Enabling the Crowd Sourcing of Very Large Product Models

Martin Hardwick, David Loffredo, Joe Fritz, Mikael Hedlind

Enabling the Crowd Sourcing of Very Large Product Models

Martin Hardwick[1], David Loffredo[2], Joe Fritz[2], Mikael Hedlind[3, 4]

[1]Rensselaer Polytechnic Institute, Troy, NY 12180, USA
hardwick@steptools.com

[2]STEP Tools, Inc., Troy, NY 12180, USA
[3]KTH Royal Institute of Technology., SE-100 44 Stockholm, Sweden
[4]Production Engineering R&D., Scania CV AB, 151 87 Södertälje, Sweden

Abstract. Part 21 is a public domain file format for the geometry of assemblies and is widely used by industry to describe design and manufacturing specifications for airplanes, automobiles, ships, buildings and other products. We describe a proposed new edition of Part 21 that includes URI's and JavaScript to enable the crowd sourcing of very large product models.

Keywords: Data exchange, Product Models, CAD, CAM, STEP, STEP-NC,

1 INTRODUCTION

Part 21 is a specification for how to format entities describing product data [1]. The format is minimal to maximize upward compatibility and simple to allow for quick implementation. It was invented before XML, though not before SGML, and it makes no special allowance for URL's [2].

Several technical data exchange standards use Part 21. They include STEP for mechanical products, STEP-NC for manufacturing products and IFC for building products. Over twenty years, substantial installed bases have been developed for all three with many Computer Aided Design (CAD) systems reading and writing STEP, a growing number of Computer Aided Manufacturing (CAM) systems reading and writing STEP-NC, and many Building Information Management (BIM) systems reading and writing IFC.

The data described by STEP, STEP-NC and IFC is continually growing [3]. STEP was first standardized as ISO 10303-203 for configuration controlled assemblies, and as ISO 10303-214 for automotive design. Both protocols describe the same kinds of information, and have taken turns at the cutting edge. Currently they are being replaced by ISO 10303-242 which will add manufacturing requirements, such as tolerances and surface finishes, to the product data [4].

STEP-NC is a related standard for manufacturing process and resource data. It has been tested by an industry consortium to verify that it has all the features necessary to replace traditional machining programs. They recently determined that it is ready for

implementation and new interfaces are being developed by the Computer Aided Manufacturing (CAM) system vendors [5].

IFC describes a similar set of standards for building design and construction. IFC has made four major releases with the most recent focused on enabling the concurrent modeling of building systems. These include the systems for electric power, plumbing, and Heating, Ventilation and Air Conditioning (HVAC). The building structural elements such as floors, rooms and walls were already covered by previous editions. With the new release, different contractors will be able to share a common model during the construction and maintenance phases of a building [6].

All three models are being used by a large community to share product data but Part 21 has been showing its age for several years. In the last ten years there have been six attempts to replace it with XML [7]. To date, none has succeeded but there is a growing desire for a more powerful and flexible product data format.

This paper describes an extension to Part 21 to enable the crowd sourcing of very large product models. Extending the current format has the advantage of continuing to support the legacy which means there will be a large range of systems that can already read and write the new data. The new edition has two key new capabilities:

1. The ability to distribute data between model fragments linked together using URI's.
2. The ability to define intelligent interfaces that assist the user in linking, viewing and running the models.

The next section describes the functionalities and limitations of Part 21 Editions 1 and 2. The third section describes how the new format enables massive product databases. The fourth section describes how the new format enables crowdsourcing. The fifth section outlines the applications being used to test the specification. The last section contains some concluding remarks.

2 EDITIONS 1 AND 2 OF PART 21

STEP, STEP-NC and IFC describe product data using information models. Each information model has a schema described in a language called EXPRESS that is also one of the STEP standards [3]. EXPRESS was defined by engineers for engineers. Its main goal was to give clear, concise definitions to product geometry and topology. An EXPRESS schema defines a set of entities. Each entity describes something that can be exchanged between two systems. The entity may describe something simple such as a Cartesian point or something sophisticated such as a boundary representation. In the latter case the new entity will be defined from many other entities and the allowed data structures. The allowed data structures are lists, sets, bags and arrays. Each attribute in an entity is described by one of these data structures, another entity or a selection of the above. The entities can inherit from each other in fairly advanced ways including AND/OR combinations. Finally EXPRESS has rules to define constraints: a simple example being a requirement for a circle radius to be positive; a

more complex example being a requirement for the topology of a boundary representation to be manifold.

Part 21 describes how the values of EXPRESS entities are written into files. A traditional Part 21 file consists of a header section and a data section. Each file starts with the ISO part number (ISO-10303-21) and begins the header section with the HEADER keyword. The header contains at least three pieces of information a FILE_DESCRIPTION which defines the conformance level of the file, a FILE_NAME and a FILE_SCHEMA. The FILE_NAME includes fields that can be used to describe the name of the file, a time_stamp showing the time when it was written, the name and organization of the author of the file. The FILE_NAME can also include the name of the preprocessing system that was used to write the file, and the name of the CAD system that created the file.

```
ISO-10303-21;
HEADER;
/* Exchange file generated using ST-DEVELOPER v1.5 */

FILE_DESCRIPTION(
/* description */ (''),
/* implementation_level */ '2;
1');
FILE_NAME(
/* name */ 'bracket1',
/* time_stamp */ '1998-03-10T10:47:06-06:00',
/* author */ (''),
/* organization */ (''),
/* preprocessor_version */ 'ST-DEVELOPER v1.5',
/* originating_system */ 'EDS - UNIGRAPHICS 13.0',
/* authorisation */ '');

FILE_SCHEMA (('CONFIG_CONTROL_DESIGN')); /* AP203 */
ENDSEC;
DATA;
#10 = ORIENTED_EDGE('',*,*,#44820,.T.);
#20 = EDGE_LOOP('',(#10));
#30 = FACE_BOUND('',#20,.T.);
#40 = ORIENTED_EDGE('',*,*,#44880,.F.);
#50 = EDGE_LOOP('',(#40));
#60 = FACE_BOUND('',#50,.T.);
#70 = CARTESIAN_POINT('',(-
1.31249999999997,14.594,7.584));
#80 = DIRECTION('',(1.,0.,3.51436002694883E-15));
…
ENDSEC;
END-ISO-10303-21;
```

One or more data sections follow the header section. In the first edition only one was allowed and this remains the case for most files. The data section begins with the keyword DATA, followed by descriptions of the data instances in the file. Each instance begins with an identifier and terminates with a semicolon ";". The identifier is a hash symbol "#" followed by an unsigned integer. Every instance must have an identifier that is unique within this file but the same identifier can be given to another instance in another file. This includes another version of the same file.

The identifier is followed by the name of the entity that defines the instance. The names are always capitalized because EXPRESS is case insensitive. The name of the instance is then followed by the values of the attributes listed between parentheses and separated by commas. Let's look at instance #30. This instance is defined by an entity called FACE_BOUND. The entity has three attributes. The first attribute is an empty string, the second is a reference to an EDGE_LOOP and the third is a Boolean with the value True. The EXPRESS definition of FACE_BOUND is shown below. FACE-BOUND is an indirect subtype of representation item. The first attribute of FACE-BOUND (the string) is defined by this super-type. Note also that the "bound" attribute of face_bound is defined to be a loop entity so EDGE_LOOP must be a subtype of LOOP.

```
ENTITY face_bound
  SUBTYPE OF (topological_representation_item);
  bound       : loop;
  orientation : BOOLEAN;
END_ENTITY; -- face_bound

ENTITY topological_representation_item
SUPERTYPE OF (ONEOF (vertex, edge, face_bound, face, con-
nected_edge_set, connected_face_set, vertex_shell,
wire_shell, (loop ANDOR path)))
  SUBTYPE OF (representation_item);
END_ENTITY; -- topological_representation_item

ENTITY representation_item;
  name : label;
WHERE
  WR1: SIZEOF (using_representations(SELF)) > 0;
END_ENTITY; -- representation_item
```

There were two major design goals for Part 21 Edition 1.

1. Make it easy for CAD software engineers to create readers and writers for the new format.
2. Maximize the upward compatibility by minimizing the number of keywords and structural elements in the data exchange files.

The first goal was met by requiring the files to be encoded in simple ASCII, by requiring all of the data to be in one file, and by the requiring every identifier to be an unsigned integer that only has to be unique within the context of one file. The latter condition was presumed to make it easier for engineers to write parsers for the data.

The second design goal was met by minimizing the number of keywords and structural elements (nested parentheses). In most cases, the only keyword is the name of the entity and unless there are multiple choices there are no other keywords listed. The multiple choices are rare. They happen if an attribute can be defined by multiple instances of the same type. An example would be an attribute which could be a length or a time. If both possibilities are represented as floating point numbers then a keyword is necessary to indicate which has been chosen.

Making the Part 21 format simple was helpful in the early years as some users developed EXPRESS models and hand populated those files. However as previously mentioned there are thousands of definitions in STEP, STEP-NC and IFC and to make matters worse the definitions are normalized to avoid insertion and deletion anomalies. Consequently, it quickly became too difficult for engineers to parse the data by hand and a small industry grew up to manage it using class libraries. This industry then assisted the CAD, CAM and BIM vendors as they implemented their data translators.

The two design goals conflict when minimizing the number of keywords makes the files harder to read. This was dramatically illustrated when XML became popular. The tags in XML have allowed users to create many examples of relatively easy to understand, self-describing web data. However, for product models contain thousands of definitions the tags are less helpful. The following example recodes the first line of the data section of the previous example in XML.

```
<data-instance ID="i10">
   <type IDREF="oriented_edge">
    <attribute name="name"
        type="label"></attribute>
    <attribute name="edge_start"
        type="vertex" value="derived"/>
    <attribute name="edge_end"
        type="vertex" value="derived"/>
    <attribute name="edge_element"
        type="edge"><instance-ref
            IDREF="i44820"/></attribute>
    <attribute name="orientation"
        type="BOOLEAN">TRUE</attribute>
   </type>
</data-instance>
```

Six XML data formats have been defined for STEP in two editions of a standard known as Part 28 [6]. The example above shows the most verbose format called the Late Binding which tried to enable intelligent data mining applications. Other formats were more minimal though none as minimal as Part 21. In practice the XML tags add

small value to large product models because anyone that wants to parse the data needs to process the EXPRESS. Plus they cause occasional update problems because in XML adding new choices means adding new tags and this can mean old data (without the tags) is no longer valid.

The relative failure of Part 28 has been mirrored by difficulties with Part 21 Edition 2. This edition sought to make it easier for multiple standards to share data models. At the time STEP was moving to an architecture where it would be supporting tens or hundreds of data exchange protocols each tailored to a specific purpose and each re-using a common set of definitions. Edition 2 made it possible to validate a STEP file in all of its different contexts by dividing the data into multiple sections each described by a different schema. In practice, however, the applications have folded down to just three that are becoming highly successful: STEP for design data, STEP-NC for manufacturing data and IFC for building and construction date.

The failures of XML and Edition 2 need to be balanced against the success of Edition 1. This edition is now supported by nearly every CAD, CAM and BIM system. Millions of product models are being made by thousands of users. Consequently there is an appetite for more, and users would like to be able to create massive product models using crowd sourcing.

3 MASSIVE PRODUCT MODELS

The following subsections describe how Edition 3 enables very large product models. The first two subsections describe how model fragments can be linked by URI's in anchor and reference sections. The third subsection describes how the transport of collections of models is enabled using ZIP archives. The fourth subsection describes how the population of a model is managed using a schema population.

3.1 Anchor section

The syntax of the new anchor section is simple. It begins with the keyword ANCHOR and ends with the keyword ENDSEC. Each line of the anchor section gives an external name for one of the entities in the model. The external name is a reference that can be found using the fragment identifier of a URL. For example, the URL www.server.com/assembly.stp#front_axle_nauo references the front_axle in the following anchor section.

```
ANCHOR;
<front_axle_nauo> = #123;
<rear_axle_nauo> = #124;
<left_wheel_nauo> = #234;
<right_wheel_nauo> = #235;
ENDSEC;
```

Unlike the entity instance identifiers of Edition 1, anchor names are required to be unique and consistent across multiple versions of the exchange file. Therefore, alt-

hough the description of the front_axle in chasis.stp may change, the front_axle anchor remains constant.

3.2 Reference section

The reference section follows the anchor section and enables references into another file. Together the reference and anchor sections allow very large files to be split into fragments.

The reference section begins with the keyword REFERENCE and ends with the keyword ENDSEC. Each line of the reference section gives a URI for an entity instance defined in an external file. In this example, the external file contains references to the anchors given in the previous example. The file is defining a manufacturing constraint on an assembly [7].

```
REFERENCE;
/* assembly definitions for this constraint */
#outer_seal_nauo = <assembly.stp#outer_seal>;
#outer_bearing_nauo =
    <assembly.stp#outer_bearing>;
#right_wheel_nauo = <assembly.stp#right_wheel>;
#rear_axle_nauo = <assembly.stp#rear_axle>;

/* Product definitions */
#seal_pd = <assembly.stp#seal_pd>;
#bearing_pd = <assembly.stp#bearing_pd>;
#wheel_pd = <assembly.stp#wheel_pd>;
#axle_pd = <assembly.stp#axle_pd>;
#chasis_pd = <assembly.stp#chasis_pd>;
ENDSEC;
```

The example uses names for the entity identifiers. This is another new feature of Edition 3. Instead of requiring all entity identifiers to be numbers they can be given names to make it easier for casual users to code examples, and for systems to merge data sets from multiple sources.

```
DATA;
#124 = SPECIFIED_HIGHER_COMPONENT_USAGE ('id',
       'name', 'outer seal on outer bearing',
  #chasis_pd,#seal_pd,'outer', #125,
       #outer_seal_nauo);
#125 = SPECIFIED_HIGHER_COMPONENT_USAGE ('id',
       'name', 'right wheel on rear axle ',
       #chasis_pd,#bearing_pd,,'rear', #126,
       #outer_bearing_nauo);
#126 = SPECIFIED_HIGHER_COMPONENT_USAGE ('id',
       'name', 'right wheel on rear axle ',
```

```
        #chasis_pd,#wheel_pd,'outer',
        #rear_axle_nauo,#right_wheel_nauo);
ENDSEC;
```

Numbers are used for the entities #124, #125 and #126 because it is traditional but the rest of the content has adopted the convention of giving each instance a name to indicate its function and a qualifier to indicate its type. Thus “chasis_pd” indicates that this instance is the product_definition entity of the chasis.stp file.

3.3 ZIP Archives and Master Directories

The anchor and reference sections allow a single earlier-edition file to be split into multiple new files but this can result in management problems. The old style led to files that were large and difficult to edit outside of a CAD system, but all of the data was in one file which was easier to manage.

ZIP archives allow Part 21 Edition 3 to split the data and continue the easy data management. A ZIP archive is a collection of files that can be e-mailed as a single attachment. The contents of the archive can be any collection including another archive. A ZIP archive is compressed and may reduce the volume by as much as 70%. Many popular file formats such as “.docx” are ZIP files and can be accessed using ZIP tools (sometimes only after changing the file extension to .zip)

Edition 3 allows any number of STEP files to be included in an archive. Each file in the directory can be linked to the other files in the ZIP using relative addresses and to other files outside of the ZIP using absolute addressing. Relative addresses to files outside the ZIP are not allowed so that applications can deploy the zipped data at any location in a file system.

References into the ZIP file are allowed but only via a master directory stored in the root. This Master directory describes where all the anchors in the ZIP can be found using the local name of the file. Outside the archive the only visible name is that of the archive itself. If there is a reference to this name then a system is required to open the master directory and look for the requested anchor.

In Figure 1 the file ISO-10303-21.txt is the master directory. It contains the forwarding references to the other files and it is the file that can be referenced from outside of the archive using the name of the archive.

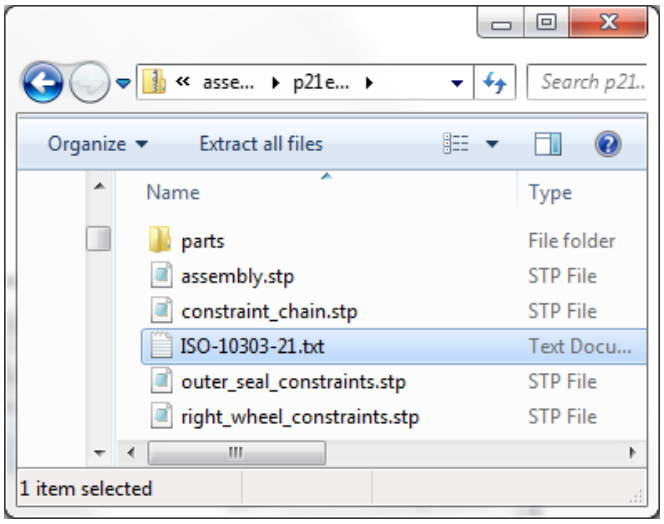

Fig. 1. - Zip Archive

3.4 Schema population with time stamps

The support for data distribution in Part 21 Edition 3 gives rise to a problem for applications that want to search a complete data set. If the data is distributed and normalized then there may be "orphan" files that contain outbound references but no inbound ones. For example, Figure 2 shows how a file may establish a relationship between a workplan and a workpiece by storing URI's to those items but not have a quality that needs to be referenced from anywhere else.

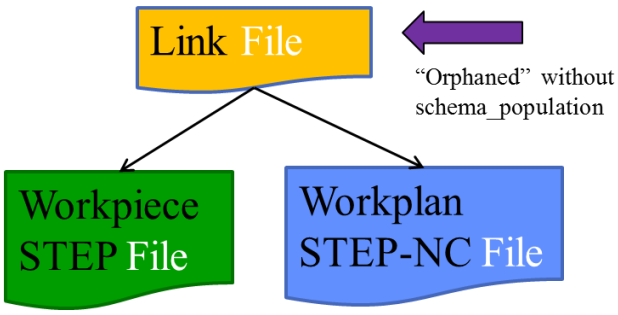

Fig. 2. - Link files for massive databases

Part 21 Edition 3 solves this problem using a Schema Population which is defined as follows:

- The schema population includes all the entity instances in all the data sections of the file.

- If there is a reference section, then the schema population also includes the schema populations of all the files referenced by URIs.
- If the header has a schema_population definition then the schema population also includes the schema population of each file in the set of external_file_locations.

The last inclusion catches the "orphan files". The following code fragment gives an example. In this example the STEP file shown is referencing two other files. There are two other attributes in each reference. An optional time stamp shows when the reference was last checked. An optional digital signature validates the integrity of the referenced file.

```
SCHEMA_POPULATION(
('http://www.acme.net/first_file.stp',
 '2013-02-11T17:35:00', /* padlock below */      /* Time
stamp */
 '30dd879c-ee2f-11db-8314-0800200c9a66');
('http://www.giant.com/second_file.stp',
 '2013-02-11T17:30:00', /* time stamp left */
 ''));  /* no padlock for this instance*/         /* not
locked */
```

The time stamp and signature enable better data management in contractual situations. Clearly if the data is distributed there will be opportunities for mistakes and mischief.

4 INTELLIGENT INTERFACES

The following subsections describe how Edition 3 enables crowdsourcing using intelligent interfaces. The first subsection describes how JavaScript has been added to the model. The second subsection describes how some of the data rules have been relaxed for easier programming. The third subsection describes how application specific programming is enabled using data tags. The last subsection summarizes the options for making the programming more modular.

4.1 JavaScript

The goal of adding JavaScript to Part 21 Edition 3 is to make managing the new interfaces easier by encapsulating the tasks that can be performed on those interfaces as methods. For example, the following code checks that a workpiece is valid before linking it to a workplan.

```
function link(workplan, workpiece, name)
{
    with workpiece.anchor.credentials.tag {
```

```
        if (stage != 'manufacturing')
    return null;
        if (status != 'released')
           return null;
    }
    if (name != 'as_is shape'
  && name != 'to-be shape')
return NULL;
    if (workpiece.anchor.shape.tag.type
  != 'product_definition_shape')    // alternate is to
check using an EXPRESS compiler
        return NULL:
    if (workpiece.anchor.shape.tag.geometry
  != 'brep')
        return NULL;
    name = type;
    shape = workpiece.anchor.shape;
    exec = workplan.anchor.executable;
    return model;
}
```

In Edition 3 a file can include a library of JavaScript functions to operate on an object model of the anchors and references but not necessarily the data sections. In many cases the volume of the data sections will overwhelm a JavaScript interpreter.

The following three step procedure is used for the conversion:

1. Read the exchange structure and create a P21.Model object with anchor and reference properties.
2. Read the JavaScript program definitions listed in the header section.
3. Execute the JavaScript programs with the "this" variable set to the P21.Model object.

For more details see Annex F of the draft specification at www.steptools.com/library/standard/. The procedure creates one object for each exchange file and gives it the behaviour defined in the JavaScript. The execution environment then uses those objects in its application. For example, in the code above workplan and workpiece are object models for two exchange structures and the application is checking for compatibility before linking them.

4.2 Data relaxation

In order to be reusable a product model needs to flexible and extensible. The information models defined by the STEP, STEP-NC and IFC standards have been carefully designed over many releases to achieve these qualities.

Interface programming is different because an interface can be created as a contingent arrangement of anchors and references for a specific purpose. The information

model has not changed so application programming for translation systems stays the same, but for interface programming the requirements are different. Therefore, two relaxations have been applied to the way data is defined for interfaces in the new edition.

1. The instance identifiers are allowed to be alphanumeric.
2. The values identified can be lists and literals.

Editions 1 and 2 of Part 21 restricted the format so that every identifier had to be an unsigned integer. This helped emphasize that the identifiers would not be consistent across files and at the time it was thought to make it easier for parsers to construct symbol tables. The symbol table reason is false. Every modern system uses a hash table for its symbols and these tables are agnostic with respect to the format of the identifier.

Requiring numbers for identifiers has always made hand editing harder than necessary. Therefore, Edition 3 supports alphanumeric names. The following example of unit definition illustrates the advantage. Each unit definition follows a pattern.

```
ANCHOR;
<NEWTON> = #newton;
<PASCAL> = #pascal;
ENDSEC;

REFERENCE;
#metre =
<http://www.iso10303.org/part41/si_base_units.stp#METRE>;
#kilogram =
<http://www.iso10303.org/part41/si_base_units.stp#KILOGRA
M>;
#second =
<http://www.iso10303.org/part41/si_base_units.stp#SECOND>
;
ENDSEC;

DATA;
/* Content extracted from part 41:2013 */
#5_newton=DERIVED_UNIT_ELEMENT(#meter,1.0);
#15_newton=DERIVED_UNIT_ELEMENT(#kilogram,1.0);
#25_newton=DERIVED_UNIT_ELEMENT(#second,-2.0);
#new-
ton=SI_FORCE_UNIT((#5_newton,#15_newton,#25_newton),*,$,.
NEWTON.);

#5_pascal=DERIVED_UNIT_ELEMENT(#meter,-2.0);
#25_pascal=DERIVED_UNIT_ELEMENT(#newton,1.0);
```

```
#pas-
cal=SI_PRESSURE_UNIT((#5_pascal,#25_pascal),*,$,.PASCAL.)
;
```

Unit definitions also show why a more flexible approach to defining literals is desirable. The following is a file that defines some standard constants.

```
ANCHOR;
<ARCHIMEDES_CONSTANT_PI >= #pi;
<EULER_NUMBER_e> = #e;
<GOLDEN_RATIO> = #golden;
ENDSEC;

REFERENCE;
#pi = 3.14159265359;
#e = 2.71828;
#golden = 1.61803398874;
ENDSEC;
END-ISO-10303-21;
```

The new identifiers can be used in the data section as well as the anchor and reference sections.

4.3 Tags for fast data caching

In many cases the JavaScript functions operating on the interfaces need additional data to be fully intelligent. Therefore, the new edition allows additional values to be tagged into the reference and data sections.

Each tag has a category and a value. The category describes its purpose and the value is described by a literal. The following example shows the data checked by the JavaScript function of the previous example.

```
ANCHOR;
<credentials> {stage:'manufacturing'}
              {status:'released'}
    = $;
<shape>       {geometry:'brep'}
   = #217652;
ENDSEC;
```

The tag data may be initialized by a pre-processor or created by other means. In the above example two pieces of data are being linked and it is important to know that the workpiece is ready for use by manufacturing.

Another role for tags is as a place to cache links to visualization information. Again this data may be summarized from the STEP, STEP-NC or IFC information model and the tags allow it to be cached at a convenient place for rapid display.

```
ANCHOR;
<tool_tip> {visualization:<facets.xml>}
           {picture: <item.jpg>}
    = #197;
ENDSEC;
```

The last example shows tags being used to document the STEP ARM to AIM mapping. Those who have worked on STEP and STEP-NC know that they have two definitions: a requirements model describing the information requirements; and an interpreted model that maps the requirements into a set of extensible resources [3]. The tags can become a way to represent the mapping between these two models in the data.

```
REFERENCE;
#1234 {x_axis:(#3091,#3956)}{y_axis: (#2076)} = <#ma-
chine_bed>;
#4567 {z_axis:(#9876,#5273)}= <#tool_holder>;
ENDSEC;
```

A quick summary of the new data options in Edition 3 is that it allows URL's to be placed between angular brackets ("<>") and application specific data to be placed between curly brackets ("{}").

4.4 Modularity options

Part 21 Edition 3 has three options for modularizing STEP, STEP-NC and IFC data.

- Continue using traditional files, but surround those files with interfaces referencing into the data
- Create a product data web linked by URL's.
- Create a ZIP archive of replaceable components.

The traditional approach to STEP implementation creates a massive symbol table using an EXPRESS compiler and then reads the exchange data into objects described by the table. This is expensive both with respect to processing time and software investment, but efficient if all of the data is being translated into a CAD system.

The new edition allows the Part 21 data to be arranged into interfaces for light weight purposes such as checking tolerances, placing subsystems and running processes. Therefore, alternate implementation paradigms are possible. Three being considered include:

1. A JavaScript programming environment can process just the data in an interface. In this type of implementation the Part 21 files are rapidly scanned to create the objects required for the anchor and reference sections.
2. A web browser environment can be activated by including an "index.html" file in the ZIP archive along with code defining the P21 object model of the interface.

This type of implementation will be similar to the previous one but with steaming used to read the Part 21 data.

3. The third type of implementation is an extended Standard Data Access Interface (SDAI). The SDAI is an application programming interface for Edition 1 and 2 data that can be applied to Edition 3 because of upward compatibility.

One option for an SDAI is to merge all the Edition 3 data into one large file for traditional CAD translation processing. Another option is to execute the JavaScript and service web clients

5 APPLICATIONS

5.1 PMI Information for Assemblies

The first in-progress application is the management of Product Manufacturing Information (PMI) for assemblies. Figure 3 shows a flatness tolerance on one of the bolts in an assembly. In the data, a usage chain is defined to show which of the six copies of the bolt has the tolerance.

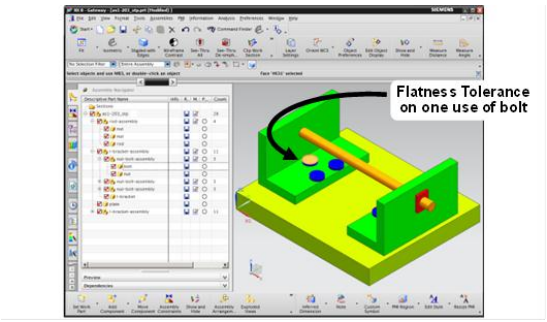

Fig. 3. - Assembly tolerances

The data for the example can be given many organizations. One is the traditional single file which will work well for small data sets. For large assemblies the new specification enables a three layer organization. The lowest layer is the components in the model. The second layer is the assemblies and sub-assemblies. The third layer is the PMI necessary to manufacture the assembly.

In order for this organization to work the product components must expose their product coordinate systems to the assembly modules and their faces to the PMI modules. Similarly the assembly modules must expose their product structure to the PMI modules. The following shows the resulting anchor and reference sections of the PMI module.

```
ANCHOR;
<as1-pe> = #141;
END_SEC;
```

```
REFERENCE;
#bolt = <bolt.stp#bolt>;
#nut = <nut.stp#nut>;
#rod = <rod.stp#rod>;
#plate = <plate.stp#plate>;
#l-bracket = <l-bracket.stp#l-bracket>;

#bolt_shape = <bolt.stp#bolt_shape>;
#nut_shape = <nut.stp#nut_shape>;
#rod_shape = <rod.stp#rod_shape>;
#plate_shape = <plate.stp#plate_shape>;
#l-bracket_shape = <l-bracket.stp#l-bracket_shape>;

#bolt_wcs = <bolt.stp#bolt_wcs>;
#nut_wcs = <nut.stp#nut_wcs>;
#rod_wcs = <rod.stp#rod_wcs>;
#plate_wcs = <plate.stp#plate_wcs>;
#l-bracket_wcs = <l-bracket.stp#l-bracket_wcs>;

#bolt_top_face = <bolt.stp#bolt_top_face>;
ENDSEC;
```

This code is then referenced in the data sections of the PMI modules.

5.2 Data Model Assembly

STEP and STEP-NC are related standards that share definitions. The two models can be exported together by integrated CAD/CAM systems, but if different systems make the objects then they must be linked outside of a CAD system.

In STEP-NC a workplan executable is linked to the shape of the workpiece being machined. The two entities can be exported as anchors in the two files and an intelligent interface can link them on-demand. The following code shows the interface of a linker file with open references to the two items that must be linked. The linker JavaScript program given earlier sets these references after checking the validity of the workpiece and workplan data sets. It also sets the name of the reference to indicate if workpiece represents the state of the part before the operation (as-is) or after the operation (to-be).

```
REFERENCE;
#exec = $;
#shape = $;
#name = $;
ENDSEC;

DATA;
```

```
#10=PRODUCT_DEFINITION_PROCESS(#name,'',#exec,'');
#20=PROCESS_PRODUCT_ASSOCIATION('','',#shape,#10);
ENDSEC;
```

A number of CAM vendors are implementing export interfaces for STEP-NC[5]. They export the process data which needs to be integrated with workpiece data to achieve interoperability. The workpieces define the cutting tools, fixtures and machines as well as the as-is and to-be removal volumes.

5.3 Next Generation Manufacturing

The last application is the control of manufacturing operations. Today manufacturing machines are controlled using Gcodes generated by a CAM system [8]. Each code describes one or more axis movements. A part is machined by executing millions of these codes in the right order with the right setup and the right tooling. Change is difficult which makes manufacturing inflexible and causes long delays while engineers validate incomplete models.

Part 21 Edition 3 can replace these codes with JavaScript running STEP-NC. The broad concept is to divide the STEP-NC program into modules each describing one of the resources in the program. For example, one module may define a toolpath and another module may define a workingstep. The modules can be put into a ZIP archive so the data volume will be less and the data management easier. The JavaScripts defined for each module makes them intelligent. For example for a workingstep, the script can control the tool selection and set tool compensation parameters.

Before a script is started other scripts may be called to make decisions. For instance an operation may need to be repeated because insufficient material was removed, or an operation may be unnecessary because a feature is already in tolerance. Such functionalities can be programmed in today's Gcode languages but only with difficulty.

The JavaScript environment is suited to manufacturing because it is event driven. Performance should not be an issue because in practice machine controls operate by running look-ahead programs to predict future movements. Changing the look-ahead to operate on JavaScript instead of Gcode is probably a better use of resources.

For an example of how such a system might operate see Figure 4 which is a screen capture of the following WebGL application:

http://www.steptools.com/demos/nc-frames.html?moldy/

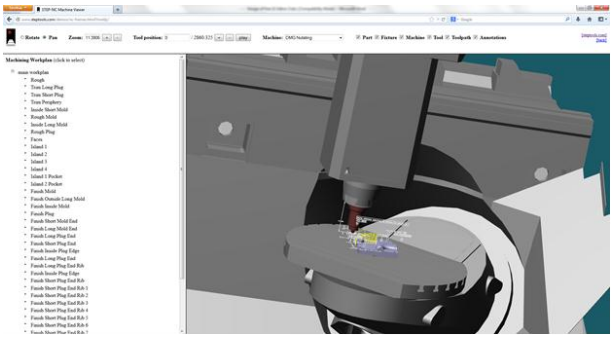

Fig. 4. - WebGL Machining

6 CONCLUSIONS

Part 21 Edition 3 is an upward compatible enhancement to Part 21 that enables data linking and intelligence. Editions 1 and 2 have been highly successful. Nearly every CAD, CAM and BIM system has a Part 21 interface.

The new edition adds URI's and JavaScript to enable the crowdsourcing of massive product models. Other supporting features include a schema population to keep track of all the components, ZIP archives to enable better data management, data relaxation to enable easier interface programming, and data tags to allow application specific programming.

The new specification is not yet finished. Additional extensions are still being considered. They include allowing the files in a ZIP archive to share a common header section, merging the anchor and reference sections into an interface section and loosening the syntax to allow for lists of URI's.

The current specification can be accessed at:
http://www.steptools.com/library/standard/p21e3_dis_preview.html

Implementation tools are being developed to enable testing. They include libraries to read and write the data into a standalone JavaScript system called NodeScript, libraries to steam the data into web browsers, and libraries to run the scripts in an SDAI. The NodeScript implementation is available as open source at the following location.

http://www.steptools.com/library/standard/

The next steps (pun intended) are to:

1. Complete the prototype implementations so that they can be used to verify the specification.
2. Submit the specification to ISO for review as a Draft International Standard (DIS).
3. Respond to the international review with additional enhancements for the additional requirements.

4. Begin the development of common object models for design and manufacturing applications. For example, object models for the execution of machine processes, and object models for the definition of assembly tolerances.
5. Create applications to demonstrate the value of the specification.

The applications will include attention grabbing ones that use kinematics to show the operation of products and machines, value added ones that use the JavaScript to link data sets and create massive product models, and manufacturing ones to verify tolerances while processes are running.

7 REFERENCES

1. ISO 10303-21: Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure (2002).
2. Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force RFC 2396 August 1998, Available from World Wide Web: http://www.ietf.org/rfc/rfc2396.txt (1998).
3. ISO 10303-1: Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles (1994).
4. ISO/CD 10303-242: Industrial automation systems and integration - Product data representation and exchange — Part 242: Application protocol: Managed Model-based 3D Engineering (2012).
5. ISO 10303-238: Industrial automation systems and integration — Product data representation and exchange — Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. (2007).
6. ISO 16739: Industry Foundation Classes for data sharing in the construction and facility management industries (2013).
7. ISO/TS 10303-28: Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas (2007).
8. M. Hardwick, Y. F. Zhao, F. M. Proctor, A. Nassehi, Xun Xu, Sid Venkatesh David Odendahl, Leon Xu, Mikael Hedlind, Magnus Lundgren, et al. “A roadmap for STEP-NC-enabled interoperable manufacturing”, the International Journal of Advanced Manufacturing Technology, Volume 66, Nos. 1-4, Springer Verlag, March 2013.
9. ISO/IEC 16262, Information technology -- Programming languages, their environments and system software interfaces -- ECMAScript language specification (2011).