Generating a Foundation for Concurrent Engineering

Federal Manufacturing & Technologies

N. C. Christensen

KCP-613-5857

Published March 1997

Final Report

Approved for public release; distribution is unlimited.

RECEIVED
MAY 12 1997
OSTI

MASTER

Prepared Under Contract Number DE-AC04-76-DP00613 for the
United States Department of Energy

AlliedSignal
AEROSPACE

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade names, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Printed in the United States of America.

This report has been reproduced from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P. O. Box 62, Oak Ridge, Tennessee 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U. S. Department of Commerce, 5285 Port Royal Rd., Springfield, Virginia 22161.

A prime contractor with the United States
Department of Energy under Contract Number
DE-AC04-76-DP00613.

AlliedSignal Inc.
Federal Manufacturing & Technologies
P. O. Box 419159
Kansas City, Missouri
64141-6159

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

KCP-613-5857
Distribution Category UC-705

Approved for public release; distribution is unlimited.

GENERATING A FOUNDATION FOR CONCURRENT ENGINEERING

N. C. Christensen

Published March 1997

Final Report
N. C. Christensen, Project Leader

Project Team:
J. J. Zimmerman

HH
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

AlliedSignal
AEROSPACE

CONTENTS

Section	Page
Abstract.....	1
Summary.....	1
Discussion.....	3
Scope and Purpose.....	3
Prior Work	3
Activity.....	3
Situation	3
Problem.....	4
Solution	4
Protocycling.....	5
History	7
Generating Itasca From EXPRESS.....	7
Generator Implementation	10
Mapping EXPRESS to Itasca.....	13
Accomplishments	15
Future Work	15
Bibliography	17

ILLUSTRATIONS

<i>Figure</i>		<i>Page</i>
1	Advanced Manufacturing Development System (AMDS).....	6
2	Simple ATN	8
3	Portion of Working Form	10
4	Attribute Example	13

TABLE

<i>Number</i>		<i>Page</i>
1	EXPRESS Versus Itasca	14

ABSTRACT

Both Concurrent Engineering and the Agile Enterprise require as a foundation the low cost, timely sharing of information. Described is a cost-effective way to generate this foundation from the product data International Standard 10303 (informally called STEP). Also described is a prototype implementation.

SUMMARY

AlliedSignal, Federal Manufacturing & Technologies (FM&T), was the first facility in the world to manufacture a mechanical part using the then draft international standard (DIS) ISO 10303 STEP.

The Advanced Manufacturing Development System (AMDS) enabled this accomplishment. AMDS created a (DIS) ISO 10303 STEP-compliant database within two weeks of receiving documentation. This gave FM&T a significant head start over its competitors. AMDS responded so fast because AMDS was developed in parallel with STEP. Traditionally, vendors wait for DIS status before beginning implementation. This is because DIS marks technical completion of the standard, and traditional software development requires a stationary target.

AMDS using nontraditional software development was able to track a moving target. AMDS developed a Generator that automatically transforms the STEP documentation into a STEP-compliant object-oriented database. The Generator and the STEP standard matured together. Databases were generated periodically over the years as STEP evolved. These databases tested STEP and AMDS concepts. After adapting the now mature Generator to the DIS version of STEP, AMDS generated a DIS-compliant database soon after receiving electronic copies of DIS STEP.

We were left waiting with no other STEP implementations available for data exchange. The first application of DIS AMDS was to test the testers. An organization developing test data for STEP was hand coding test files. AMDS helped check and correct those files.

Later AMDS helped some vendors debug the early versions of their STEP translators. This put us in a position to demonstrate STEP capability. The demonstration was to manufacture a connecting rod from STEP solid model data. A STEP file was send via E-mail from Ford Motor Company to FM&T. Demonstrating the information pool concept, the file was processed into AMDS and a new STEP file was created by AMDS. The new STEP file was input into CAM using the CAM vendor's translator. The connecting rod was machined using tool motions derived from a solid model in that CAM system.

The ISO TC 184/SC4 Secretariat Brad Smith recognized this success as a major milestone in STEP development. At his request, FM&T reported on this success at the international STEP meeting in Berlin, October 1993.

DISCUSSION

SCOPE AND PURPOSE

The purpose of AMDS was to develop an environment so highly productive that major software development projects could be done with a small staff. The scope was applying the environment to the implementation of the new product data standard informally known as STEP. The implementation is a STEP-compliant object-oriented database generated directly from the STEP documentation.

PRIOR WORK

AlliedSignal Federal Manufacturing & Technologies has been involved in STEP development since the system's inception in 1984.

ACTIVITY

Situation

Concurrent Engineering promises dramatic improvements over the traditional sequential method in product development cycle time. In Concurrent Engineering, specialists from all phases of production share information in a timely manner. The designer receives feedback on designs early, when it will do the most good. Feedback comes on producibility, critical cost factors, new technologies, environmental impacts, and changing customer requirements.

Concurrent Engineering requires sharing of information in a more complete and timely manner than is possible with traditional methods.

Concurrent Engineering and the Agile Enterprise are being pursued by world industry as the means to dramatic productivity improvements. Both of these methodologies require the low cost, timely sharing of information. Conceptually, what is needed is an information pool where information can be stored once and used many times. The creation of such an information pool offers significant productivity improvements in a traditional manufacturing environment, but the pool is the essential *foundation* under both Concurrent Engineering and the Agile Enterprise.

The major tasks in creating the information pool are:

1. Agreement of all information sharers on the content of the pool and the representation form for the information.
2. The implementation of the agreement as an information pool in hardware and software.

3. Interfacing all systems (vendor and in house) with the information pool.

Problem

With traditional software development, creation of the information pool is a horrendous task and maintenance of the created pool is a crushing burden.

Solution

Use a nontraditional approach to create the information pool. The fundamental innovation is to reverse an axiom of software design. That axiom is "Change is trauma." Our axiom is "Change is life."

The traditional "change is trauma" leads to the belief that the design must be stable (ideally frozen) before coding starts. Traditional software development processes strive to minimize change. Unavoidable changes are patched in trying to minimize impact to existing code. The traditional stability assumption leads to optimizing code for execution speed. This often makes code fragile with respect to change.

Our nontraditional "change is life" leads to concurrent development of both design and implementation. Our software development processes strive to minimize the trauma of change. Change itself is actively encouraged. After change, software is automatically regenerated from the EXPRESS, so radical global changes cost little more than trivial local ones. Our instability assumption leads to optimizing for maintainability. There is an execution speed penalty, but this is more than made up for by increased productivity in software development.

The major tasks are addressed as follows:

1. The new International Standard ISO 10303 (informally known as STEP) becomes the agreement on information content and form. Note: For Agile Manufacturing's virtual enterprise, this greatly increases the likelihood that two companies with no previous contact will be able to speak the same language (that is, STEP).
2. The software implementation of the agreement is generated automatically from STEP. STEP is documented in EXPRESS, a formal information modeling language. The Generator is driven by EXPRESS. Note: company-specific EXPRESS models can be input into the Generator to seamlessly mix company-specific information (for in house use) into the information pool.
3. Market pressure and government requirements are forcing vendors to make STEP interfaces to their products. With a world market, high quality interfaces at reasonable prices are expected.

Task 1 and Task 3 are being accomplished by others to satisfy their own needs. This paper assumes the completion of Tasks 1 and 3 and addresses only Task 2.

The Generator could have been developed using traditional methods and still reaped ample benefit over direct hand coding of STEP's thousands of pages. We choose to continue being nontraditional.

Protocycling

Our Task 2 simplifies to the creation of a Generator that inputs EXPRESS and outputs software for an information pool. Our project to develop the Generator and create the information pool is the Advanced Manufacturing Development System (AMDS). See Figure 1. Our development approach is protocycling. Protocycling replaces the traditional one development cycle (requirements, design, make, support) with multiple small development cycles. A protocyple (small development cycle) may add a capability, adapt an existing capability, or change direction (throwing away previously written code).

The protocyple approach was necessary because we were tracking a moving target. EXPRESS, still in development, was fluctuating and the information pool concept was new and not well understood. We wanted to ability to evaluate several alternate approaches.

Protocycling requires a very productive programming environment and a development method very responsive to change. We decided to start with a powerful artificial intelligence language called KEE. KEE is object-oriented and lisp-based.

Implementation of the Generator is an object-oriented implementation of augmented transition networks (ATN). ATN are well known and in their pure form are used in schools to illustrate computer science language concepts. Creating an ATN requires decomposing a process into simple cause and effect pairs. The ATN's value in teaching comes from its making obvious these causes and effects. The value to the Generator is that this obviousness greatly improves productivity when making revisions.

The fine-grain decomposition into cause and effect pairs also greatly reduces ripple effects which are the bane of software maintenance.

ATN principles are used in commercial software, but many compromises are made for efficiency. We accept the inefficiencies of pure ATN to get its maintainability. Besides, the end users are affected by inefficiencies in the generated information pool, not by those in the Generator.

For a large software project like our Generator, the functionality of ATN needed to be increased--for example, being able to divide the ATN in smaller reusable pieces analogous to subroutines.

Figure 1. Advanced Manufacturing Development System (AMDS)

Our priorities for the Generator are (highest to lowest):

- Maintainability
- Functionality
- Execution Speed

Having the Generator create the information pool *in toto* was impractical. Instead, the Generator creates classes and methods for the Itasca object-oriented database.

History

1989 Conceived AMDS

1990 First AMDS generated and tested with hand-built files.

1990 First STEP transfer using AMDS (SDRC's GEOMOD to Spacial Technology's ACIS)

1991 Experimental Exchange NURBS to analytical surfaces.

1993 Upgrade AMDS to Draft International Standard (DIS) release of STEP.

1993 Meeting in Berlin of ISO TC 184 /SC4--reported the first manufacture of a mechanical part using DIS STEP.

1995 Upgrade AMDS to International Standard release of STEP.

Generating Itasca From EXPRESS

The task is to generate Itasca statements from EXPRESS. Itasca is an object-oriented database, and EXPRESS is the information modeling language used to document STEP. The dynamic state of both STEP and EXPRESS made inadvisable the use of traditional software development methods. There was also uncertainty about the best way to map EXPRESS into Itasca.

To allow quick response to changes, the Generator was conceived as a augmented transition network (ATN) paired with an object-oriented working form (WF). The ATN isolates each fine-grained stimulus/response. This isolation simplifies making changes and minimizes ripple effects from changes. In the WF, inheritance aides sharing of code while polymorphism permits customization.

The ATN parses the EXPRESS and populates the WF as a side effect of the parse. The WF then creates the Itasca classes and methods. Note: with reasonable effort, the WF could generate software in other languages as well.

Context-Sensitive Languages

Augmented Transition Networks normal use is as a teaching aid for context-sensitive languages. Languages may be context sensitive or context free. In a context-free language, a statement can be understood without knowledge of preceding statements. For example, in a computer assembly language the statement "addreg 1 7707" means add the contents of memory location 7707 to register 1 regardless of preceding statements

In a context-sensitive language like English, meanings can be changed by preceding statements. So "good night" can be a polite parting in "Good night, Irene" and a poetic reference to death in "Do not go gentle into that good night," the poem by Dylan Thomas.

Seeking to become more intuitive, computer languages become more English like. So, parsers must deal with context-sensitive computer languages. ATN is a useful teaching aid.

Arcs and Nodes

The basic components of an augmented transition network (ATN) are arcs and nodes. A node represents state. An arc represents a change of state. An arc may only be traversed in one direction. An example is Figure 2. Simple ATN. This example shows a typical task parsing a sequence of characters into either names and integers as appropriate.

Figure 2. Simple ATN

We start at node *Wait* and wait for a stimulus (input).

Two arcs leave node *Wait*. If the stimulus is an alphabetic character, we

traverse the arc to node *Name*. If the stimulus is a numeric character, we traverse the arc to node *Integer*.

If at node *Name* and the stimulus is an alphanumeric character, we traverse the arc which loops back to node *Name*. If the stimulus is a space, we traverse to node *Wait*.

If at node *Integer* and the stimulus is a numeric character, we traverse the arc which loops back to node *Integer*. If the stimulus is a space, we traverse to node *Wait*. If the stimulus is an alphabetic character, it is an error.

Side Effects

Just traversing an ATN is not very useful. The word "augmented" in augmented transition network indicates that traversing an arc may have side effects. In Figure 2, the side effects need to be the creation of either a name or an integer.

The side effect of traversing the alphabetic arc from node *Wait* to node *Name* is to start the accumulation of characters to form the name. Each traversal of the alphanumeric arc appends the stimulus character to the end of the name. Traversing the space arc from node *Name* causes completion of the name's creation.

The side effect of traversing the numeric arc from node *Wait* to node *Integer* is to start the accumulation of numeric digits to form an integer. Each traversal of the numeric arc appends the stimulus character to the end of the integer. Traversing the space arc from node *Integer* has the side effect of converting the accumulated numeric characters into a true integer.

An ATN accomplishes its work as side effects of traversing arcs.

Networks and Subnets

For teaching purposes, arcs and nodes are sufficient. To use ATN to solve a real-world problem (that is, parse EXPRESS), more structure is needed. In the Generator, the arcs and nodes are grouped into subnets analogous to subroutines and subnets are grouped into networks analogous to programs. The subnets can be called recursively.

Stimuli are applied at the network level. In the Generator, two networks operate concurrently. The Syntax Network tokenizes. The Semantic Network populates the working form (WF). Individual characters from the EXPRESS file being parsed are asserted (input) into the Syntax Network as stimuli. Side effects in the Syntax Network assert tokens into the Semantic Network.

Blackboards

Subnets normally communicate by posting information to a blackboard. A blackboard is visible to all subnets in a network. Each network has a blackboard. Typical activity starts with the creation and posting of an instance (most often an

instance of class entity) to the blackboard. As subnets parse EXPRESS statements and phrases, attribute values are added to the posted instance.

Working Form

The working form (WF) is an object-oriented model of EXPRESS language. Classes in the WF represent concepts and structures from EXPRESS. For example, there is a class in the WF corresponding to the EXPRESS "entity" structure. Parsing an EXPRESS "entity" structure creates an instance of class entity in the WF.

Figure 3 shows a portion of the working form. The figure represents the subclass/superclass graph. Each name represents a class in the WF. The arrow represents a subclass relationship. The class pointed to is the super class. The naming style is to name classes as plurals.

The classes functions, entities, defined_types, attributes, and derived_attributes correspond one-to-one with structures in the EXPRESS language. The classes units, contexts, units_in_context, and constrained_units aid sharing of methods and attributes.

Parsing the EXPRESS will create instances of these classes. Invoking methods on the instances will create the Itasca statements corresponding to the EXPRESS.

Figure 3. Portion of Working Form

Generator Implementation

Because frequent change is expected, easy identification of what needs to change is essential. Two guiding principles in the Generator aid this identification. First is "one to one." Ideally, one EXPRESS language concept relates to one subnet in the ATN and one class in the WF. Second is consistent stylized naming. For example, the EXPRESS language ENTITY structure is parsed by the ATN entity network to create a WF instance of class entities.

Language Choice: KEE

The implementation language for the Generator is the Knowledge Engineering Environment (KEE) from Intellicorp. The rule of thumb for deciding to use an AI approach is "If neither the problem nor the solution is well understood, it's a good candidate for an AI approach." At the beginning, the only thing we were sure of was change. EXPRESS, the source information modeling language, was still in development. Itasca, the target language, had only recently become a commercial product and its capabilities were evolving. When exploring in hostile territory, it's wise to take a powerful friend. We choose KEE, one of the most powerful AI languages available.

Nodes

Nodes are implemented as KEE classes and subclasses of the TN.NODES class (Note: KEE style is to name classes as plurals). The nodes are implemented as classes rather than instances because nodes are instanced for each call of a subnet. That is, each time a subnet is called, every one of its nodes is instanced. These node instances store the invocation-specific data so the subnets can be recursively called. Methods specific to the subnet are defined on the subnets nodes so the data is conveniently available (local attributes of the node with the method).

Arcs

Arcs are implemented as instances of class TN.ARCS. An arc's attributes include the *from node*, the *to node*, the test for when the arc is to be traversed, and the program statements to execute as a side effect of traversing the arc.

Arcs are stored in an attribute of its *from node*. Arcs are tested sequentially when a stimulus is applied to the node and the first arc with a true result is traversed. An arc's test may be an arbitrarily complex expression, but a short form is available for common tests: specific keyword, specific type (such as, number or simple identifier), "always", and "otherwise".

Subnets and Blackboards

A subnet is a useful collection of nodes and arcs analogous to a subroutine in typical programming languages. A subnet is implemented as an instance of class TN.NETWORKS which has an attribute for the included nodes (arcs are attached to the nodes). A subnet has a current-node attribute with an initial value of its starting node.

The blackboard (a common area for sharing information) is implemented as predefined attributes of a network.

Network Execution

The execution flow of a network goes this way:

1. Wait stimulus.
2. A stimulus is asserted to the network.
3. The network relays the stimulus to its current subnet.
4. The subnet sequential tests the arcs of the subnet's current node to find the arc to traverse. If no arc tests true, an error is signaled.
5. Once an arc is found, its side effects statements are executed with the stimulus as an input.
6. The *to node* of the traversed arc becomes the subnet's current node.
7. If the side effects included a subnet call, the called subnet becomes the new current subnet. The called subnets are stacked (first in, last out) so a return from a subnet simply pops the stack.
8. Go to 1.

Variations on ATN

During implementation, some variations on the ATN were introduced for pragmatic reasons.

Reasserts (recursive): Often a single stimulus serves more than one function. A simple example is the "+" in "12+98". The "+" not only functions as the addition operator but also as the terminator of the integer 12. Normally, a stimulus is consumed by traversing an arc. Permitting a stimulus to be reasserted (put back in the input stream) allows one stimulus to traverse multiple arcs, thus fulfilling multiple functions.

Once reasserting was implemented, an extension proved useful. Reassert was implemented as a stack (first in, last out). Stimuli not originally from the input stream can be put on the stack, and more than one can be stacked at once. This can be used to expand short hand notations where a single stimulus has the same effect as a sequence of stimuli.

Some useful classes were created to be super classes of nodes. The super classes share (by inheritance) useful methods and attributes among their child nodes. Defined super classes share functionality for accumulation, parsing algebraic expressions, and making subnet calls.

Working Form

The principle means of communication from the ATN to the WF is "call by name". When a token is found by the Syntax Network, the first operation is to find if the token is the name of an object (class or instance) in the WF. If the token is the name of an object, then that object is asserted into the Semantic Network. This

makes the ATN less sensitive to changes in EXPRESS (and the WF). For example, when "MOD" the modular operator was added to EXPRESS, "MOD" became an instance of WF class `binary_operators`. The ATN did not need to be changed.

In the WF, EXPRESS entities are treated in the obvious way: one WF instance of class entity for each EXPRESS entity structure. Attributes are less obvious.

Figure 4. Attribute Example, shows the WF representation of A-circle-radius, the radius attribute of entity circle. The "A" is a prefix for instances of class attributes. The entity name and attribute name are concatenated together because the attribute name by itself is not unique (that is, a cylinder has a radius too).

`Y-length_measure` is a type created during parsing of the EXPRESS.

`A-circle-radius` is an instance of both class attributes and class

`Y-length_measure`. Methods and attributes inherited from both parents combine to form the proper mix to generate the appropriate Itasca statements.

Figure 4. Attribute Example

Mapping EXPRESS to Itasca

Itasca is a fully functional object-oriented database. The Generator creates Itasca objects and methods which represent STEP. Each entity in EXPRESS becomes a class in Itasca with methods to read and write that class's instances in STEP physical file format. To eliminate the possibility of name conflicts and to specify STEP identifiers exactly as they appear in STEP documents, STEP identifiers are specified as strings which (transparent to the user) are changed to standard Lisp characters and interned in package STEP (a package is a name space in Lisp, and Itasca is a Lisp-based product.)

An EXPRESS attribute is mapped into an attribute and two methods, a get method to get the attribute value and a set method to set the attribute value. The methods are required because EXPRESS allows a derived attribute to mask an attribute and Itasca does not. See Table 1 below.

Table 1. EXPRESS Versus Itasca

EXPRESS	Itasca	Notes
Integer	Integer	
Real	Double float	
String	Simple string	
Enumeration	Symbol	
Array	Instance of class express-array	
List	Instance of class express-list	
Set	Instance of class express-set	
Identifier	Symbol in package a STEP	The macro s-name converts the string to a symbol.
Type	Instance of class step-type	
Attribute	Attribute + 2 methods	The methods are to get and set the attribute value.
Attribute, derived	Method	
Entity	Class with methods for writing and reading instances in STEP format.	

ACCOMPLISHMENTS

AMDS was successful.

We demonstrated the feasibility of concurrent design and implementation of software. Implementation experience improved the design as the design progressed.

With a single programmer we were able to compete successfully with projects having programmer teams. AMDS was a key enabler for FM&T's publicity coup with the Ford connecting rod. Our success came partly from starting well before STEP was a frozen design and partly from using an extremely productive programming environment.

AMDS demonstrated the information pool concept. Information formally black boxed inside the vendor's system is now completely visible. This visibility permitted AMDS to help debug immature STEP translators. For example, one of our first programs checked the Euler consistency of solid models. This was an easy task for us but awe inspiring to those with lesser tools.

AMDS remains one of the top in STEP implementation. We have inquiries for its uses by commercial industry. Ford Motor company has a copy of the generated database.

FUTURE WORK

Experience with AMDS and a preliminary effort to market AMDS have identified two needed major enhancements. The first is conversion of the Generator from KEE to a less costly language, and the second is a customized graphic users interface.

Experience has shown that the full power of KEE is not needed, and improvements in implementations of the Common Lisp Object System (CLOS) make CLOS a viable alternative. CLOS costs much less than KEE and is available on personal computers. Personal computer software has a larger market and lower pricing. Note: we have started promising research on automatic conversion of the Generator from KEE to CLOS.

Currently, revisions of the Generator are made using the standard KEE GUI (graphic user interface) and the Emacs text editor. This is too awkward for a commercial product. A GUI needs to be tailored specifically for the Generator. The new GUI will need three interfaces: one for the ATN, one for the WF, and one for WSN.

The ATN interface presents subnets as in Figure 2. Simple ATN plus menus for nodes with predefined capabilities--like Subnet caller). The user jumps to a called ##Subnet by mousing the calling node. The interface supports editing nodes (attributes and methods) and editing arcs (selection test and side effects).

The WF interface presents a hierarchy of classes. The user can create or edit classes and instances. The primary function of the WF is to create software, so this needs to be easy. Templates would be best. The user creates a template in the target language using an editor customized for the target language. Where items are to be computed by the WF, the user will type an example marked by font or color as an example.

(s-send "radius" circle1)

When the example is moused, a CLOS editor will appear with the statements that will calculate the value to be inserted.

(s-send (*role-name* x) circle1)

The third interface will be Wirth Syntax Notation (WSN). EXPRESS syntax is formally defined in WSN. WSN is not used in AMDS, but WSN would be a useful addition. WSN was not used before because it is not sufficient in itself and initially STEP did not contain it. WSN could aid ATN development by showing what inputs are allowed into a state. More important would be using a WSN of newly released EXPRESS to highlight changes from an existing ATN.

The three interfaces will have to be integrated, of course.

STEP is planning a new version of EXPRESS and a mapping language called EXPRESS-X. Both are ripe opportunities for AMDS.

BIBLIOGRAPHY

EXPRESS; ISO 10303-11:1994, Industrial automation systems - Product data representation and exchange - Part 11: The EXPRESS Language Reference Manual.

STEP Physical file; ISO 10303-21:1994, Industrial Automation Systems - Product Data Representation and Exchange - Part 21: Clear Text Encoding of the Physical File Exchange Structure.

WIRTH, N.; *"What can we do about the unnecessary diversity of notation for syntactic definition?"*, Communications of the Association for Computing Machinery, Volume 20, Number 11, November 1977.

Parasolid; Solid Modeler from Electronic Data Systems, Unigraphics Division.

SHAPES; Solid Modeler from XOX Corporation.