A DATA TRANSLATION TOOL FOR ENGINEERING SYSTEMS

David L. Spooner
Donald Sanderson
Georgios Charalambous

Computer Science Department and Rensselaer Design Research Center Rensselaer Polytechnic Institute Troy, New York 12180

Abstract

Exchange of data between engineering systems is a common problem in engineering and manufacturing computing environments because of heterogeneous hardware and software. Formal data exchange standards exist in only limited cases. Early work focused primarily on "flat" file structures, and is not easily extended to the complex object data structures required to model engineering data. This paper describes a Data Translation Tool developed explicitly for engineering data. It is based on the ROSE 1.0 object-oriented engineering database system. The ROSE data model is flexible enough to handle a wide variety of engineering data structures, and the ROSE extended relational algebra provides the primitive operations needed to edit these data structures to translate engineering data from one form to another. The motivation, design, and implementation of this Data Translation Tool are described, along with lessons learned.

Introduction

Complex engineering and manufacturing computing environments require that multiple heterogeneous systems be able to share data efficiently in unpredictable ways. This requires aspects of both distributed data management and data exchange. Distributed data management is required to move data from one node to another in a network of engineering workstations and computers, and to maintain the consistency and integrity of the data. Data exchange is needed because the network of hardware and software systems in such environments is highly heterogeneous so that data produced by one system may not be in the correct form to be used by other systems. A coordinated solution to these two problems is needed to build an effective engineering and manufacturing environment.

This paper addresses the second of these two problems - data exchange. The goal is to provide an easy to use tool for implementing data exchange translators between pairs of systems in an engineering and manufacturing computing environment for situations where no data exchange standards exist. The tool supports a graphical "point and click" style of user interface in which a user describes interactively on a graphics workstation the edits that must be made to the data from one application system to put it into a form suitable for use in another application. The tool is also responsible for performing these translations on the actual data to produce the data structured as required for the target application system.

Data exchange standards such as IGES [Lie83], PDES [H088], and others, have been developed to address this problem for specific types of data -- for example, geometry and product definition data. One of the major motivations to the development of these standards is the desire to provide a common exchange format. Thus, any two application systems dealing with these types of data can exchange the data as long as both applications support a translator to and from the exchange format. This limits the number of translators that must be written to two the times the number of applications systems -- one to convert to the exchange format, and one to convert from the exchange format, for each application system.

Data exchange standards work well for specific problem domains. However, for most situations no data exchange standards exist. The Data Translation Tool (DTT) is intended for these situations. Thus, DTT must have the flexibility to model data from a wide variety of engineering application databases, not just object-oriented ones, and must provide a set of primitive edit operations to translate data in these databases from one structure to another.

To accomplish this, the decision was made to base DTT on the ROSE 1.0 object-oriented engineering database system [Hat88]. This was done for several reasons. First, the ROSE data model organizes data into objects using several types of abstractions commonly found in semantic data models and useful for modeling a wide variety of types of data (i.e., flat file structures and relational, hierarchical, and network database structures, as well as more complex structurally object-oriented data models for engineering data [Kem87]). In addition, the ROSE system provides an extended relational algebra to efficiently manipulate and edit the objects stored in a ROSE database. This algebra provides the basic primitive operations needed for data translation. Much of the memory management functionality required during the translation process can be handled by ROSE. And finally, ROSE was readily available for the DTT project.

Early work in data translation centered on the problems of translating record-oriented structures to other record-oriented structures; that is, from a flat file to a relational DBMS, or from one relational DBMS to another [Ber80, Fry77 Lum76, Sho75, Shu77, Tho79]. For engineering and manufacturing systems, this is not sufficient. Engineering data is often modeled using complex hierarchical data structures with multi-valued...
components. It is generally agreed that simple "flat" data models are not sufficient to capture the semantics of engineering data, and that more complex object-oriented data models are needed [Sid80, Har87]. Thus, the primary contribution of this work is the extension of data translation concepts to deal with complex engineering objects.

The next section provides background material on the ROSE data model and its extended relational algebra. The following section discusses the design and implementation of the ROSE Data Translator which forms the core of DTT. The extensions and modifications needed to expand this system into DTT are then described.

**Overview of the ROSE Data Model**

Since DTT is built using the ROSE object-oriented engineering database system, it is necessary to describe the data model supported by this system. This will be done via an example. Consider a data structure that might be built in a programming language such as Pascal, or C. Such a data structure using a hierarchy of records to describe a text or graphics window for an engineering workstation is illustrated in Figure 1. In this data structure, a record of type Window is used for the basic window data. It contains fields to describe the x and y coordinates for the location of the window as well as the length and width of the window. This record also contains two other fields, both pointers to records. The first field, labeled border, points to a record with data that describes the border to be placed around the window. The second field, labeled contents, contains a pointer to a Text record describing the data that is to be displayed in the window. Since windows can alternatively have graphics in them, the pointer in the contents field of a Window record might instead point to a Graphics record containing graphics data to be displayed in the window (as illustrated in the figure). Thus, the type of the contents field of a Window record is a union data type allowing as its value a pointer to either a record of type Text or of type Graphics.

**AND/OR Trees for Modeling Hierarchical Structures**

Hierarchical data structures such as the one in Figure 1 can be modeled using AND/OR trees in which AND nodes represent records and OR nodes represent union data types. For example, the data structure in Figure 1 can be modeled by the AND/OR trees in Figure 2. Note that in the figure an AND node has an arc immediately below it to distinguish it from an OR node.

---

**Figure 1: Data Structure to Describe a Window**

```
Window
  x  y  length  width  border  contents
  1.0 3.0 6.0 4.0

Border
  size  label  pattern
  0.25  My Window  solid

Text
  font  psize  text
  Times 12  this is the text...

(Graphics)
  line type  scale  rotation  geometry
  solid  1.0  0.0

Figure 1: Data Structure to Describe a Window
```
The root node of each tree defined by an AND node declares a record type. This record type is also called a domain. The fields in the record type are named by the leaf nodes in the tree. These names are called attribute names. The type of each field is listed under its leaf node in capital letters. This type is also called a domain. Thus, in the figure, attribute names are in lower case while domain names are in upper case. The easiest way to view a tree defined by an OR node is that it declares a record with one field. The type for that field is the union of the attributes named by the leaf nodes in the tree. Thus, the tree with root labeled CONTENTS in Figure 2 can be viewed as declaring a record type with one field, the attribute of which may be either text (in which case the field contains a pointer to a TEXT record), or graphics (in which case the field contains a pointer to a GRAPHICS record). Note that this differs slightly from the data structure of records in Figure 1. In Figure 1, a Window record points directly to a Text or Graphics record. Using AND/OR trees, an extra level of indirection is introduced so that a WINDOW record points to a single-field record of type CONTENTS. This record, in turn, points to a TEXT or GRAPHICS record. This extra level of indirection is necessary to provide a generic tool (such as AND/OR trees) for declaring hierarchical data structures.

Suppose now that the data structure for windows is made more complicated. In particular, consider the Text record type. A Text record allows a window to contain one string of text formatted using one font and one point size. Suppose it is necessary to allow a window to contain many text strings, each with its own font and point size. In this case, each string can be modeled by a Text record, but a list of Text records is necessary to describe the complete contents of a window. Also, in this case, a Window record must be able to point to a list of Text records, not just one.

This situation can be modeled in an AND/OR tree with a new kind of AND node, called a Multiple AND node. A Multiple AND node defines a list of records, where each record has the fields specified by the children of the AND node in the AND/OR tree. An attribute defined over this domain can be viewed as a pointer to the list of records. This list may be empty, and is of unbounded size. Thus, the definition of domain TEXT in the AND/OR trees of Figure 2 should be changed to a Multiple AND node to correctly model the new data structure allowing lists of Text records. The new AND/OR tree for the TEXT domain is shown in Figure 3a. Note that an asterisk is placed under the domain name to indicate that it is a Multiple node.

Now suppose it is desirable to allow text and graphics to be intermixed in the same window, with multiple text and graphics items allowed in the window. That is, the contents field of a Window record should point to a list of records, some of which are Text records, and some of which are Graphics records. (Actually, if we assume that the modification of the preceding paragraph has been made, then the contents field points to a list of items, some of which are Graphics records and some of which are sublists of Text records.)

This situation can be modeled in an AND/OR tree with a new kind of OR node, called a Multiple OR node. A Multiple OR node defines a list of records, each with a single field. The type of that field is the union of the attributes specified by the children of the Multiple OR node in the AND/OR tree. An attribute defined over this domain can be viewed as a pointer to the list of records. As in the case of Multiple AND nodes, the list of records for a Multiple OR may be empty, and is of unbounded size. Thus, the definition of the CONTENTS domain in Figure 2 should be changed to a Multiple OR node to correctly model the new data structure. Again, note the asterisk.

This leads to four types of nodes in an AND/OR tree: AND nodes without an asterisk (Single AND), AND nodes with an asterisk (Multiple AND), OR nodes without an asterisk (Single OR), and OR nodes with an asterisk (Multiple OR). Using database design terminology, an AND node is sometimes called a generalization abstraction, an OR node is sometimes called an association abstraction. Lastly, to complete the definition of AND/OR trees, it is necessary to have a set of primitive domains. Figure 2 assumes that the primitive domains integer, float, and string are available.

The data model just described is the data model of the ROSE system. ROSE 1.0, the version of ROSE used for DIT, is a structurally object-oriented database system supporting the creation and manipulation of complex engineering objects. For more details on the ROSE 1.0 system, see [Har88].

Object Sets

A complete AND/OR tree such as the one in Figure 2, describes the data structure for a complex engineering object. For example, the AND/OR tree in Figure 2 describes the structure of a Window object. In ROSE, such an object is called an Application Object. The individual AND/OR trees that collectively define the complete AND/OR tree for an Application Object describe subobjects of this Application Object. For example, the AND/OR tree with root node labeled BORDER in Figure 2 describes the data structure for a Border subobject of a Window Application Object.
ROSE groups Application Objects defined over the same domain into Object Sets. Subobjects of an Application Object are stored as part of the Application Object in its object set. ROSE uses these object sets for management of secondary storage and main memory. An entire object set can be read and written as a unit. Each Application Object in an object set has a unique name within the set. This name is used to identify the object and serves much the same purpose as an object identifier in object-oriented programming languages, and as a key in relational database systems.

**ROSE Extended Relational Algebra**

To manipulate objects defined using this data model, ROSE has an extended relational algebra [Har87b]. This algebra includes operators for building records (tuples) and hierarchies of records, and operations for breaking a hierarchy of records (tuples) into its constituent parts. It is similar in many ways to the relational algebras of nested relational systems such as NF² [Pis86]. In addition, the algebra contains several operations for efficiently searching the hierarchical structure of an object. One of the most useful of these is an extended projection [Har87b] operation that projects from an object all occurrences of an attribute regardless of how deeply nested they are inside the object. This algebra provides the primitive functionality for implementing the translation operators supported by D'T.

**The ROSE Data Translator**

The major component of D'T is the ROSE Data Translator which is a tool for editing the structure of a ROSE database. This system is more than a schema editor. It edits both the schema for a ROSE database and the database itself so that it remains consistent with its new schema. The ROSE Data Translator has a graphical user interface. The user selects an object set for translation, and a picture of the AND/OR trees describing objects in that object set is displayed on the workstation. Using a sequence of translation operations selected from menus, the user then makes modifications to these AND/OR trees, indicating the changes to be made to the structure of the objects in the object set. Once done, the sequence of operations specified by the user is applied to the objects in the object set to create the new (reorganized) database.

**An Example**

As an example of translating a ROSE database, consider a simple graphics application database. The initial definition for this database is shown in Figure 4a. A sketch is composed of an identification number, a line type, and a set of points, some of which are x-coordinate, y-coordinate pairs, and the rest of which are polar coordinates. Suppose that it becomes necessary to break them into three objects sets that contain the data needed for these point lists. The original points attribute in Sketch objects will be replaced with a reference to the appropriate point list object in the new object set.

1. **Reference Insertion applied to the points attribute** -- this operation will extract the list of points from each Sketch object, and construct a new object set containing these point lists. The original points attribute in Sketch objects will be replaced with a reference to the appropriate point list object in the new object set.

2. **Extended Projection on the point1 attribute** -- this operation will extract all subobjects with the point1 attribute from the objects in the new object set created in step 1. These subobjects are used to create the new object set named Cartesian in Figure 4b.

3. **Extended Projection on the point2 attribute** -- this operation will extract all subobjects with the point2 attribute from the objects in the new object set created in step 1. These subobjects are used to create the new object set named Polar in Figure 4b.

4. **Delete object set applied to the object set created in step 1** -- this operation deletes the object set created in step 1 since all useful information in it has been extracted to produce the object sets Cartesian and Polar.

The new Point Application Objects created in step 1 will have the same names as the Sketch Application Objects.
from which they came. The Cartesian and Polar Application Objects created in steps 2 and 3 will again use these same names, but append serial numbers to make them unique. Thus, the reference inserted into Sketch objects in step 1 can be used to reference the individual points in the Cartesian and Polar object sets. This is the reason that step 1 is needed. Without it, there would be no way for the final Sketch objects to reference points in the other two object sets.

Architecture of the ROSE Data Translator

The ROSE Data Translator consists of three major components as illustrated in Figure 5. The first component is the Graphical User Interface mentioned above. This interface displays the AND/OR tree definition for an object set, and allows the user to edit these trees by pointing to nodes and applying translation operations to them selected from a menu. The output of this component is a "program" of translation operations that can be run off-line to modify the database to conform to the new definition.

The second component is the Front-End Processor. This component operates as an interpreter for the "programs" produced by the Graphical User Interface. It verifies correctness of individual operations; manages main memory, secondary storage, and the movement of objects and object sets between the two; and invokes the appropriate routines to process the translation operations.

The final component in the ROSE Data Translator is the Library of edit operations and utilities. This component includes the routines that implement each of the translation operations, and all utilities needed by these routines. The next section contains brief descriptions of the translation operations currently supported by the ROSE Data Translator.

ROSE Data Translator Operations

As discussed above, the ROSE data model supports three basic constructs -- AND nodes (single and multiple), OR nodes (single and multiple), and object sets. Each of these constructs enforces a particular data organization. Thus, specific translation operations are required for each of these constructs. The operations were designed with a view towards preserving the semantics of engineering objects as much as possible. Thus, the operations provided are those that seem rational for translation from one consistent object structure to another. Operations such as changing an AND node to an OR node and vice versa where not implemented. The set of operations was also designed to be minimal in number, yet able to perform any desired translations.

The operations for AND nodes (aggregation abstractions) include:

- **Aggregation**: Construct a new object set containing objects that are the aggregations of the objects in the argument list of object sets.
- **Aggregation Merge**: Application Objects in an object set whose domain is defined by an AND node are merged into an AND node in another object set.
- **Aggregation Split**: Remove one or more subobjects from an AND node. The subobjects are used to form a new object set.
- **Association Deletion**: Convert a multiple AND node to a single AND node by duplicating the rest of the object once for each subobject in the multiple AND node. (Also works for OR nodes.)
- **Association Insertion**: Convert a single AND node to a multiple AND node. (Also works for OR nodes.)

The operations for OR nodes (generalization abstractions) include:

- **Generalization**: Construct a new object set using a generalization abstraction to contain the objects in the argument list of object sets.
- **Generalization Merge**: Application Objects in an object set whose domain is defined by an OR node are merged into an OR node in another object set.
- **Generalization Split**: Remove one or more subobjects from an OR node. The subobjects are used to form a new object set.
- **Association Deletion**: Convert a multiple OR node to a single OR node by duplicating the rest of the object once for each subobject in the multiple OR node. (Also works for AND nodes.)
- **Association Insertion**: Convert a single OR node to a multiple OR node. (Also works for AND nodes.)

Operations that apply to object sets include:

- **Delete Object Set**: Delete an object set and all the objects in it.
- **Object Set Function Application**: A user supplied function is applied to every object in an object set to produce a new set of objects in a new object set.
- **Apply Function**: A user supplied function is applied to an object set as a whole to produce a new object set as its result.
Object Set Merge: Several object sets with objects from the same domain are merged.

Object Set Unmerge: The objects in an object set are grouped and partitioned into several object sets. Objects are grouped either by similar name, or by common value of a specified subobject.

Extended Projection: An extended projection operation is performed on all the objects in an object set, and the result used to create a new object set.

Additional translation operations include:

Attribute Change: Change the name of an attribute for the objects in an object set.

Domain Change: Convert a subobject from one domain to another.

Rename: All objects in an object set are renamed.

Reference Deletion: A subobject that references an object in another object set is replaced by the referenced object.

Reference Insertion: A subobject of an Application Object is extracted from the Application Object and used to create a new object in a new object set. The original subobject is replaced by a reference to the new object.

The functions required by some of the operations are functions written in the ROSE programming language. These functions take an object (or set of objects) as an argument, and produce a new object (or set of objects) as a result.

Architecture of the Data Translation Tool

The Data Translation Tool (DTT) exploits the ROSE Data Translator to do the major work of specifying and executing desired data translations. DTT can be viewed as a translator generator. It allows a user to define via a graphical interface a translator "program" to translate data from one application for use in another. This "program" can then be run any time data must be exchanged between these two applications. The architecture for DTT is shown in Figure 6.

Data from a source application database is imported into a ROSE database where it can be translated with the ROSE Data Translator. The resulting database is then exported to the target application database. When data is imported from the source application database, it is placed in a ROSE database with a structure that matches the structure of the data in the application database as closely as possible. Similarly, the final structure of the edited ROSE database from which data is exported should match the structure of data in the target application database as closely as possible.

This simplifies the processes of importing and exporting the data. Because of the flexibility of the ROSE data model, it is possible to model the structure of any application database that can be viewed as a hierarchy of records or tuples (i.e., most commercial DBMSs, file systems, and structurally object-oriented database systems for complex engineering objects).

Importing data into ROSE from a source application, and exporting it from ROSE to the target application, is done with four sets of procedures -- application-specific and application-independent import/export procedures. Data is extracted from the source application database by the application-specific import procedures. This data is used to build a ROSE database by the application-independent import procedures. The application-independent export procedures initiate the process of exporting a ROSE database object to a target application database by partitioning the ROSE object into appropriate pieces for the target application database. The application-specific export procedures take these pieces and store them into the target application database. The application-specific import and export procedures must be written once for each application database that will use DTT. The application-independent import and export procedures are generic and are used in all translation processes. These procedures are guided by the AND/OR tree definitions of objects in ROSE. The concept of application-specific and application-independent translation procedures has been explored previously in the Common Data Interchange Language (CDIL) tool for data exchange developed in the Rensselaer Center for Interactive Computer Graphics [Lea84]. Table I contains a preliminary set of application-independent import tools, and Table II contains a preliminary list of application-independent export tools. These tools are invoked through a graphical user interface.

Using the Data Translation Tool

To use the Data Translation Tool, the application-independent import tools are used through a graphical interface.
to construct an AND/OR tree definition of a ROSE database that corresponds as closely as possible to the source application database. An empty object set is then constructed to contain objects defined by this AND/OR tree.

Once the AND/OR tree and object set have been constructed, the "get" import tools are used to fetch data from the source application database to fill the object set. In some cases, it may be more efficient to build the objects in this object set one by one (in which case get object is used). In other cases, it will be more efficient to construct all the objects in parallel (in which case get object set is used). This decision is dependent on the structure of the source application database and the application-specific procedures that access it.

When either the get object or get object set operation is selected, the AND/OR tree definition for the objects is used to create a sequence of calls to the application-specific import procedures to extract data from the source application database to build an object in the ROSE database. Objects are built from the leaves of the AND/OR tree up using a postorder traversal of the tree. As a result, all subobjects of an object are constructed before the object itself so that they are available when needed to build the parent object.

Unfortunately, it is not this simple. OR nodes in an AND/OR tree represent places in an object where all but one branch below this node are pruned from the tree. Thus, when an OR node is encountered during the postorder traversal of the tree, it is necessary to invoke an application-specific procedure to determine which branch should be followed for the current object. All other branches are ignored for this object.

During the postorder traversal of the AND/OR tree, application-specific import procedures are called for any AND node in which:

1) One or more children of the AND node are attributes with primitive domains whose data value must be extracted from the application database.

2) The AND node is a multiple abstraction (association) and it is necessary to determine if there are additional occurrences to be included for this AND node.

In the first case, the application-specific procedure returns values for all the primitive domains in this AND node, allowing the representation for the AND node to be constructed in the ROSE database. In the second case, if the operation indicates that additional occurrences exist for the AND node, the postorder traversal of the AND node and its children is repeated.

Table II: Application-Independent Export Routines

<table>
<thead>
<tr>
<th>Fetch Object Set</th>
<th>Fetch object set from secondary storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put Object</td>
<td>Export an object to the target application</td>
</tr>
<tr>
<td>Put Object Set</td>
<td>Export an object set to the target application</td>
</tr>
<tr>
<td>Delete Object Set</td>
<td>Delete an object set that is no longer needed</td>
</tr>
</tbody>
</table>

2) The AND node is a multiple abstraction (association) and it is necessary to determine if there are additional occurrences to be included for this AND node.

In the first case, the application-specific procedure returns values for all the primitive domains in this AND node, allowing the representation for the AND node to be constructed in the ROSE database. In the second case, if the operation indicates that additional occurrences exist for the AND node, the postorder traversal of the AND node and its children is repeated.

During the postorder traversal of the AND/OR tree, application-specific import procedures are called for any OR node in which:

1) The relevant child for this instance of the OR node is an attribute with a primitive domain whose value must be extracted from the application database.

2) The OR node is a multiple abstraction (association) and it is necessary to determine if there are any more occurrences to be included for this OR node.

These two cases are similar to the cases above for AND nodes. As mentioned above, it is also necessary to call an application-specific procedure when first encountering an OR node to determine which branch to take below the OR node.

From this discussion, it is evident that application-specific import procedures must exist for all AND nodes and OR nodes with children that have attributes with primitive domains, and for all nodes which represent associations. Also, an application-specific procedure is needed for each OR node in the AND/OR tree to determine which branch beneath the node to follow for each specific object.

Once the application data has been converted into a ROSE database using the import procedures, the ROSE Data Translator is used to translate the ROSE database into the form required for the target application database. The export procedures are then used to enter the data into the target application database. This process is the reverse of importing the data.

Table I: Application-Independent Import Routines

<table>
<thead>
<tr>
<th>Attribute Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>Define a new attribute</td>
</tr>
<tr>
<td>Generalization</td>
<td>Define a new AND node</td>
</tr>
<tr>
<td>Association Insertion</td>
<td>Convert a node to a multiple node (association)</td>
</tr>
<tr>
<td>Object Set Definition</td>
<td>Define an object set for an AND/OR tree</td>
</tr>
<tr>
<td>Name Object</td>
<td>Generate the name of a particular object</td>
</tr>
<tr>
<td>Get Object</td>
<td>Get an object (as defined by the current AND/OR tree) from the source application</td>
</tr>
<tr>
<td>Get Object Set</td>
<td>Get all the objects (as defined by the current AND/OR tree) from the source application</td>
</tr>
<tr>
<td>Store Object Set</td>
<td>Store the current object set as a ROSE database in secondary storage</td>
</tr>
<tr>
<td>Start Object</td>
<td>Start constructing a new object</td>
</tr>
</tbody>
</table>

The first version of the ROSE Data Translator has been implemented and tested. It is implemented using the programming language supported by the ROSE 1.0 database system, and consists of about 6000 lines of code. Experiments have been done on a database containing 3000 3-dimensional points organized into polygons to describe a wire-frame image.
of the space shuttle. The experiments were run on a Sun 3 workstation with four megabytes of memory. It takes approximately 25 seconds for initialization of ROSE and DTT to load the necessary code modules and allocate work areas. Two experiments with the reference insertion operation that removed 45% and 14% of the data in the shuttle object required 129 and 132 seconds of CPU time, respectively. The second experiment required more time even though it removed less data because the subobjects removed were nested deeper inside the space shuttle object, and because more temporary subobjects are created in the process of computing the result of the reference insertion operation. Another experiment using the aggregation split operation to put 25% of the data in the shuttle object into a new object required 120 seconds of CPU time to complete. In all these experiments, much of the time is spent in circumventing the type system of ROSE as is discussed below. The authors are unaware of other systems to which these performance figures can be compared. Since these figures are for the ROSE Data Translator only, no import/export routines are needed.

Several important lessons where learned during the design and implementation of DTT. As mentioned above, ROSE was chosen for the implementation of DTT because of its flexible data model, extended relational algebra for editing data structures, and because ROSE could assume most of the responsibilities for memory management. In the end, we were not able to exploit all of these features to the degree expected.

The biggest problem resulted from the fact that the ROSE data model is strongly typed, and data translation can be viewed as breaking and modifying a type system. As a result, much effort was required to circumvent the ROSE type system in many situations, and this had a major negative impact on the performance of DTT. The problem is that the type of a name in ROSE, whether a domain name or attribute name, cannot be changed once declared. Any operation that changes the structure of an object changes the domain for that object. Thus, in ROSE, the modified domain must be given a new name, and the object completely copied from the old domain to the new domain with appropriate changes. As a result, large amounts of time and space are spent in copying objects. This also means that the final object has different attribute and domain names from the original object. A postprocessing step is needed to convert these names back to the original names. Thus, the first lesson learned is that a flexible data model is not sufficient by itself. A mechanism is also needed to redefine type names and temporarily break the type system so that objects can be changed in place rather than requiring that they be copied every time a change is made.

The second problem concerns memory management. Since ROSE assumes full responsibility for memory management, we were not able to monitor and influence how memory was managed. Thus, for example, it was not possible to base decisions on when to flush objects to secondary memory on how full the main memory workspace was becoming. As a result, we were forced to use a conservative policy of flushing all objects to secondary storage after each operation unless the object was needed by the next operation. This also has a significant negative impact on the performance of DTT. Thus, lesson two is that management of both main memory and secondary storage is critical to the performance of data translation, and tools are needed to monitor and influence the way it is done.

The ROSE Data Translator and DTT are being developed as part of the DARPA Initiative for Concurrent Engineering (DICE), and are being enhanced and modified to integrate with other tools being developed as part of this project. This new version of DTT will allow both problems above to be partially solved. The performance of DTT should be greatly improved as a result. This new version of DTT will be implemented with compiled code rather than with the interpreter of the original ROSE 1.0 system. This should also improve the performance of DTT.

Conclusions

The goal of DTT is to provide a flexible and user-friendly tool for generating data translators to support exchange of data between engineering systems. Such a tool is needed in any dynamic system with heterogeneous hardware and software systems if these systems are to share data in a flexible and efficient way.

The major component of DTT is a ROSE Data Translator. This system provides a set of operations useful for translating the structure of a ROSE database. The ROSE data model includes several forms of abstractions which allow it to model a wide variety of types of data, including complex engineering objects. Thus, to translate data from one engineering system to another, the data is first converted into a ROSE database, and then the ROSE Data Translator used to change its structure to that required by another application. Once this is complete, the data is exported from the final ROSE database to the database of the target application system.

Future work will enhance DTT and integrate it into the suite of tools being developed for management of data in the DICE project. DTT is expected to have a major role in integrating heterogeneous engineering systems within this project. As part of this project, a new version of ROSE, called ROSE-IC, and a new protocol for managing ROSE objects, call the ROSE File Object Protocol (ROSE FOP), are being developed [Har89]. ROSE-IC extends the capabilities of the original ROSE 1.0 system to include such things as support for concurrent editing of objects and cooperative engineering work environments. Its current implementation is as a set of classes in the Objective-C programming language. As a result, it provides persistent objects for Objective-C programs, along with version control for persistent objects, scripting of changes made to persistent objects, and other functions related to managing concurrent editing and cooperative work environments for persistent objects. As mentioned above, DTT is being re-implemented using ROSE-IC, and this implementation is expected to overcome many of the problems of the original DTT system discussed above.

References


