Manufacturing Integration using the STEP-NC DLL

Martin Hardwick,
ISO STEP-Manufacturing
RPI & STEP Tools, Inc.
hardwick@steptools.com

Abstract

The STEP-NC DLL is a data pipe for connecting design and manufacturing systems. It enables the assembly of data from design, process planning, machining and inspection applications into an integrated data set that shop floor applications can use to deliver new value to end users. Applications that have been implemented using the DLL include just in time simulation and verification, integrated machining and inspection, five axis cutter compensation, and supply chain traceability. This paper describes the STEP-NC DLL and its applications.

Introduction

Today shop machine tools are programmed using languages that describe movements as Goto (G) codes and machine functions as Machine (M) codes. The languages are very primitive and vendor specific. A similar confusion of languages once existed for computer printers. In the mid to late 80's these languages were super-ceded by a more descriptive language called Postscript. This language allowed the vendors to hide their codes inside the printers and allowed the users to program using one standard language.

Industry has wanted to replace the RS274D language for M and G codes with something more descriptive for more than 20 years. One barrier has been the level of computation necessary to translate three dimensional coordinates and orientations into displacements for motors controlling the five axes of a machine tool. Until recently it was not cost effective to perform these translations on the CNC even though doing so would make the machine more flexible.

Many projects have shown that shop floor machines can be made easier to operate if more information is sent to the CNC [1]. For example, the process used to prove-out a new CNC program can be stressful and labor intensive. An operator has to be ready to stop the machine if the cutting tool is about to damage something. If a model of the product was sent to the machine along with models of the fixtures, cutting tools and other variables, then the CNC could check for these collisions using geometry intersection algorithms.

The algorithms to perform a geometry intersection are already implemented in CAD/CAM systems. These systems run the tool paths against an evolving model of the part and make sure that none of the design constraints of the part are violated while also checking for collisions. What is needed is a method to transfer the information from the CAD/CAM systems to the CNC controls so that the simulations can be run on the CNC as well.

The first standards for CAD/CAM data transfer were developed in the 1980's. Initially there were national and focused on geometric data exchange. They included SET in France, VDAFS in Germany and the Initial Graphics Exchange Specification (IGES) in the USA. Later a grand unifying effort was started under the International Organization for Standardization (ISO). The new standard was to be for all aspects of technical product data and named STEP for the Standard for Product Model Data [2].

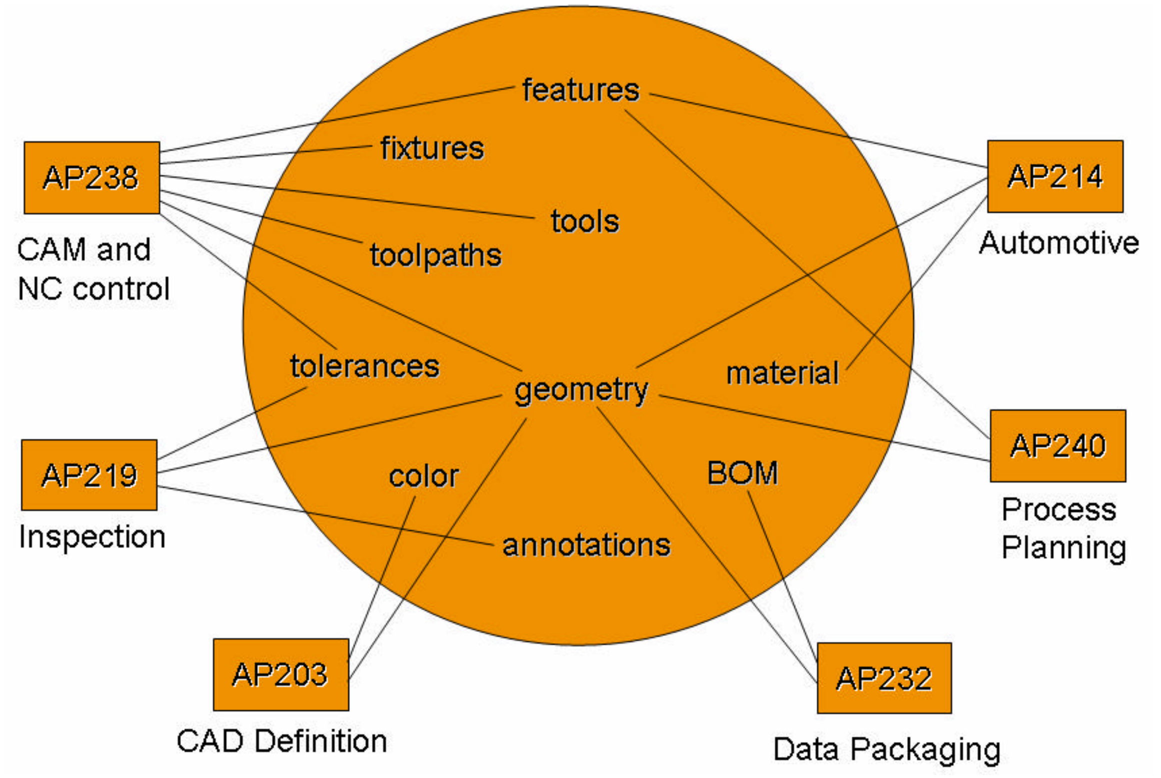


Figure 1. Information Defined by STEP for manufacturing

Today nearly every CAD system supports a STEP protocol for geometry data exchange. In the USA the protocol most widely used is AP-203 and in Europe it is AP-214. The two protocols are harmonized and can be processed by common software so the choice does not matter to most CAD vendors. In the late 90's this success allowed a European, Korean and Japanese team to begin developing a new 3D CNC programming language called STEP-NC [3]. The new language uses the geometry of STEP to define how a CNC machine should remove material volumes from a part. There are now more then 40 STEP Applications Protocols (see Appendix A). Figure 1 illustrates the range of information defined in these protocols for manufacturing applications.

This paper describes a Dynamic Link Library (DLL) for STEP-NC that can read, write, and process the data shown in Figure 1. This is the first time that a single library has been able to read, write and process manufacturing data with this range of complexity. Therefore, the STEP-NC DLL represents a new type of data pipe that can be used to connect design to manufacturing to enable more powerful manufacturing applications.

The next section describes how STEP translators are implemented. The third section gives a brief history of the efforts to make STEP applications easier to program. The fourth section gives an overview of the new STEP-NC DLL. The fifth section describes its implementation so that other vendors can implement competing systems. The sixth section describes the shop floor application testing that has taken place over the last three years. The seventh section summarizes the programming properties enabled by the DLL. The last section contains concluding remarks.

Overview of the STEP Implementation Methods

To implement a STEP translator the software developer needs to understand three things.

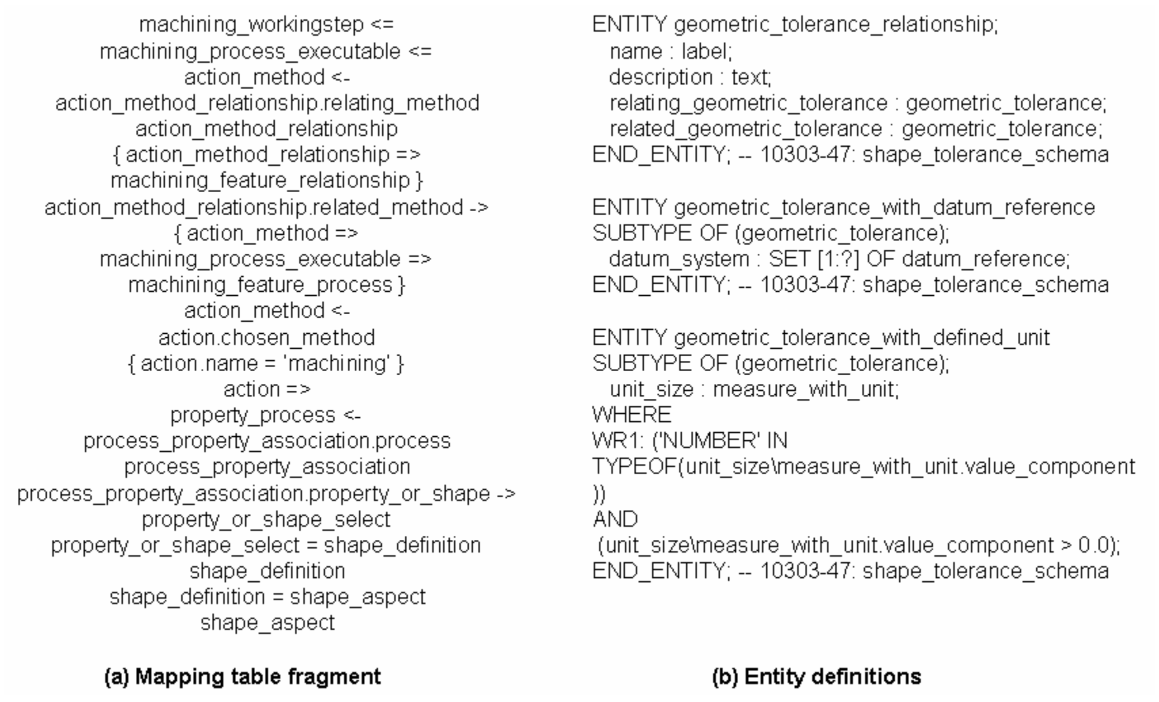
1. The first and most important is the scope and functionality of the Application Protocol. Each AP is divided into units of functionality whose information requirements are described by application objects. For a large standard there can be many objects. For example, there are 41 application objects in the AP-203 edition 1 CAD geometry standard, 237 application objects in the AP-224 process planning standard and 416 application objects in the AP-238 CNC machining standard.

2. The second is how each application object is represented as a set of entities defined by the STEP integrated resources. The integrated resources give STEP data the flexibility to grow as it moves across the engineering life cycle. For example, the value 5 in conceptual design becomes 5 millimeters in CAD, then 5 millimeters plus or minus 1 in process planning, then 5 millimeters plus or minus 1 millimeter as referenced from this datum plane on the CMM, and so on.

3. The third is how to encode the information as a data stream. In STEP the EXPRESS language is used to define all of the entities in the standard. EXPRESS has a very powerful inheritance model and a robust rule definition language. The powerful inheritance model is necessary because the STEP standards need to define entities for 3D geometry. The rule language is necessary so that all of the linkages required to represent an application object can be defined and checked. A file format for EXPRESS called Part 21 describes how entities with an EXPRESS definition are encoded in ASCII. Another file format called Part 28 describes how to encode the entities in XML. Programming tool kits usually manage this aspect of STEP implementation.

Figure 2a shows a mapping for one of the attributes of an Application object [2]. Figure 2b gives the EXPRESS language definition for some of the entities used in AP-238.

The mapping to the entities seems straight forward because all that is required is the mechanical translation of the mappings. However, this has to be done for all of the application objects. For most programmers this is too much. They are already having a lot of trouble understanding exactly how to use the application objects and to then have to break them down into about 5 entities each and meet the requirements of about 10 value and relationship constraints is too difficult. They become lost. Debugging the data is the biggest problem because searching for the cause of an error is like searching for a needle in a haystack. Only a few can handle the complexity and most of them are heartily glad when they can move on to a new project.



```
machining_workingstep <=
machining_process_executable <=
action_method <-
action_method_relationship.relating_method
action_method_relationship
{ action_method_relationship =>
machining_feature_relationship }
action_method_relationship.related_method ->
{ action_method =>
machining_process_executable =>
machining_feature_process }
action_method <-
action.chosen_method
{ action.name = 'machining' }
action =>
property_process <-
process_property_association.process
process_property_association
process_property_association.property_or_shape ->
property_or_shape_select
property_or_shape_select = shape_definition
shape_definition
shape_definition = shape_aspect
shape_aspect
```

(a) Mapping table fragment

```
ENTITY geometric_tolerance_relationship;
  name : label;
  description : text;
  relating_geometric_tolerance : geometric_tolerance;
  related_geometric_tolerance : geometric_tolerance;
END_ENTITY; -- 10303-47: shape_tolerance_schema

ENTITY geometric_tolerance_with_datum_reference
SUBTYPE OF (geometric_tolerance);
  datum_system : SET [1:?] OF datum_reference;
END_ENTITY; -- 10303-47: shape_tolerance_schema

ENTITY geometric_tolerance_with_defined_unit
SUBTYPE OF (geometric_tolerance);
  unit_size : measure_with_unit;
WHERE
WR1: ('NUMBER' IN
TYPEOF(unit_size\measure_with_unit.value_component
))
AND
 (unit_size\measure_with_unit.value_component > 0.0);
END_ENTITY; -- 10303-47: shape_tolerance_schema
```

(b) Entity definitions

Figure 2. STEP Mapping definitions and Entity definitions in AP-238.

A brief History of efforts to make STEP programming easier

STEP was a success at modeling product geometry. Today, nearly every CAD system includes an AP-203 or AP-214 translator.

However, after AP-203 was implemented it became clear that the larger information models were going to encounter greater implementation problems and a number of increasingly radical efforts were initiated to overcome these problems. The first were incremental in nature. They assumed that STEP implementation would be easier if standards were defined for manipulating STEP data. A series of such standards were defined for C, C++, IDL and Java [5]. However, the solution was incremental and time has shown that it was not sufficient.

The second solutions omitted the mapping from the application objects to the integrated entities. In STEP terminology the model of the application objects is called the Application Requirements Model (ARM), and the mapped model is called the Application Interpreted Model (AIM). Only the AIM is allowed to be implemented in a STEP standard, but sister standards have been developed that do not have this requirement.

For example, there are two models in the building and construction domain called the IFC model for the Industry Foundation Classes and the CIS model for the CIM Integrated Steel model respectively. Initially these standards were more popular than their equivalent STEP standards (AP-225 and AP-232) but now they are encountering problems as they try to extend themselves into second editions. The issue is that their second editions are not upward compatible with the first editions. For the STEP standards the same technology that allows them to share data across multiple applications domains also allows them to grow into second editions without requiring changes to existing applications.

The third set of solutions is the most radical. They try to develop better data sharing architectures using more mainstream technologies such as XML Schema. So far they have not succeeded because they are not able to model all aspects of manufacturing data. For example, XML Schema does not model inheritance relationships, and three-dimensional geometry with its many inheritance relationships is at the heart of STEP. What is needed is a way for these mainstream modeling languages to become more powerful without gaining the complexity that stopped the wider world from adopting EXPRESS in the first place.

Architecture of the STEP-NC DLL

For design to manufacturing data integration we need a solution that spans the CAD, CAM, CNC and CMM application domains. For the reasons described in the last section we cannot avoid the complexity, but STEP is an International standard so once a solution has been implemented it can be used for many years. Figure 3 shows the functionality of our solution. We implemented a Microsoft Dynamic Link Library (DLL) that uses a three schema architecture to manage complexity by dividing the problem into levels of abstraction. The three schemas are the AIM schema of STEP, the ARM schema of STEP and a new interface schema.

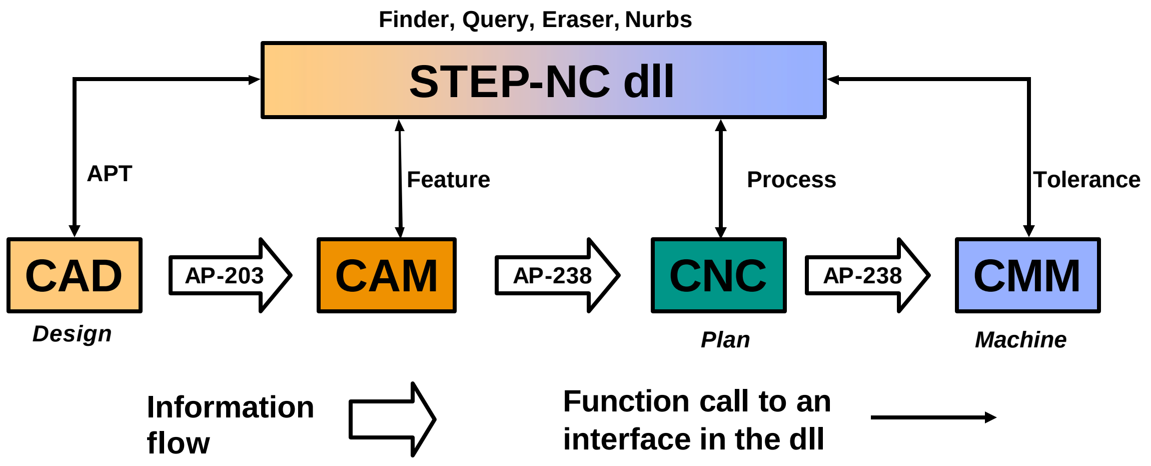


Figure 3. Functionality of the STEP-NC DLL

The interface schema is similar to the application view schema of database systems. Like a view schema, the goal is to create a layer that is safe and easy to use. Easy means that each interface defines something that is easy to understand within the context of the application. Safe means each interface defines operations that take the database from one valid state to another valid state, as defined by the EXPRESS schema of the relevant Application Protocol.

There can be many interfaces at the interface level. Unlike the ARM and AIM schemas they do not have to be defined slowly and carefully to ensure consistency and coverage without redundancy. Instead like a database view they can be defined in a more casual fashion to meet the currently understood requirements of an application. One of the key qualities is that an interface can always be replaced without harming the rest of the system.

Figure 3 shows the interfaces defined in the DLL for CAD, CAM, CNC and CMM applications. So far the APT interface has been tested using the Catia, UGS NX and Pro/engineer CAD/CAM systems. The feature interface has been tested using the Mastercam system. The process interface has been tested using the Siemens, Fanuc and Heidenhain CNC systems, and the Tolerance interface has been tested using the DMIS measurement language and the Zeiss CMM system. The other interfaces, finder, query, eraser and Nurbs are generic, and have been tested by all of the systems. Table 1 summarizes the roles of the interfaces

Table 1. The interfaces in the DLL.

APT is used to define the geometries of the product model. This includes the geometry of the tool paths which can be extracted from APT-CL files, RS274D files or by direct translators, and the geometry of the workpieces including the rawpiece, the final workpiece, the fixtures, and the cutting tools.
Feature is used to define manufacturing features such as pockets, slots and round holes. The features can be defined by geometry, by parametric quantities such as length and width, and by both.
Process is used to create enhanced process data for milling, drilling and turning operations by applications that are working directly with a CAM system.
Tolerance is used to create geometric tolerance and dimension data and to describe measurement methods for testing those tolerances and dimensions.
Finder is used to find all of the data.
Query is also used to find all of the data but unlike finder which can only return nominal values, query uses a more complex interface to also return the plus/minus limits of toleranced values.
Nurbs is used to convert geometric surfaces into Nurbs for display and to enable geometric calculations on those surfaces and their related tool paths.
Eraser deletes data.

As is allowed by the architecture, there is some redundancy between the interfaces. For example, many of the operations defined by the finder interface are duplicated and enhanced in the query interface by the addition of tolerance information. Thus query is more complete, but Finder is easier to use because information that can be accessed in one function call in Finder can take several in Query.

Implementation of the DLL

Figure 4 shows the internal architecture of the DLL. As shown the top layer of the DLL is a set of interfaces managed as Microsoft COM components. These interfaces each define a set of operations appropriate to the scope of an application. Appendix B lists the operations in the Feature interface.

The next level of the system consists of C++ code. This code implements the operations. The operations are designed to be transactions that take the database from one consistent state to another consistent state. Each operation queries, makes or modifies one or more application objects. Most operations work on several application objects because this is necessary to ensure consistency at the end of the operation. For example, an operation cannot create a workingstep without also making a feature because the STEP-NC model requires every workingstep to have a feature.

As noted in the previous sections making all of the entities required for all of the application objects is extremely tedious when done by hand. Automating the production of this code is one of the key innovations of the DLL. To do this we implemented a parser that reads the STEP mapping tables as shown in Figure 2a and generates C++ code. A STEP expert then annotates this data with information about the required data sharing (which is defined in the STEP integrated resources) and the result is a set of C++ classes containing the methods necessary to create each attribute of each application object.

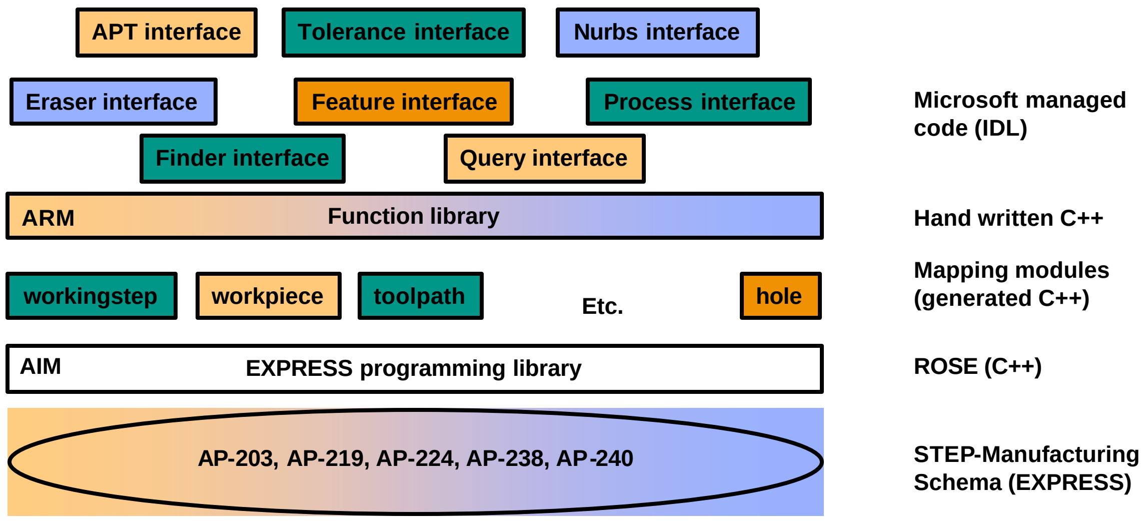


Figure 4. Internal Architecture of the DLL.

The code generated for the mappings is applied to an EXPRESS data management library. This library was previously implemented over a period of ten years and contains a large, robust set of functions for manipulating entities defined by EXPRESS. It has been used to implement STEP translators for many years with an estimated one million copies being deployed in various CAD stations world wide [5].

For the STEP-NC DLL a schema was developed to unify the data allowed by the STEP Manufacturing Application Protocols. This schema is called the STEP-Manufacturing schema. Currently it is a unification of the definitions given in AP-203, AP-219, AP-224, AP-238 and AP-240. Other Application Protocols can be added as new domains are added to the integration scope of the DLL.

Testing

Development of the STEP-NC DLL began at the end of the Model Driven Intelligent Control for Manufacturing (MDICM) program in 2004. The first round of testing in February 2005 showed how the DLL can be used to drive machining programs for five axis controllers. Two controllers, one using AB tool tilt and the other using BC table tilt, were run using data taken from four CAM systems. The demonstration while basic was important because if STEP-NC is to replace RS274D in the same way that Postscript replaced RS272, then it must be machine independent. Figure 5 illustrates this demonstration.

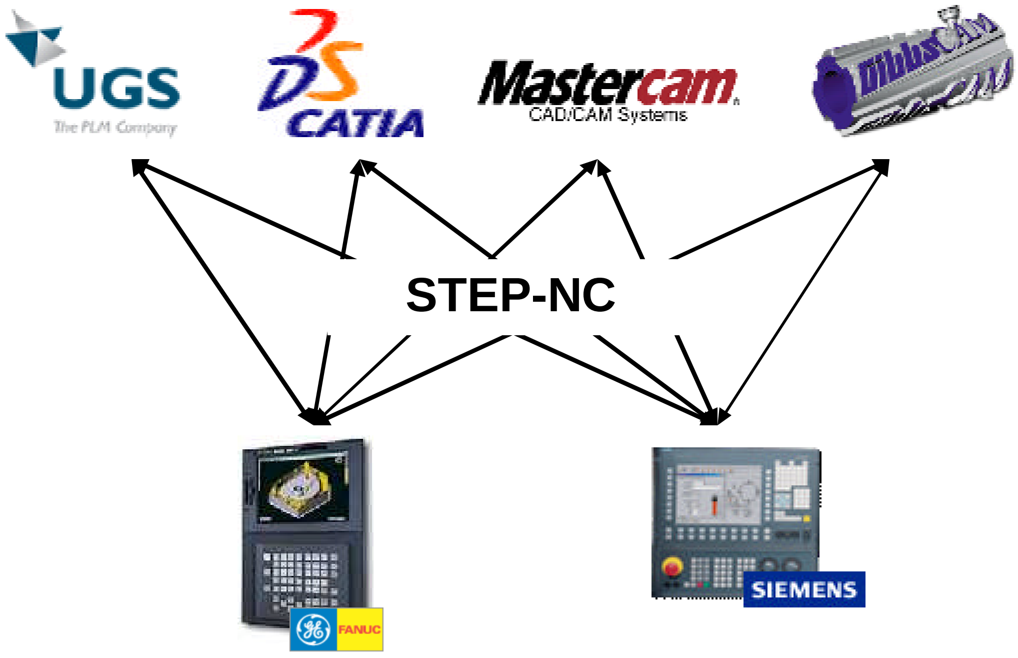


Figure 5. February 2005 Interoperability Demonstration

The second round of testing in June 2006 showed how a site can machine a part using a program created at multiple sites. A feature driven three-axis machining program for roughing was created using Mastercam, and a tool path driven five axis machining program for finishing was created using Catia. The two programs were combined using the STEP-NC DLL and used to machine the part at a third site.

The third round of testing in June 2007 showed how the same part can be machined and measured at multiple sites. For this round of testing support for measurement machines was added to the DLL by adding commands to read tolerance data from AP-203 edition 2 and plan probing operations. An export interface was then added to the DLL to generate DMIS measurement codes for off machine probing, and Renishaw probing codes for on-machine probing.

The fourth round of testing in 2008 will demonstrate simulation and verification using the DLL. In this round of tests the plan is to show: just in time generation of optimal speeds and feeds; just in time generation of five axis cutter compensation codes from part geometry; and the run time tracing of machining information in a form that can be understood at a later time by the customer of a machine shop.

Properties of the DLL

The success of the DLL can be attributed to four properties enabled by the three schema architecture. The four properties are Automation, Consistency, Isolation and Documentation. Properties with similar names were enabled for database systems by its three schema architecture [4].

Table 2 summarizes the four properties. The automation property means the automation of the mappings from the ARM to the AIM using a code generator. The mappings were a frequent source of errors in traditional STEP implementation. In fact, so frequent that they often took away all the time of the programmer so he or she did not have any for the semantically far more important ARM level mistakes. Using automation to generate the mappings has considerably reduced this number of errors.

Table 2. The Four Properties enabled by the Three Schemas

Property	Advantage	Implementation
Automation	Greater accuracy and far less effort	A mapping table parser is used to convert the STEP mapping tables to C++ code. A STEP expert then annotates with the necessary rules for data sharing
Consistency	Easier identification of the operation that was the original source of an error	Require that every operation takes the database from one valid state (as defined by the EXPRESS model) to another valid state.
Isolation	Easier identification of the function calls necessary to create a data item.	Require each operation to complete its data in a single function call so that the programmer does not have to navigate through long data sequences to find the necessary data handles.
Documentation	Easier understanding of the data in a file and easier identification of missing or invalid data.	Add comments to the STEP file to show how each property of each ARM object is represented in that file.

The consistency property is enabled by requiring the operations that modify data to take the database from one valid state to another valid state. Within the DLL consistency is enabled by waiting for a complete set of information to be available before adding data to the database. For example, a change to the coolant value does not immediately result in a new coolant object being added to the database. Instead the DLL waits until a new tool path that uses that coolant value is ready for insertion and then adds the coolant value to the database. Consistency is desirable because in the event of an error the problem can be traced back to the first operation that put the database into an inconsistent state.

The isolation property is enabled by requiring each operation to perform its function using a single function call. Many object oriented systems have operations that require their functions to be given a handle to the results of other functions. The STEP information models are so complex that this requirement becomes a major source of frustration to programmers. For example, consider a drilling operation. One of the factors that must be tested in this operation is status of the bottom of the hole. If the sequence starts with a handle to the working step, then the STEP-NC DLL will find out the required status using a single function call to the method GetHoleBottomType of the Finder interface, but a handle based interface such as the ones provided by the EXPRESS tool kits will require:

1. Six function calls to reach the handle of the round hole.
2. Five function calls to reach the handle of the hole bottom.
3. One function call to decode the type of the hole bottom.

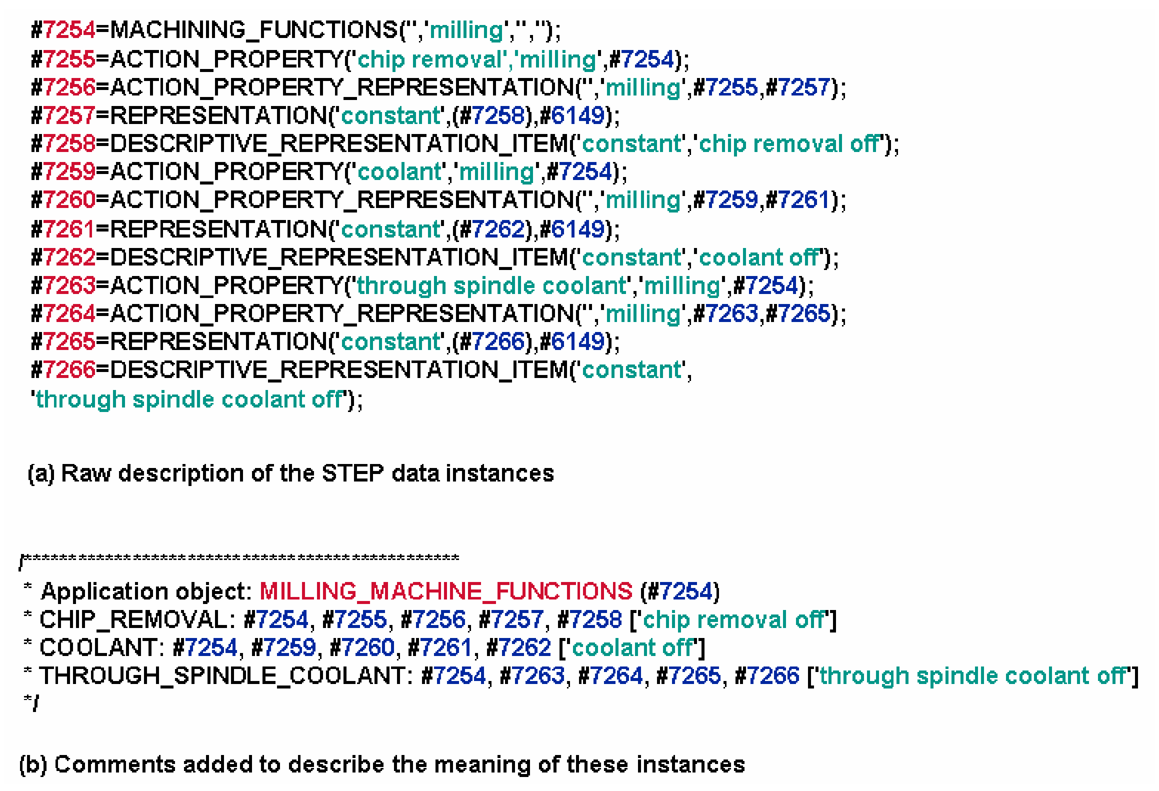
Obviously, each function must be given the right arguments. Therefore, in writing this code there is much searching through the standards to check the requirements and the net effect is like trying to solve a crossword puzzle with limited clues.

The requirement for isolation means that some of the functions in the DLL are relatively wide, so it can be argued that the DLL replaces excessive depth with excessive width. However, two factors mitigate this problem. The first is the IntelliSense gadget of the Microsoft Integrated Development Environment (IDE) and other environments. This gadget shows the programmer how to complete a function call. Therefore, the necessary documentation is provided automatically. The other mitigating factor is the familiarity of the programmer with the application domain. For example, the parameters required to make a round hole are well known so even if there are a lot of arguments, they can be deciphered by giving them intelligent names such as “diameter” and “depth”. It is very difficult to give a sequence of twelve depth oriented function calls meaningful names.

The last property is Documentation. This property is inspired by XML which showed the value of self documenting data. Documentation is very important because efficient debugging requires the programmer to rapidly determine what objects are in the data and whether or not those objects have all the required properties. The STEP-NC DLL implements documentation by adding comments to the Part 21 (ASCII) and Part 28 (XML) formats. Figure 6 illustrates. Figure 6a shows some raw uncommented STEP data.

In this data the "#" symbol is used to denote both an entity identifier and an entity reference. Clearly there are few clues as to what this data represents though it can be deciphered with effort.

Figure 6b shows the comments added to the data by the DLL to show that this data cluster represents a machine_functions object for milling processes and that this object has the properties shown. The new data can be recognized at a glance. In the header the sequence of numbers shows the sequence of entities used in each mapping. As well as making the correctness of the mapping easier to check, the sequence tells the system how to cluster the entities in the representation of the object. In this respect the illustration given in Figure 6a is misleading because all of the entities in the object have been clustered together. In a "raw" Part 21 or Part 28 file without comments there is no reason to cluster them in this way and there could be 100,000 or more entity instances between two related data items in a file.



```
#7254=MACHINING_FUNCTIONS('','milling','','');
#7255=ACTION_PROPERTY('chip removal','milling',#7254);
#7256=ACTION_PROPERTY_REPRESENTATION('','milling',#7255,#7257);
#7257=REPRESENTATION('constant',(#7258),#6149);
#7258=DESCRIPTIVE_REPRESENTATION_ITEM('constant','chip removal off');
#7259=ACTION_PROPERTY('coolant','milling',#7254);
#7260=ACTION_PROPERTY_REPRESENTATION('','milling',#7259,#7261);
#7261=REPRESENTATION('constant',(#7262),#6149);
#7262=DESCRIPTIVE_REPRESENTATION_ITEM('constant','coolant off');
#7263=ACTION_PROPERTY('through spindle coolant','milling',#7254);
#7264=ACTION_PROPERTY_REPRESENTATION('','milling',#7263,#7265);
#7265=REPRESENTATION('constant',(#7266),#6149);
#7266=DESCRIPTIVE_REPRESENTATION_ITEM('constant',
'through spindle coolant off');
```

(a) Raw description of the STEP data instances

```
/*************************************************
 * Application object: MILLING_MACHINE_FUNCTIONS (#7254)
 * CHIP_REMOVAL: #7254, #7255, #7256, #7257, #7258 ['chip removal off']
 * COOLANT: #7254, #7259, #7260, #7261, #7262 ['coolant off']
 * THROUGH_SPINDLE_COOLANT: #7254, #7263, #7264, #7265, #7266 ['through spindle coolant off']
 */
```

(b) Comments added to describe the meaning of these instances

Figure 6. Better Documentation.

Conclusion

Collaboration is extremely important to industry because it leads to economies of scale. More efficient data exchange means more efficient collaboration. The goal of the STEP-NC DLL is to enable collaboration for technical product data by integrating the information required for CAD, CAM, CNC and CMM applications.

The information of these domains is already available in many different files with different formats. The STEP-NC DLL makes it possible for an application to put these files together. Shop floor manufacturing applications can then deliver new functionality to end users. Examples include just in time simulation and verification, integrated machining and measurement, five axis cutter compensation and manufacturing traceability.

The STEP-NC DLL enables these applications because it allows the geometry and tolerances of a product to be aligned with the operations and tool paths of a process to create an integrated data set. Previously these data sets belonged to different systems. The geometry and tolerance data was created by a CAD system, and the process and tool path data was created by a CAM system. An application that wanted to use both had to resolve the differences between the coordinates and units of the two systems and then infer all of the connections. For example, it had to align the geometry of the part, which is usually in the coordinate space of the product, against the geometry of the tool paths which is usually in the coordinate space of the machine, and then infer which face of the part was being machined by each tool path in order to check its validity.

With the STEP-NC DLL, an application can be written to integrate and connect the data at the data source, so that intelligent shop floor applications can deliver new value to the operators. For example:

- Connecting the tool path data to the geometry data means an intelligent CNC application can detect gouges to the part geometry.
- Connecting the process to the geometry means an application can optimize the speeds and feeds to minimize machining time or maximize cutting tool life.
- Connecting the machining set-up to the part geometry means an application can implement five-axis cutter compensation for tool wear.
- Connecting the tolerances to the process means an on-machine inspection application can check for errors and discrepancies at the end of each operation instead of waiting for the part to be transferred to an inspection machine at the end of the machining.
- Connecting the CAD part to the CAM process means a traceability application can report back on the quality of the machining process to the product designer.

The STEP-NC DLL uses the STEP standards to define its data. Since STEP was started in 1984, the STEP committees have achieved several notable successes. For example, they have a procedure in place that allows global consensus to be achieved on the required information content, they have a methodology that allows all the data requirements for a new standard to be identified, they have an architecture that allows the whole life cycle of a product to be modeled, and they have developed a standard for geometry that is supported by nearly every CAD and CAM vendor.

However in recent years these successes have been marred by the slow rate of implementation for the attribute rich manufacturing protocols. As the STEP Application Protocols have grown bigger, the rate of implementation has grown slower because of the complexity of writing code for hundreds of objects that each have to be mapped into thousands of connected entities. The first protocols only had 41 object types. Some of the recent application protocols have as many as 700 object types.

The STEP-NC DLL uses a three schema architecture to manage complexity using abstraction. The DLL has been used to read and write CAD, CAM, CNC and CMM data defined by a wide range of commercial off the shelf systems. For the first time one library has been able to process data from multiple manufacturing application domains.

We attribute the success of the DLL to the four properties enabled by the architecture: Automation, Consistency, Isolation and Documentation. The automation property allowed us to rapidly generate code for the DLL that is safe and easy to use. The consistency property allowed us to quickly identify the operation that was the original source of an error. The Isolation property allowed us to complete an operation in one function call instead of multiple inter-related nested function calls. The Documentation property allowed us to rapidly debug errors in the data.

The STEP-NC DLL has been tested in multiple scenarios over the last three years. Starting in 2001 with interoperability testing of tool paths, the scenarios have shown how the DLL can be used to implement the integrated simulation, machining, measurement and tracing of manufacturing operations. The next stage is for the key vendors of the manufacturing community to start using the DLL to connect their applications to the data defined by the standards.

REFERENCES

[1] Suh, S.H., Cho, J.H., and Hong, H.D., 2001, "On the architecture of intelligent STEP-compliant CNC," *Int'l J. Computer Integrated Manufacturin*g, Vol. 15, No. 2, January 2002, pp. 168-177.

[2] ISO 10303-1:1994 Industrial automation systems and integration Product data representation and exchange - Overview and Fundamental Principles, International Standard, ISO TC184/SC4, 1994.

[3] ISO 14649-1:2001 Industrial automation systems and integration Physical Device Control-Part 1: Overview and Fundamental Principles, Draft International Standard ISO TC184/SC4, 2001.

[4] P.A. Bernstein, V. Hadzilacos, and N. Goodman. "Concurrency Control and Recovery in Database Systems ". Addison-Wesley, 1987.

[5] M. Hardwick, D.L.Spooner, T. Rando, K.C. Morris, "Sharing manufacturing information in virtual enterprises", Communications of the ACM, Volume 39, No. 2, 1996.

APPENDIX A THE STEP APPLICATION PROTOCOLS

Each Application Protocol defines a data exchange standard. Only some of the protocols have an impact on end-users. Others establish a capability that will be harvested by a subsequent standard.

AP 201	Explicit Drafting
AP 202	Associative Drafting
AP 203	Configuration Controlled Design
AP 204	Mechanical Design Using Boundary Representation
AP 205	Mechanical Design Using Surface Representation
AP 206	Mechanical Design Using Wireframe Representation
AP 207	Sheet Metal Dies and Blocks
AP 208	Life Cycle Product Change Process
AP 209	Design Through Analysis of Composite and Metallic Structures
AP 210	Electronic Printed Circuit Assembly, Design and Manufacturing
AP 211	Electronics Test Diagnostics and Remanufacture
AP 212	Electrotechnical Plants
AP 213	Numerical Control Process Plans for Machined Parts

AP 214	Core Data for Automotive Mechanical Design Processes
AP 215	Ship Arrangement
AP 216	Ship Molded Forms
AP 217	Ship Piping
AP 218	Ship Structures
AP 219	Dimensional Inspection Process Planning for CMMs
AP 220	Printed Circuit Assembly Manufacturing Planning
AP 221	Functional Data and Schematic Representation for Process Plans
AP 222	Design Engineering to Manufacturing for Composite Structures
AP 223	Exchange of Design and Manufacturing DPD for Castings
AP 224	Mechanical Product Definition for Process Planning
AP 225	Structural Building Elements Using Explicit Shape Rep
AP 226	Shipbuilding Mechanical Systems
AP 227	Plant Spatial Configuration
AP 228	Building Services
AP 229	Design and Manufacturing Information for Forged Parts
AP 230	Building Structure frame steelwork
AP 231	Process Engineering Data
AP 232	Technical Data Packaging
AP 233	Systems Engineering Data Representation
AP 234	Ship Operational logs, records and messages
AP 235	Materials Information for products
AP 236	Furniture product and project
AP 237	Computational Fluid Dynamics
AP 238	Integrated CNC Machining
AP 239	Product Life Cycle Support
AP-240	Process Planning

APPENDIX B Functions in the Feature Interface

Functions in the Feature interface of the DLL as of June 2006.

```
void OpenWorkpiece(LPCTSTR file_name, LPCTSTR piece_name, long force_p21);
void Inches();
void Millimeters();
void SetDirection(double i, double j, double k, double a, double b, double c);
void SetLocation(double x, double y, double z);
void AddFace(long ws_id, long face_id);
void RemoveFace(long ws_id, long face_id);
void LineTo(LPCTSTR label, double x, double y, double z);
void Arc(LPCTSTR label, double x, double y, double z, double cx, double cy, double cz, double radius, long ccw);
long ClosedRectangularPocket(long ws_id, LPCTSTR name, double depth, double length, double width, double
orthogonal_radius);
long ClosedCircularPocket(long ws_id, LPCTSTR name, double depth, double diameter);
long ClosedGeneralPocket(long ws_id, LPCTSTR name, double depth);
long OpenPartialCircularPocket(long ws_id, LPCTSTR name, double depth, double radius, double sweep_angle);
long OpenGeneralPocket(long ws_id, LPCTSTR name, double depth);
long RoundHole(long ws_id, LPCTSTR name, double depth, double diameter);
long HoleFlatBottom(long feature_id);
long HoleConicalBottom(long feature_id, double tip_angle, double tip_radius);
long HoleFlatWithRadiusBotttom(long feature_id, double corner_radius);
long HoleSphericalBottom(long feature_id, double radius);
long ClosedRectangularOutsideProfile(long ws_id, LPCTSTR name, double depth, double length, double width);
long ClosedCircularOutsideProfile(long ws_id, LPCTSTR name, double depth, double diameter);
long PartialCircularOutsideProfile(long ws_id, LPCTSTR name, double depth, double radius, double sweep_angle);
long LinearOutsideProfile(long ws_id, LPCTSTR name, double depth, double length);
long ClosedGeneralOutsideProfile(long ws_id, LPCTSTR name, double depth);
long OpenGeneralOutsideProfile(long ws_id, LPCTSTR name, double depth);
long PocketFlatBottom(long feature_id, double planar_radius);
long PocketRadiusedBottom(long feature_id, double radius, double x, double y, double z);
void QualifyDepth(long pocket_id, double lower, double upper);
long RectangularBoss(long feature_id, LPCTSTR name, double height, double length, double width);
long CircularBoss(long feature_id, LPCTSTR name, double height, double diameter);
long GeneralBoss(long feature_id, LPCTSTR name, double height);
long PlanarFace(long ws_id, LPCTSTR name, double depth, double length, double width);
void Reset();
void OpenNewWorkpiece(LPCTSTR file_name);
void Shutdown();
long CompoundFeature(long ws_id, LPCTSTR name);
void CompoundAddFeature(long compound_id, long feature_id);
void WorkingstepAddFinalFeature(long ws_id, long feature_id);
```