Approaches to Retargeting within Model Transformations
Basic Premise

• Model transformations map one or more source models to target model
• Typically, source model(s) are more abstract than target model, thus have to know little or nothing about target platform
• In order to produce valid and reasonable target models, transformation may require additional input that has knowledge of target platform
• In order to keep source model(s) clear of any target platform dependencies, this additional transformation input can be kept in annotations
Some Vocabulary

- **Metamodel**: Definition of structure, semantics, and constraints for class of models
- **Model**, then, is instance of metamodel
- Models that are instances of metamodels that describe properties of particular platform are **platform-specific**
- Models that are instances of metamodels that describe system at level of abstraction, one that’s sufficient to allow use of their entire contents for implementing system on different platforms, are **platform-independent**
Model, Metamodel, and Platform
Some More Vocabulary

- **Platform**: Specification of an execution environment for models
- **Platform realization**: Implementation of specification that platform represents
- **Primitive realization**: Platform realization that realizes only one platform
- **Composed realization**: Platform realization that realizes two or more platforms
- Set of all platforms and realizations forms **platform stack**
Platform and Platform Realization
And Yet More Vocabulary

- **Mapping**: Application of rules and algorithms to particular model in order to reach some related model
- Mapping should be applicable to large range of models that share some common structure, rather than specific to one model
- **Mapping technique**: Description of rules for transformation performed by mapping
- Rules are described at metamodel level in such a way that they’re applicable to all sets of source models
Mapping and Mapping Technique
The Final Vocabulary Terms

- **Annotation**: Sticky note attached to source model that contains information that supplements model and directs model transformer

- Mapping may use several annotations on source models; conversely, annotation may cater to several different mappings

- Annotations are sensible if modeler wants to anticipate design decisions or reuse design decisions across transformation techniques

- **Annotation model**: Description of structure and semantics of annotations....Mapping technique specifies annotation models of which it requires instances (annotations)
PIMs, PSMs, and Annotations
Annotation Models as Adapters

- Annotation models are adapters between metamodels: Model transformation technique may require information not defined by its source metamodels; it may produce information that can’t be accommodated by its target metamodels
- Example: Source model doesn’t know about distributed objects and remote accessibility, while target technology offers locally accessible objects as well as remotely accessible objects
- Annotations allow modeler to avoid polluting source model with extraneous information
More About Annotation Models

• Annotation model should describe information required by transformation technique to find unique transformation rule based upon source model element and its transformation context

• Annotation model consists of pairs of source metamodel elements and possible design decisions; it relates to a transformation technique

• We don’t know about constraints placed upon transformation rules in terms of source model structure and transformation context; therefore, as first guess, we require that annotation model enumerate all transformation rules that match one or more elements of the source metamodel, so modeler can choose from them
Avoiding “Lossy” Model Transformations

- **Required annotation** is attached to source model; it provides additional input to one or more mappings
- **Provided annotation** is attached to target model; it represents output of mapping
- Can also add annotations to both source and target that contain surplus information that’s not suitable for inclusion in given model
Annotation Model Structure

- Since mappings are executions or applications of mapping techniques, mapping techniques actually dictate structure of annotations.
- Result is that general structure of annotations is same for all applications of same mapping technique.
- Elements that are likely to be used together by existing and future mapping techniques should be grouped together in one annotation model.
Annotation Grouping Example

- UML-like metamodel serves as platform “component paradigm,” while another metamodel, oriented toward source code, is present for particular EJB 1.1–compliant container implementation
- In between these two platform definitions, EJB 1.1 specification serves as intermediate platform
- Ideal annotation model partitioning uses two partitions: One for EJB–specific annotations, and another for product-specific annotations; this enables the reuse of EJB annotations in event that development team chooses another EJB implementation
Example: Transforming Analysis Model to Design Model

- Analysis entities: Customer, Address, Accounts

- Customer and Account should be represented as business entities in design model, which implies that their IT equivalents are stateful, persistent, identifiable, and remotely accessible

- Address might be dependent object, since it doesn’t have to be identifiable as a first-level object and it will be remotely accessed by value rather than by reference

- Customer would be backed up by one annotation stating that it’s stateful, another stating that it’s persistent, and so forth
Example: Transforming Design Model into Relational Schema

• Business entity Customer would correspond with Customer table, with columns for SSN, Name, Date Of Birth, etc.

• Dependent object Address would correspond with number of columns for City, Zip Code, and Street in that table

• Business entity Account would correspond with Account table, with columns for Account Number, Balance, etc.

• Account Ownership table would have columns for SSNs of Customers owning given Accounts and Account Numbers of Accounts owned by some Customers
**Database Performance Tuning Example**

- No straightforward way to connect Balance attribute of Account entity with Balance column of Account table
- Typical SQL query:

```sql
SELECT sum (Balance) 
FROM Account 
WHERE Account_Number IN ( 
    SELECT Account_Number FROM 
    AccountOwnership WHERE 
    Customer_SSN = 4711 )
```
Database Performance Tuning
Example (cont.)

• There are fewer updates of Balance column in Account table than queries on that column; queries eat up all power of database server

• DBA would like to suggest introduction of additional column that effectively caches calculation of account balance, with column held consistent by update triggers on Balance column

• This is easy decision if annotations exist on relational database: DBA thinks additional column should go in Customer table; DBA proposes additional query on that column to software designer; designer easily models this with annotation
Backwards Traceability

• Appropriate annotation model should require database tables and columns to be backed up by annotations that refer to respective design model elements

• In example, Customer table would be backed up by annotation referring to Customer business entity, Name column by annotation referring to Name attribute, etc.

• Target metamodel of transformation technique doesn’t have to be able to accommodate any information contained in its respective source metamodels in situations where following model transformations use this surplus information.
Example: Analysis Model to Design Model to Schema

- Suppose analysts have provided instance quantities in analysis model: System to be developed will have to handle, say, 100,000 Customers and 500,000 Accounts
- But instance quantities aren’t necessary to perform transformations from design model into programming language source code and SQL data definition statements
- Solution: Back up business entities with annotations that hold specified instance quantities specified in analysis phase, then transform Quantity attribute of analysis notion into Quantity annotation of a design business entity for later transformation into optimized SQL DDL statements
Modeling Annotations with UML

- Must be some generic mechanism to attach information to model elements that’s beyond scope of the respective metamodel
- In UML 1.4, tagged values have become similar to attributes on class, in that they’re typed and have multiplicities—so it’s possible to define tagged value whose value is of primitive data type or enumeration type, or reference to another model element
Modeling Annotations with UML: Example

• Customer notion within analysis model would have tagged value whose tag is be designElement and whose value is BusinessEntity

• UML 1.4 would allow definition of this tagged value as having instance of enumeration design element Enumeration as its value, where literals of this Enumeration would be named BusinessEntity and DependentObject
Required Tagged Values

- Required tagged values used in defining UML profile for a metamodel; they provide means to properly reflect structural features of extended model elements
- Idea is that if model element has attached stereotype, this stereotype requires presence of number of tagged values
- Example then introduces DesignAwareNotion stereotype, which applies to Customer notion, that requires designElement/BusinessEntity tagged value on Customer
Annotation Model as UML Profile

- Multiple profiles can be applied to one package of model elements; model element can have multiple stereotypes
- So, it’s possible to define annotation model as UML profile and to use stereotypes and required tagged values to implement annotations
- Thus, UML 1.4 infrastructure facilitates consistency checks on annotations
Using Annotations to Resolve Rule Ambiguity

• Sketch transformation technique to formalize set of transformation rules, with each rule containing (1) precondition on source model element that defines when technique will apply to element, (2) postcondition on target model element that defines what result of its application shall be and allows one to deduce way to get there, and (3) set of component transformation rules to apply to components of the source model element, where instructions to retrieve those components (for example, aggregated source model elements) is implicitly provided by source metamodel.
Using Annotations to Resolve Rule Ambiguity (cont.)

• Start technique with transformation rule stating that source models complying with source metamodels shall be transformed into target models complying with target metamodels

• Figure out those component transformation rules that either overlap with regard to their preconditions or that require precondition information that isn’t specified in source metamodels

• If rules overlap with respect to preconditions, find discriminator, then add enumerating element to the annotation model, and relate annotation model element to element of source metamodel for which transformation rule is sensitive
Using Annotations to Resolve Rule Ambiguity (cont.)

- If rule requires additional information, add primitive element to the annotation model and relate it to respective source metamodel element
- Descend into composition hierarchy of source model element and recurse
- Do something similar when looking for annotations that support potential of optimization: (1) Look for rules that don’t refer to all target model features and relations; (2) Decide which omissions offer optimization potential; (3) For each one, introduce element to target annotation model and relate it to corresponding element of source metamodel