An IDL to Ada95 Mapping to Support Propagation Modeling *

D. Needham
Computer Science Dept.
United States Naval Academy
Annapolis, Maryland 21402
Telephone: (410)293-6812, Fax: (410)293-2686
needham@scs.usna.navy.mil

S. Demurjian and T. Peters
Computer Science & Eng. Dept.
The University of Connecticut
Storrs, Connecticut 06269
Telephone: (860)486-4818, Fax: (860)486-4817
{steve,tpeters}@eng2.uconn.edu

ABSTRACT

Representing dynamic interdependencies between design objects is an essential part of modeling the critical software communications found in complex software systems. This paper investigates the modeling of propagations (our term for dynamic interdependencies), which are captured using design-level triggers for specifying dynamic behavior across object types. We focus on the CORBA-compliant utilization of our propagation model to support distributed, propagation-focused applications. We develop a propagation-specific interface in IDL, and examine Ada 95 source code that meets the requirements specified by the IDL design. An example of the CORBA-compliant propagations involved in the maintenance of topological invariance in the industrial domain is examined.

1 Introduction

Representing dynamic interdependencies between design objects is an essential part of modeling the critical software communications found in today's complex software systems. Current software design methodologies, software development environments (SDEs), and computer aided design (CAD) systems lack the ability to represent and manipulate these interdependencies, even though the specific information that must be represented is often well understood by the designer. For example, a CAD designer altering the pitch of a gas turbine engine's fan blade may need to consider the stress where the blade is mounted to the turbine disk. Changes in the stress (caused by the change in fan blade pitch) are said to *propagate* to the attachment of the fan blade to the disk. *Propagation modeling* seeks to represent such interdependency knowledge as an integral part of the design process.

This paper furthers our previous propagation modeling work [11] by considering the communications typically found in object-oriented, distributed environments. The CORBA [6] (Common Object Request Broker Architecture) model released by the OMG (Object Managament Group) seeks to transparently provide interoperability among heterogeneous applications. CORBA focuses on allowing objects to interact with each other irregardless of the languages and/or hardware architecture used to develop the individual objects. A CORBA-compliant propagation model can be used to raise the application-specific activity of propagation modeling so that it promotes interdependency modeling between distributed, dissimilar applications. A critical need in the modeling

*The author acknowledges, with appreciation, partial funding for this work received from the Naval Academy Research Council and the Office of Naval Research under N0001498WR20010, and from the Naval Research Laboratory, Stennis Space Center under N0017399WR00226. The views expressed herein are of the authors, not of the funding agencies.

of distributed interdependencies is the description of the interface between the distributed objects. The IDL (Interface Design Language) developed by OMG as part of CORBA provides this interfacing requirement. An IDL specification serves as a contract between distributed objects that need to communicate with each other. Once an IDL specification has been developed, the distributed objects must meet the message passing requirements indicated by the IDL in order to ensure correct communication between the other objects in the system. This meeting of the IDL specification may be accomplished regardless of whether the objects were written in different programming languages or reside on different architectures.

Work related to our distributed propagation modeling efforts includes a subset of UML's [4] interaction diagrams referred to as *collaboraton diagrams*. Collaboration diagrams emphasize the structural relationships of objects that participate in an interaction by sending and receiving messages. Our work seeks to more formally apply the underlying notion of UML collaborations as a means to moderate the interaction of distributed objects.

Previous efforts at the programming language level have sought to provide programmers with the tools needed to resolve consistency maintenance issues. The APPL/A [14] language adds the concept of programmable triggers to Ada 83 [8]. The programmable triggers of APPL/A provide a mechanism through which programmers may capture portions of inter-object relationships. However, APPL/A's approach does not provide a means for the designer to specify which objects interact, when they interact, and precisely how the interaction takes place. In the related area of database triggers and alerters [5], the emphasis has been on implementation-time rather than design-time solutions to interdependency modeling.

The remainder of this paper is organized as follows. In Section 2, we briefly review an industrial domain propagation example. In Section 3, we present a UML model for our propagation example using collaboration diagrams. In Section 4, we examine both the IDL and subsequent Ada95 mapping for our propagation, with an emphasis on the distributed nature of our modeling. In Section 5, we analyze our approach, present our conclusions, and discuss directions for future research.

2 An Industrial Domain Example

To better illustrate the issues surrounding our distributed propagation modeling efforts, we present an example propagation supportive of rapid prototyping a mechanical artifact within an industrial domain. Rapid prototyping technologies can utilize propagations to assist in allowing timely inclusion of experimental verification within an iterative design process.

Work introducing *topological tolerances* [12] as a tool for retaining desired design invariance during the production of successively refined prototypes has provided valuable insight towards the formation of our propagation model. For a more complete examination of the theoretical mathematical basis upon which topological tolerance is based, please see [1]. A simplified overview of this work, sufficient for our propagation modeling purposes, is that for any finite triangulated polyhedron S within R^3, there exists a scalar ν such that any vertex in S may be perturbed a distance of less than $\nu/2$ without altering the topological form of S.

The propagation scenario we have chosen centers on change notification surrounding a decision to alter supporting geometry data structures for rapid prototyping. At each successive local modification of the geometric data representing S, the effected topological tolerances must be dynamically updated. These updates provide a useful basis from which to examine design-time propagation modeling issues.

The "... *de facto* solid freeform fabrication industry standard..." [3], called the .STL file format, is a data representation that consists of triangulations of the boundary surfaces of a solid.

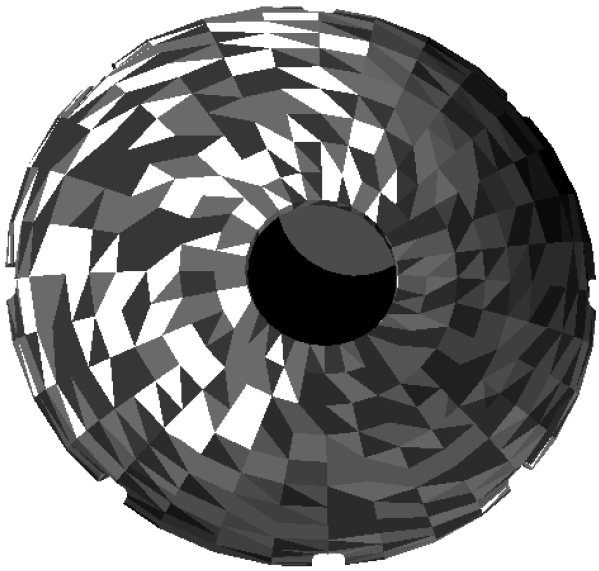

Figure 1: The P&WA Ball Box.

The emphasis of our propagation modeling is to use propagation to help in the preservation of the topological integrity of a .STL file when the part geometry is being modified during rapid prototyping. Specifically, if a vertex perturbation is allowed from a topological tolerance viewpoint, then the geometric data representing the .STL file is in turn modified via a propagation.

For our purposes, a representative subset of an industrial .STL file from Pratt and Whitney Aircraft (P&WA) will be described. The overall design is known as a ball box [13] (please see Figure 1) and includes over 5,700 facets. Visual inspection was used to decide a candidate area on the inside of the ball box (please see Figure 2) for local editing (vertex perturbation), and that subset is shown in Figure 3. As a first step in deriving the culling set, a particular vertex was chosen, shown in Figure 3 as the vertex common to the 90 degree angles of the black and dark grey triangles at the center of the image. An initial candidate for the appropriate culled set was selected by visual inspection, and a representative point of the corresponding actual .STL data is given in Table 1 as indicated by the arrow.

3 Modeling Propagation

For editing triangulated data, it is clear that all possible modifications can be accomplished by editing individual vertices. While achieving the full desired edit may require moving many vertices, the individual vertex movement is the focus of our attention. Hence, natural object candidates for an object-oriented design of the .STL file editing we have described are:

1. A vertex, the subject of a particular edit.

2. The portions of the .STL file that are effected if a given vertex is edited (moved).

A UML modeling of the Ball Box application appears in Figure 4, which represents the object types and inheritance, containment, and dependency relationships found in our model of the Ball Box example.

In Figure 4, five object types appear: `Triangle`, which represents a triangle making up part of S, `STL_File` which represents the abstraction for a physical .STL file, `Vertex` which models a

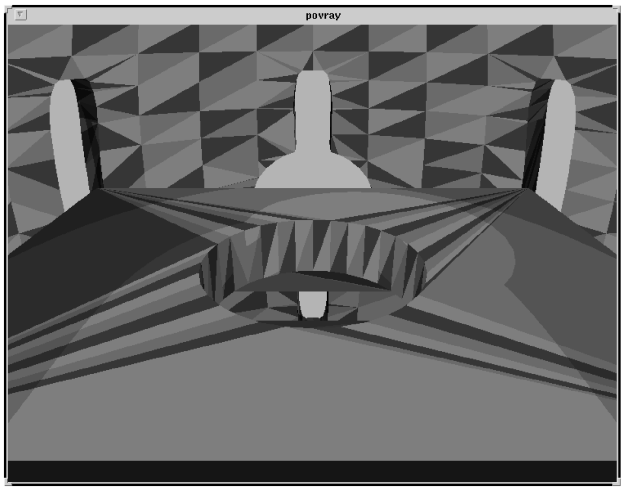

Figure 2: Internal View of the P&WA Ball Box.

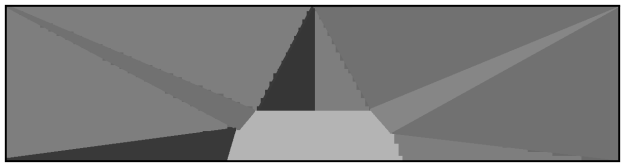

Figure 3: Selected Region of P&WA Ball Box Interior.

vertex of S, `Intact_STL` which allows treatment of the entire .STL file, and `V_Culled_STL`, which represents a vertex specific subset, possibly improper, of S, culled to contain only the features of S effected if the given vertex is perturbed a distance less than $\nu/2$. Note that `Intact_STL` and `V_Culled_STL` both inherit from `STL_File`. As shown in Figure 4, a containment relationship exists between a triangle and the three verticies that make up the triangle.

The Ball Box example from Section 2 contains interdependencies between its design objects. These interdependencies are represented in Figure 4 as the `Update_Vertex` dependency relationship (propagation) between `Vertex` and `V_Culled_STL` instances. A fundamental action that must be undertaken upon moving a vertex v less than $\nu/2$, is to update that subset of the .STL equal to star(v), where star(v) is defined as the union of all of the facets containing v as a vertex [2]. Since `Vertex` represents a particular vertex being moved, and `V_Culled_STL` represents the portion of the .STL file that is effected by this movement, we can focus on a simple propagation circumstance, where a simple numeric comparison, made by a vertex instance, has determined that the total intended perturbation distance is strictly less than $\nu/2$.

For clarity, we will refer to the interdependency relationship between a `Vertex` and a `V_Culled_STL` as the `Update_Vertex` propagation. The sequence of the propagation, known to the designer of the ball box application, that should be embedded within the control logic of *Update_Vertex*, is as follows:

1. The `Vertex` instance provides the `Update_Vertex` propagation with a new location as a result of the designer's edit request.

2. The `Update_Vertex` instance calls upon the `V_Culled_STL` instance to determine an updated ν value for the edited vertex. This updated ν value is then passed back to `Update_Vertex`.

```
FACET NORMAL 8.4186706E-02 -8.5468310E-01 5.1227863E-01
  OUTER LOOP
    VERTEX 6.04999992E+00 7.37871887E+00 5.20000014E+00
    VERTEX 6.00000007E+00 7.38364388E+00 5.20000014E+00
--> VERTEX 6.00000007E+00 7.33035169E+00 5.11108773E+00
  ENDLOOP
ENDFACET
FACET NORMAL 8.4186706E-02 -8.5468310E-01 5.1227863E-01
   OUTER LOOP
--> VERTEX 6.00000007E+00 7.33035169E+00 5.11108773E+00
    VERTEX 6.00000007E+00 7.38364388E+00 5.20000014E+00
    VERTEX 5.95000023E+00 7.37871887E+00 5.20000014E+00
  ENDLOOP
ENDFACET
```

Table 1: Subset of STL Data For P&WA Ball Box.

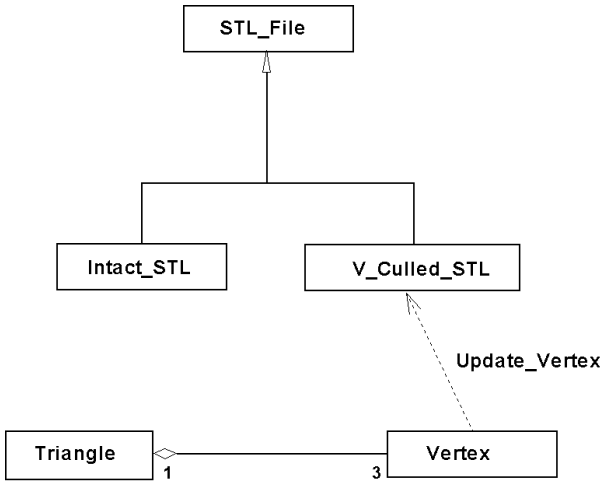

Figure 4: P&WA Ball Box Object Types.

3. `Update_Vertex` passes the updated ν value back to the `Vertex` instance.

4 Distributed Interprocess Communication

In this section, we examine the interprocess communication required by the P&WA Ball Box `Update_Vertex` propagation. Our focus is on the conduction of runtime propagation-required communication in a distributed fashion supportive of a client/server-based model for propagation. Distributing the communication required in our P&WA Ball Box propagation example allows the design editor perturbing a vertex to be located in a different environment from the .STL-based apparatus being used to render the boundry surfaces of the solid being fabricated. Specifically, our goal is to support distribution so as to allow the design editor to edit a .STL file over the Internet.

Using a UML collaboration diagram, Figure 5 shows the communication between the `Vertex` instance and the `V_Culled_STL` instance as overseen by the `Update_Vertex` propagation. The CORBA ORB abstracts the interobject communication necessary to establish the CORBA-based communication between the interrelated objects. As indicated in Figure 5, the `Update_Vertex`

propagation is a transient object, created when a `Vertex` is perturbed. Part of this creation process involves associating the vertex being perturbed with the appropriate portion of the .STL file. In particular, the `Update_Vertex` propagation determines the new location of the perturbed vertex, and then ensures that the `V_Culled_STL` updates its value for ν as needed.

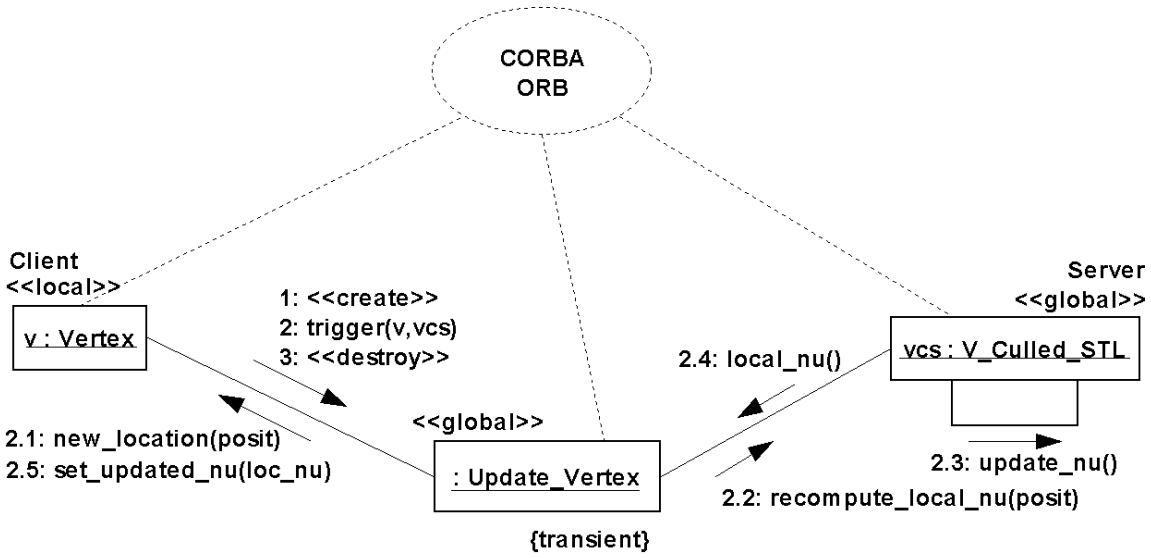

Figure 5: Ball Box Interprocess Communication (UML Collaboration Diagram).

4.1 An IDL-based Propagation Specification

Modeling interprocess communication in IDL is an important part of developing a CORBA-based application. IDL is a universal notation for describing software interfaces. Although an IDL specification appears as software "code", IDL is not an actual implementation. Instead, language bindings are used to translate IDL into the proper constructs for the targeted software language. IDL makes it possible for client software to be implemented on dissimilar hardware architectures and programming languages from those used for server object implementations.

Figure 6 gives the IDL specification for the interprocess communication found in our Ball Box example from Figure 5. The `trigger` operation is used as an abstraction for the inter-dependent communication between the `Vertex` and `V_Culled_STL` objects. The specific sequence of messages being passed between these objects, as shown in Figure 5, is not specified in the `Update_Vertex` propagation's IDL specification. The IDL specification focuses on what communication the inter-dependant objects require of the propagation. All of the other operations shown in Figure 6 serve to initially set up for the invocation of the propagation, and are not directly related to the inter-dependencies between the `Vertex` and `V_Culled_STL` objects. Figure 6 also shows the `ICViolation` exception. This exception is raised by the propagation's `trigger` operation if, during the course of servicing the object interdependencies, either of the objects are asked to modify themselves in a manner inconsistent with their internal integrity constraints [11].

4.2 Ada95 Propagation Specification

Figure 7 shows the Ada95 source code resulting from using MITRE's IDL to Ada95 translator [10] with the IDL from Figure 7 as input. The MITRE translator generates Ada95 client-side specifications. The implementation (not shown) for the `trigger` primitive operation given in Figure 7 contains the specific message passing that is conducted between the `Vertex` and

```
interface UpdateVertex {

   exception ICViolation{};
   attribute short propIdent;
   void create(inout Vertex Vertex_in,
               inout VCulledSTL VCulledSTL_in);
   void associateSrc(inout Vertex Vertex_in);
   void srcReady(inout Vertex Vertex_in,
                 inout VCulledSTL VCulledSTL_in,
                 inout string result);
   void trigger(inout Vertex Vertex_in,
                inout VCulledSTL VCulledSTL_in)
                      raises (ICViolation);
};
```

Figure 6: `UpdateVertex` Propagation IDL Specification.

`V_Culled_STL` objects as specified in Figure 5. Note that the specification for the `trigger` primitive operation is the only operation from the set of messages given in the UML design from Figure 5 that appears as a primitive operation in the Ada95 source code for the `Update_Vertex` propagation. The remainder of the primitive operations in Figure 7 are needed to invoke the propagation, and are not part of the interdependencies between the the `Vertex` and `V_Culled_STL` objects. The primitive operations given in Figure 5, less the `trigger` operation, that communicate directly with the `Update_Vertex` propagation are used in the implementation of the `trigger` primitive operation. The `update_nu` operation labeled as 2.3 in Figure 5 indicates the requirement for such a primitive operation on the `V_Culled_STL` object type.

5 Analysis and Conclusions

In this effort, we sought to provide a vehicle for more fully describing CORBA-based object inter-dependency through the use of UML collaboration diagrams and IDL specifications. We found IDL to be useful in describing a component's boundaries, and allowing for focus on the contractual interfaces with the potential clients of a component. An IDL specification seeks to include a description of resources that a server component intends to expose to its clients. From our work, it is clear that IDL by itself does not have the necessary constructs to meaningfully describe the interprocess communication required by a software propagation. In particular, the triggering operation of a propagation needs to have access to and invoke operations on the inter-related objects. Such inter-dependancy knowledge is not represented when a model is translated into an IDL specification. We used UML collaboration diagrams as the basis for describing the interdependencies between design objects. Most of this rich design knowledge was lost when an IDL specification was developed from the UML collaboration diagram.

MITRE's IDL to Ada95 translator was used to generate the Ada95 source code for our industrial example. The translator generates Ada95 specifications for the client side (only) of a CORBA-based system. To date, MITRE has not further developed the translator to include mapping on the server side. It is clear that further automation of propagation modeling in CORBA-based environments would be beneficial. Such automation would serve to lessen the loss of design knowledge that occurs when a UML collaboration diagram representing a propagation is translated to IDL.

The distributed communication provided by CORBA has proven to be useful in implement-

```
with Corba;
with Corba.Object;
package UpdateVertex is
    type Ref is Corba.Object.Ref with null record;
    ICViolation : exception;

    procedure create( Self          : in Ref;
                      Vertex_in     : in out Vertex;
                      VCulledSTL_in : in out VCulledSTL );

    procedure srcReady( Self          : in Ref;
                        Vertex_in     : in out Vertex;
                        VCulledSTL_in : in out VCulledSTL;
                        result        : in out Corba.String );

    procedure trigger( Self          : in Ref;
                       Vertex_in     : in out Vertex;
                       VCulledSTL_in : in out VCulledSTL );
                        -- This procedure raises the exception: ICViolation

    function To_UpdateVertex( The_Ref : in Corba.Object.Ref'class ) return Ref;
    function To_Ref( From : in Corba.Any ) return Ref;
    function To_Any( From : in Ref ) return Corba.Any;
    ...
end UpdateVertex;
```

Figure 7: `UpdateVertex` Propagation Ada95 Specification.

ing the communication required by distributed propagations. The propagation behaves as a transient instance that exists only to service the interdependencies between objects. It is not immediately clear whether the propagation instance itself is better suited to being regarded as a CORBA client or server or both or neither. Further work in this area is needed to determine the client/server-specific nature of the relationship between a propagation and the inter-dependent objects' communication overseen by the propagation.

References

[1] L-E. Andersson, S. M. Dorney, T. J. Peters, N. F. Stewart, "Polyhedral Perturbations That Preserve Topological Form", *CAGD*, 12 (1995) pp. 785-799.

[2] R. H. Bing, "The Geometric Topology of 3-Manifolds", *American Mathematical Society Colloquium*, Volume 40, Providence, RI: American Mathematical Society, 1983.

[3] J. H. Bohn, and M. J. Wozny, *A Topology-Based Approach for Shell-Closure*, IFIP Transactions, Geometric Modeling for Product Realization, eds., P.R. Wilson, M.J. Wozny, and M.J. Pratt, Aug. 1992, pp. 297 - 320.

[4] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[5] C. Carlson and K. Adarsh, "Toward the Next Generation of Data Modeling Tools", *IEEE Trans. on Software Engineering,* Vol. 11, No. 9, Sept. 1985.

[6] OMG, "The Common Object Request Broker Architecture and Specifications", July 1995, revision 2.0.

[7] Department of Defense, *Ada 95 Reference Manual*, International Standard, ANSI/ISO/IEC-8652:1995, Jan. 1995.

[8] Department of Defense, *Reference Manual for the Ada Programming Language*, ANS/MIL-STD-1815A-1983, Feb. 1983.

[9] ICAD Design Language Documentation, Provided by Pratt & Whitney Aircraft, East Hartford, CT, 1995.

[10] MITRE, IDL to Ada'95 Translator Design Language Documentation, Bedford, MA, 1995.

[11] D. Needham, S. Demurjian, K. El Guemhioui, T. Peters, "An Ada95 Basis For Propagation Modeling", Proceedings of TRI-Ada'97, St. Louis, Missouri, Nov. 11-14, 1997, pp. 263-272.

[12] T. Peters, S. Demurjian, D. Needham, R. Peters, S. Dorney, "Propagating Topological Tolerances for Rapid Prototyping", Proceedings of the International Mechanical Engineering Conference and Exposition (IMECE), MED-Vol 4, Atlanta, Georgia, Nov. 17-22, 1996, pp. 487-498.

[13] STL Formatted Ball-Box Representation Documentation, Provided by Pratt & Whitney Aircraft, East Hartford, CT, 1995.

[14] S. Sutton, D. Heimbigner, and L. Osterweil, "Language Constructs for Managing Change in Process-Centered Environments", *Proc. of the Fourth Annual ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, Dec. 1990.