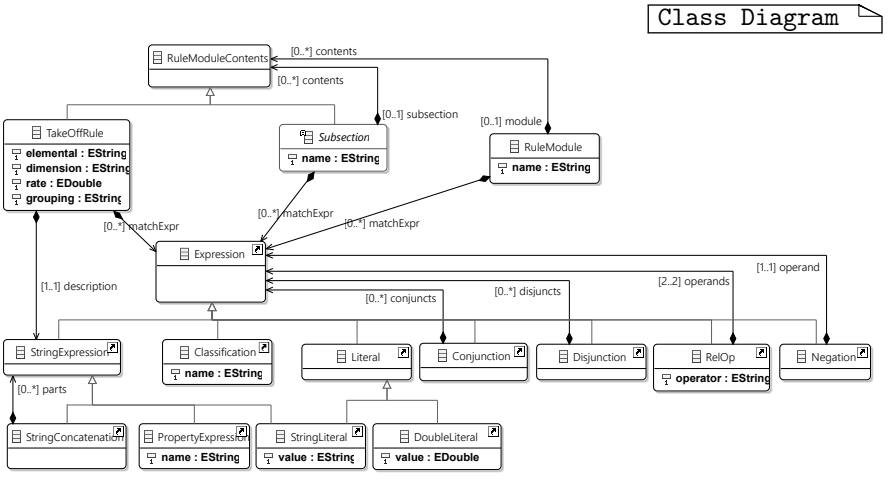

Figure 15.4 The Take-Off Rules metamodel.

The expression language used in the Take-Off Rules language is a simplified variant of the expression language from the Tefkat model transformation language [86]. It includes negation, conjunction, and disjunction, literals for strings and numbers, and binary relation operators for value comparison (=, <, >,