NISTIR 6219

Design and Development of a Dictionary Translator

John V. Messina
James A. St. Pierre
Michael R. McCaleb

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

January 1999

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary R. Bachula, Acting Under Secretary
for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Raymond G. Kammer, Director

U.S. DEPARTMENT OF ENERGY
Washington, D.C. 20858

Design and Development of a Dictionary Translator

By: John Messina, Jim St. Pierre, Mike McCaleb

Abstract:

In support of the electronic industry's need for quick access to technical data associated with electronic components (such as typically found in hard-copy "databooks"), the Electronic Information Technology Group of the National Institute of Standards and Technology (NIST) has been doing research related to the development of on-line databooks. One facet of this research involves the development of on-line data dictionaries that contain definitions of technical terms (e.g., rise time) that might be used (e.g., hyperlinked) within on-line databooks. Relating to on-line dictionaries, the Silicon Integration Initiative (Si2) consortium has defined an SGML (Standard Generalized Markup Language) DTD (Document Type Definition), that is referred to throughout the remainder of this paper as the CIDS DTD. This DTD enables the viewing of dictionary data, which conforms to the DTD, with an SGML browser. However, as existing dictionary files are not conformant with the CIDS DTD, a translator, called 2CIDS, and an intermediate file format have been developed to aid in the conversion of dictionary data files from their existing formats to files that comply with the CIDS DTD. This paper discusses the development of this translator and the intermediate file format. Both the translator and the intermediate file format represent elements of the infrastructure necessary to support electronic commerce of component information.

1. Background

1.1 Relationship between the NIST ECCI project and the Si2 ECIX project

NIST and the Silicon Integration Initiative (Si2 – formerly the CAD Framework Initiative (CFI)) consortium co-sponsored a workshop on the "Electronics Industry Components Library" in October of 1990. Since that time, both NIST and Si2 have had ongoing projects related to this topic. Today the NIST project addressing this topic is known as the Electronic Commerce of Component Information (ECCI) project, and the related Si2 project is known as the Electronic Component Information Exchange (ECIX) project. One of the early activities that came out of Si2 (CFI at the time) is called the Pinnacles Component Information Standard (PCIS).

NIST has supported the PCIS project, whose goal is to define a standard format for on-line databooks for electronic components. PCIS grew out of an industry need for quick access to technical data (typically found in hard-copy "databooks" distributed by electronic component manufacturers). The PCIS project began by trying to define a set of Standard Generalized Markup Language (SGML) elements (or tags) that would provide semantic information about electronic components. Once the electronics industry agrees on the set of markup tags for the PCIS, as defined within a Document Type Definition (DTD), then component manufacturers will be able to create on-line databooks in an industry standard format. This on-line databook capability is a key piece of the infrastructure necessary to support electronic commerce for the electronics industry. Initial demonstrations of actual component manufacturers databooks, in PCIS format, were given at the 1998 Design Automation Conference.

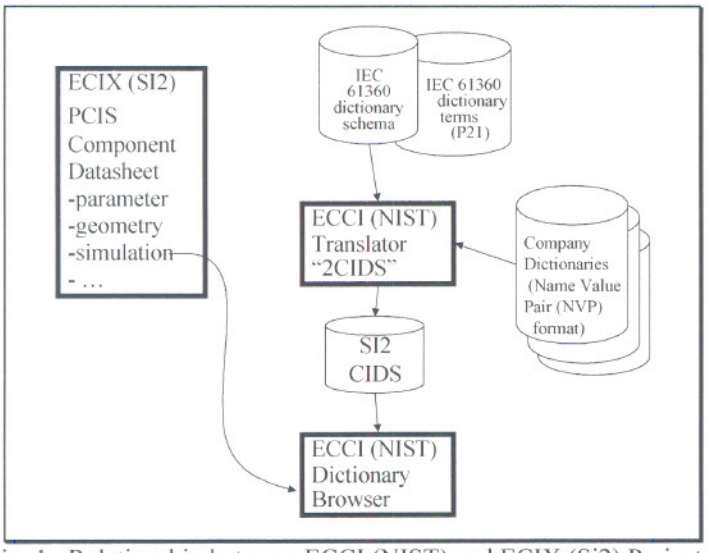

Fig. 1. Relationship between ECCI (NIST) and ECIX (Si2) Projects.

1.2 Data Dictionaries

Other key pieces of infrastructure necessary to support electronic commerce for the electronics industry are on-line dictionaries to support on-line databooks. Early in the ECCI activity, industry requested NIST's help in addressing a significant issue; the terminology used to describe electronic components. The problem is that different companies often use different terminology to describe the same concept, or in some cases they use the same terminology to described variations of a concept. This makes the task of comparing components from different manufacturers much more difficult. In an effort to address this problem, NIST and Si2 have both been working with the International Electrotechnical Commission (IEC), to investigate methods for integrating the IEC dictionary with on-line databooks. In collaboration with IEC, NIST developed software to convert the IEC 61360 [1] dictionary and schema into software objects, and associated software with a Graphical User Interface by which to browse these dictionary objects (see the box labeled "NIST Browser" in Fig. 1). At the same time Si2 has been working with industry to define a standard dictionary format, known as the Component Information Dictionary Standard (CIDS), that is based on the IEC 61360 dictionary and schema. This standard dictionary format is specified with an SGML DTD. Consequently, data dictionaries stored in this format may be viewed with SGML browsers.

It has become clear that even if the electronic version of the IEC 61360 dictionary is available at low or zero cost (at the time of this writing, the cost of accessing this standard on-line has not been defined), there is still a legacy of dictionaries (within electronic component manufacturers) which must be addressed. The easiest solution would be to define a single industry consensus dictionary. However, while the industry may evolve to that in the future, today there are multiple dictionaries that must be accessible. One possible scenario is for the IEC dictionary to be used as a base dictionary, and if company specific terminology is required (to describe new or emerging technologies), they would be submitted through a national committee to the IEC for possible inclusion in the IEC dictionary. In light of these issues, the Si2 consortium requested NIST's help in developing a software program that could translate existing dictionaries from either the IEC or private companies into SGML documents compliant with the CIDS DTD. Such a software translator is an integral piece of the infrastructure necessary for electronic commerce within the electronics industry. Unfortunately there is little commercial benefits or incentives for private industry to develop such a translator. It is for this reason that developing such a translator is a very appropriate task for NIST. John Messina of EEEL's Electricity Division has been working with Si2, IEC and others to develop such a translator, called 2CIDS. The remaining sections of this paper describe the design and development of this translator.

2. Input & Output Data Formats of the 2CIDS Translator

The 2CIDS translator is designed to convert any data dictionary that is in either name-value pair or Part 21 format into a standard dictionary format as described in the Component Information Dictionary Standard (CIDS). Data dictionaries are currently stored either as a Part 21file (see clause 2.1) based on a specific EXPRESS model or as a file in some independent proprietary format. Note: EXPRESS is a data modeling language that is defined in ISO 10303-11 [2].

It was clear from the start that the 2CIDS translator would at the very least need to support translation from the Part 21 format. This is due to the fact that the IEC selected the Part 21 format as the official distribution format of the dictionary. In addition, dictionaries maintained within companies and other standards organizations must also be considered. Rather than require companies to convert their existing dictionaries into the Part 21 format, a new generic data format was developed. This new data dictionary format, called Name-Value Pair (NVP), was designed to be a generic easy-to-generate format into which any dictionary could be converted.

The above requirements meant that the 2CIDS translator had to be able to handle three separate data formats: Part 21, NVP, and CIDS SGML. Before describing the 2CIDS translator in detail, a basic understanding of these three different data formats is desirable. Below are brief descriptions of the three data formats along with a sample data set.

2.1 Part 21 File Format

A Part 21 file is a text file that conforms to ISO 10303-21, "Clear text encoding of the exchange structure" [3]. ISO 10303-21 is an ISO standard that describes the file format for storing data based on EXPRESS schema. There are two levels of conformance to ISO 10303-21, syntactical and schema conformance. To be syntactically conformant, a file must comply with ISO 10303-21. To be schema conformant, a file must be both syntactically conformant and must fulfill all the requirements and constraints specified by a corresponding EXPRESS schema. The names of the EXPRESS schemas, as well as the Part 21 file name and file description, are stored in the header section of the Part 21 file. The data section of a Part 21 file contains instances of the entities defined in the corresponding EXPRESS schema. Attribute values are specified within the entity instances in the order in which they occur within the EXPRESS schema.

Example: Fig. 3 is an example of a portion of a data section of a Part 21 file that is based on the entity EXPRESS declarations shown in Fig. 2.

Sample Express Schema

```
ENTITY dates;
    date_of_original_definition: date_type;
    date_of_current_version: date_type;
    date_of_current_revision: OPTIONAL date_type;
END_ENTITY;

ENTITY item_names;
    preferred_name: pref_name_type;
    synonymous_names: SET OF syn_name_type;
    short_name: short_name_type;
    languages: OPTIONAL present_translations;
    icon : OPTIONAL graphics;
END_ENTITY;
```

Fig. 2. Portion of an EXPRESS schema.

Example Part 21 Data

```
#3=DATES('1998-05-21', '1998-05-21',
  $);
#552=DATES('1998-05-21', '1998-05-21',
  $);

#103=ITEM_NAMES(LABEL('reference
  temperature'), (), LABEL('T_ref'), $,
  $);

#15000=ITEM_NAMES(LABEL('male    IEC169-
  1(3.1.5) (1987)'), (),
    LABEL('H'), $, $);
```

Fig. 3. Portion of Part 21 data file.

Note: The character "$" is used as a placeholders for optional attributes in which a value is not specified.

2.2 Name-Value Pair Format (NVP)

The NVP format was designed as a generic intermediary format for data dictionaries. The NVP format was derived from both the IEC 61360-2 [1] EXPRESS schema and the Part 21 file format. However, it was designed to store all information within a single file as opposed to separate EXPRESS schema and Part 21 files. The NVP format closely matches the EXPRESS schema hierarchy, facilitating the output of the dictionary data into CIDS SGML format. As such, all dictionary parts are stored in the same hierarchical and naming format as described in the EXPRESS schema listed in IEC 61360-2 document. The main difference is that a Part 21 file stores the actual instances of data separate from their semantic meanings. Therefore, a Part 21 file requires a corresponding EXPRESS model in order to be properly interpreted. The NVP format eliminates this dependency by storing data type, attribute names, and class hierarchy information along with the actual data instances.

For each dictionary part there are two separate pieces of information stored in the NVP file (Fig. 4). First, each dictionary part contains an entity line that declares the part's unique identifier (e.g., #3 in Fig. 4), along with entity super/sub type information about the part. The entity super/sub type information is used as a marker to identify appropriate attributes as each super/sub type listed can have its own attributes. Following each entity line is a series of attribute value lines. Each attribute value line contains the class hierarchy for each attribute along with the attribute name and its value. The entity hierarchy is added to alleviate possible naming conflicts that would occur if an entity inherited multiple attributes that have the same name.

```
#3 DATES&
        DATES.DATE_OF_ORIGINAL_DEFINITION
        1998-05-21
        DATES.DATE_OF_CURRENT_VERSION 1998-
        05-21
        DATES.DATE_OF_CURENT_REVISION $

#103 ITEM_NAMES&
        ITEM_NAMES.PREFERRED_NAME reference
        temperature
        ITEM_NAMES.SYNONYMOUS_NAMES ()
        ITEM_NAMES.SHORT_NAME T ref
        ITEM_NAMES.LANGUAGES $
        ITEM_NAMES.ICON $
```

Fig. 4. Name-Value Pair Sample Data

2.3 CIDS SGML Format

"The Standard Generalized Mark-up Language (SGML)" [4] is an ISO standard (ISO 8879) for defining document structures used in markup languages. SGML provides a consistent method for applying structural information to individual elements of a document through the use of tags. SGML elements typically consist of matching opening and closing tags surrounding some component of the document. The text component contained within the matching tags is its content. Using SGML mark-up, documents can provide structural, presentational, and semantic information along with content. The grammar rules of the SGML mark-up language are declared in a DTD. CIDS includes a DTD that defines the allowable mark-ups in the standard dictionary format (Fig. 5). In fact, the CIDS DTD was developed directly from the EXPRESS schema listed in IEC 61360-2. As such, there is almost a total 1-to-1 mapping from the entity types in the schema to the SGML element tags.

CIDS SGML Sample Data

```
<TIME.STAMPS>
   <DATE.DEF>1998-05-21</DATE.DEF>
   <DATE.VER>1998-05-21</DATE.VER>
   </TIME.STAMPS>

<NAMES>
   <ITEM.NAMES>
   <PREFERRED.NAME>reference
      temperature</PREFERRED.NAME>
   <SHORT.NAME>T ref</SHORT.NAME>
   </ITEM.NAMES>
   </NAMES>
```

Fig. 5. CIDS SGML Sample Data

Example: Fig. 5 shows how the sample data in Fig. 3, and Fig. 4 would be tagged using the CIDS DTD.

3. Translator to CIDS (2CIDS)

3.1 General Description of Translation Process

In the early stages of development, it was apparent that the translator needed to be able to perform two separate tasks based on the input format. The translator needed to be able to parse both NVP and Part 21 files. Unfortunately, those two tasks are very disjointed. NVP files can be parsed without any external information, because they contain all entity and attribute information within a single file. On the other hand, Part 21 files contain only the attribute values but not their associated names. This information is stored in the EXPRESS model upon which the Part 21 file is based. Therefore, in order to parse a Part 21 file the EXPRESS schema, upon which the Part 21 file is based, must be parsed first. No parsing software developed for any single input format would work well on any of the other formats. This meant that supporting these two dissimilar input formats would require two separate processes to parse the input data

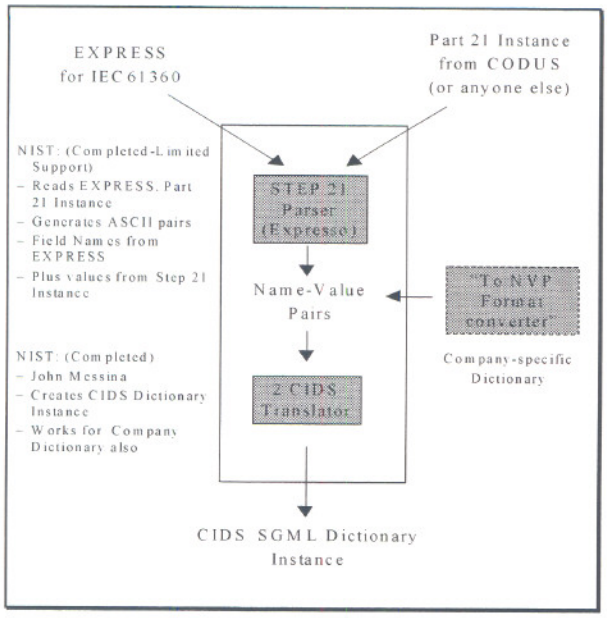

Fig.6. Overall 2CIDS Translator Architecture

Rather than developing two separate translators, the 2CIDS translator was designed around a two-part transformation to convert a data dictionary into a CIDS SGML formatted document. The key to understanding the two-part transformation is the realization that the translator does not need to convert a Part 21 file directly into a CIDS SGML document. The NVP format is designed to be a generic data format into which any data dictionary can be converted. This means that a transformation can be done to convert a Part 21 file into its equivalent NVP file. Once the data dictionary has been converted into a NVP file, a second transformation completes the translation by converting the NVP file into the final CIDS SGML document. These two separate transformations can be combined in a two-part pipeline to convert a data dictionary. Following this pipeline structure, the 2CIDS translator can incorporate both parsing functions and still remain a single process.

3.2 Reasons to Support a Two-Stage Process

Splitting the translator into a two-part pipeline offered two powerful benefits. First, the split nature of the translation process allows the translator to accept two different input data formats. A Part 21 file can be passed through the entire pipeline and converted into a CIDS SGML document. Additionally, the translator can support the NVP format by entering a NVP file directly into the second stage of the transformation pipeline. Therefore, the translator can accept both Part 21 and NVP files as input. The second benefit gained by supporting a two-stage pipeline is that it makes the translator inherently modular. Both components of the translator are orthogonal to each other. This means that either component can be modified without affecting the other component. This was especially important considering the dynamic nature of the project. At one point during the development process the CIDS SGML DTD changed significantly. These changes required the second component of the translator to be modified. However, the modular nature of the translator assured that these modifications would not inadvertently alter the first component of the translator.

3.3 Two Stage Data Transformation Pipeline

3.3.1 First Data Transformation (Part 21 Files to NVP Files)

The first stage of the translator (Fig. 6), which converts Part 21 files into NVP files, is the most complex part of the translator. Any translation of Part 21 files into NVP files must include three primary actions. First, the translator must parse the EXPRESS [1] model upon which the Part 21 file is based. Without such a model, there is no way to

identify the individual attributes for the Part 21 entities. Second, the Part 21 file must be parsed and the values stored in some internal data structure. The parsing process typically includes data type and "WHERE" rule checking on the data in the Part 21 file. These data checks ensure that the translator is receiving a valid Part 21 file. Finally, once the data has been parsed, the translator must generate output for each Part 21 entity along with associated EXPRESS model information. This process involves matching the Part 21 entity instance attributes with their class and type information from the EXPRESS model.

3.3.2 Second Data Transformation (NVP to CIDS SGML)

The second stage of the translator must by nature perform two primary actions: parsing the NVP file and generating the correct CIDS SGML representation of that data. One of the primary design goals of the translator was to avoid the need for complex parsing software such as Lex and Yacc. This design goal was achieved by selecting a simple, repetitive format for the NVP. By avoiding complex input strings and tokens, it is possible to parse the NVP file using simple pattern matching. Once the parsing is complete, the data is typically stored in some internal data representation. The only restriction on the internal representation is that each object must be identifiable based on its entity type. For example, it must be possible to identify every object of a certain entity type listed in the NVP.

After parsing the NVP file, the second stage is responsible for outputting a correct CIDS SGML document. As it is an SGML format, the output consists of a text document containing a series of markup tags, each of which can contain data and/or a series of sub tags. Each entity in the NVP format consists of a data type and a series of attributes linked to specific values. The second stage of the translator must map each NVP entity, along with its attributes, into their corresponding SGML markup tags.

The CIDS DTD clearly dictates the acceptable grammar for the CIDS SGML document. In fact, the CIDS DTD was developed from the IEC 61360-2 EXPRESS schema. As such, there is almost a 1-to-1 mapping from the entities listed in the IEC 61360-2 EXPRESS schema to the CIDS DTD entities. Therefore, for the translator to correctly map the NVP format to the CIDS SGML format, the translator must adhere to the CIDS DTD.

3.4 CIDS Implementation

As stated before, the translation process consists of two separate stages that are mutually exclusive of each other. As such, software components, which implement the data transformations, can be developed separately for each stage. The only requirement for each of the software components is to follow the general procedures described previously for each stage. The 2CIDS software package consists of several software components that were designed to perform the required transformations for each of the two stages. In addition to the main transformation components, 2CIDS also includes a series of Perl scripts designed to aid in the integration of the two parts of the translation pipeline.

3.4.1 First Stage Component

During the development process, two different pieces of software were used to convert the Part 21 file into the NVP file. The first piece of software used was a newly developed piece of software called Expresso [5], which was also developed at NIST. Expresso, written by Peter Denno of the Manufacturing Engineering Laboratory, is a language environment for the EXPRESS language. It provides tools to aid in the development and validation of EXPRESS models and their associated Part 21 files. Included in Expresso's functionality is the ability to run a Lisp command script. Peter Denno provided a Lisp script that was a program designed to output a NVP file of any Part 21 file that had been successfully parsed by Expresso. In fact, an early copy of the Lisp command script provided the original basis for the NVP format.

Due to concerns about the future maintenance of Expresso, a second alternative piece of software was developed to perform the first data transformation. The second software program is called AV, short for attribute/value, and was written by David Sauder of the Manufacturing Engineering Laboratory. AV is a UNIX based C++ program written to parse a Part 21 file based on the IEC 61360-2 EXPRESS model. AV was written using fedex_plus. Fedex_plus is an application of the NIST STEP Class Library [6] that can generate a C++ representation of an EXPRESS schema. Using this C++ representation, AV can successfully parse a Part 21 file and then can output both NVP and Part 21 files.

3.4.2 Second Stage Component

Parser, a Perl version 5.0 program, was written as the software implementation of the second component of the translator. The *Parser* program performs two main functions: NVP parsing and SGML generation. Using Perl's strong regular expression capabilities, it is very straightforward to parse NVP files. Each entity line in the NVP file identifies the start of an NVP entity (Fig. 7) and its unique identification number. Separating the components of the NVP entity line gives all the super type/sub-types of the entity. One or more lines of attribute-value combinations follow each entity line. Each attribute-value line can be easily parsed into its base components: the entity type, the attribute name, and the attribute values.

Entity Line:	#<number>=<entity-type>&[entity-type&] e.g. #3=ITEM_NAMES
Attribute-Value line:	<entity-type><attribute-name> <Value> e.g. ITEM_NAMES.SHORT_NAME pref_Tem

Fig. 7. Structure of a NVP Entity

Parser, once it finishes parsing an incoming NVP file, it stores the entities in an internal data structure. The internal data structure used is an associative array of associative arrays. The outermost array keys on the unique number associated with each NVP entity (Fig. 8). Stored in the outermost array is a pointer to another internal associative array. Each attribute-value combination is stored in the internal array, which is keyed on a combination of class hierarchy and attribute name. The additional element of the class hierarchy was added to the internal key due to the EXPRESS model's ability to redefine attributes. This resulted in some entities having multiple identical attribute fields even

Main associative array:
Main array key on #3 = associative array of entity object #3

Associative array of entity object #3
#3 key on short name = pref_tem
#3 key on preferred name = preferred temperature
#3 key on synonymous names = (empty)

Fig. 8. Example of 2CIDS internal data storage

though only one field was valid. This combination of imbedded associative arrays provided a reasonable method to access any entity element in at most two steps.

Following the CIDS DTD closely, the *Parser* program then generates a CIDS SGML data dictionary using language expansion. An SGML DTD declares element types that represent either structures or desired behaviors. Each element type typically describes three parts: a start tag, any type of content, and an end tag. The CIDS DTD specifies the allowable types of element content. The element's content can be anything from no content, to a series of rules governing which elements are allowed within the current element. Language expansion occurs by taking a given SGML element and expanding any elements that occur within the given tags' content. In this fashion, an SGML document can be generated by starting with a document's outermost element and expanding inner elements.

Parser mimics the language expansion seen in the CIDS DTD. Each SGML element is matched with a function within the Perl program. Each function accepts a pointer to the internal data structure, up to two entity

identifier keys, and an integer depth as input, and returns a character string as its result. For each element, *Parser* performs three primary actions. First, it creates a string containing only the element's start tag. Second, it generates the element's content. This is usually done by either calling functions corresponding to internal elements or by accepting data directly from the internal data structure. Any content data returned from these sub-functions or the internal data structures are added to the return string following the grammar rules specified in the CIDS DTD. Finally, the function adds any ending tag, if necessary, and passes the resulting string back to the calling function. In this fashion, *Parser* can be seen as a program that essentially takes one outer SGML element and slowly builds the internal content as one long data string.

This modular approach was taken to aid in debugging and to support future modifications to the 2CIDS translator. By implementing each element as a function, it was possible to isolate both errors and modifications to specific functions. In most cases, errors could be traced to a specific function by identifying the surrounding SGML tags. With regard to modifications, each function could be modified on an individual basis independently. The benefits of this feature were validated during the development process; near the end of the development cycle, a series of modifications were made to the CIDS DTD in order to better support current SGML browsers. The necessary modifications were made to *Parser* by incorporating the changes to the CIDS DTD one element at a time. After all the modifications were completed, only the functions that had been changed needed to be re-tested; the modular design of *Parser* eliminated the need to extensively re-test the entire program.

3.4.3 Supporting software components

In addition to supporting two separate transformation stages, 2CIDS also includes two additional Perl scripts: *Normalizer* and *Translator*. *Normalizer* is a Perl script designed to reduce inconsistencies in implementations of the NVP format. Although the NVP format is clearly specified, there turned out to be several minor differences in the AV and Expresso [5] NVP formats. For example, AV and Expresso deal differently with extraneous return characters in text strings in Part 21 files. Quite often Part 21 files will be formatted for easy human reading by adding additional return characters. AV, assuming these return characters are intentional, passes along these characters, while Expresso removes them from the output. *Normalizer's* only role is to iron out these types of minor differences until they can be addressed in the first data transformation components.

The second script, known as *Translator*, was written to improve the usability of the UNIX version of the 2CIDS translator. Currently the 2CIDS translator reads data and performs multiple transformations on it. To reduce the workload of the users, *Translator* was written so that only a single command needs to be executed. The *Translator* script runs both AV and *Parser* and controls all data flow between the two main translation stages. In addition, *Translator* was written to support error and logging functions to aid with debugging. This reduced running the 2CIDS translator to making a single UNIX command line call. Unfortunately, there is no easy way to make a single 2CIDS script to perform a similar function with the PC version of 2CIDS.

4. 2CIDS Data Execution

The testing of the 2CIDS translator occurred in two distinct phases: development testing and final testing.

4.1 Development Testing

The development testing stage consisted of running dictionary fragments through the translator to test individual modules in *Parser*. *Parser* was designed to match the CIDS DTD on an element-by-element basis. As such, there is one Perl function per CIDS element. This design choice made it possible to test the correctness of the translator a single element at a time. For each function, a sample dictionary fragment in Part 21 format was entered that contained an instance that corresponded to the relevant CIDS SGML element. The translator then generated the SGML output for that dictionary fragment and the output was visually inspected for errors. The problem was locating dictionary fragments that contained specific dictionary data elements that mapped to certain CIDS DTD data elements.

In the end, sample dictionary fragments were gathered from two primary sources: the IEC 61360-2 [1] document and Si2. The IEC 61360-2 document contains a limited example of a Part 21 file based on the IEC 61360-2 EXPRESS model. While it served as a good starting point for testing, it lacks enough scope to be of use beyond a few

instances. Si2 proved a much better source of Part 21 dictionary fragments. Members of several groups, including Si2, CODUS [1], and CIREP (Component Information Representation European Project), collaborated on specific Part 21 example files for use in a mapping exercise; converting a Part 21 file to the final CIDS SGML document. These data samples became the primary test input for the 2CIDS translator in the early development stage.

As a result of using these dictionary fragments, most of the 2CIDS translator was fully tested without the use of a complete dictionary. Unfortunately, the main problem with this testing methodology was that the 2CIDS translator was tested against fabricated data dictionaries. As such, there was no guarantee that the Part 21 example files where 100% correct. In addition, the CIDS SGML output at this stage was tested by visual inspection. At the time of development, there were no SGML browsers available to test the 2CIDS output for SGML grammatical correctness. Advanced testing of the 2CIDS output had to wait until complete data dictionaries and an SGML browser was available.

4.2 Final Testing

Finally, in order to support the Si2 Pinnacles Component Information Standard demonstration given at the 1998 Design Automation Conference (DAC), two different versions of data dictionaries were run against the 2CIDS translator. The first data dictionary was provided by CODUS to Si2 for use in the PCIS demonstration. The second dictionary was a partial dictionary provided by CIREP. Although the CIREP data dictionary was in a Part 21 file, it was based on a slightly different EXPRESS schema than the one specified in IEC 61360-2. As such, these two data dictionaries provided both the means to conduct final testing, as well as help in gathering final run results.

The 2CIDS translator can provide excellent Part 21 format checking; an unexpected result. Both the CODUS and CIREP data dictionaries, in Part 21 format, were in the early stages of development prior to DAC. As such, there were certain semantic issues that needed to be worked out for the dictionaries to be conformant to the IEC 61360-2 EXPRESS schema. The use of Expresso [5] and AV provided valuable data checking services to both CODUS and CIREP.

As expected, the use of a complete data dictionary identified weaknesses in the 2CIDS translator. One source of errors came from unexpected character interactions within the SGML output. These were mostly minor problems, such as the need to convert special characters that were being interpreted by the SGML browser to their ASCII character code equivalents. For example, '<' needs to be converted to '&lt' whenever it appears within an SGML tag's content. Once these errors were identified, through the use of the SGML browsers that were used at DAC, they were easy to fix.

Another weakness that was discovered during testing of the 2CIDS translator with the full version of CODUS's data dictionary is excessive run time. The EXPRESS language has a construct called an "inverse relationship". An inverse relationship is established whenever an attribute references an entity. The direction of this relationship is from the referenced entity to the entity containing the attribute. Because all references in a Part 21 file are one-way (towards the referenced entity), there is no easy way to calculate an inverse relationship. In particular, *Parser* must actively scan the entire list of dictionary entities each time an inverse relationship is needed. Run times of large data dictionaries have been recorded of taking over 2 hours to complete on certain slow systems. Luckily, a data dictionary conversion should never be required to run in real time. The translator will be run once or twice for each dictionary as it is released. Therefore, the excessive run times are more an irritation than an active problem.

A final problem in the 2CIDS translator is in the connection between the first stage and second stage of the translator. The second half of the translator, *Parser*, requires a syntactically correct NVP file. This was hard to achieve because the NVP format was developed along with the translator itself. As such, the two software components used for the first stage of the translator, Expresso and AV, both generate slightly different NVP files. While *Normalizer* is used to fix most of these problems, it is clearly a patch intended as a temporary solution. Now that the NVP format is more clearly defined, it should be possible to improve the compatibility between the two transformation components.

---

[1] CODUS: A British company under contract with the IEC to maintain and distribute the IEC 61360 dictionary

Other than these few minor problems and concerns, the 2CIDS translator works well on both the CIREP and CODUS dictionaries. Data from both sets of dictionaries were used in demonstrations at the 1998 Design Automation Conference in San Francisco.

Note: The 2CIDS software can be downloaded from the following World Wide Web page:
http://megavolt.eeel.nist.gov/~messina/translator

5. Possible Future Enhancements & Work

5.1 Enhancements

There are several possible enhancements that will be considered for any future versions of the 2CIDS translator. First, while not strictly necessary, one enhancement would be to improve the run time of the 2CIDS translator. There are actually two different ways a speedup could be achieved. One way would be to modify the data structure so that each entity contained pointers to both its children and parents. This would shift the time spent searching for inverse relationships to the creation of the data structure rather than during the generation of the SGML output. The other possible modification would be to break the one main associative array into several different arrays based on the entity type. This would reduce the number of entities that have to be searched for each inverse relationship.

Another much needed improvement is a better interface to the PC version of the 2CIDS translator. The UNIX version of the 2CIDS translator has a single Perl script, *Translator*, which executes both stages of the data transformation and handles all data flow from one component to another. The PC version does not have the benefit of a single execution software application. In other words, there is no current way to execute both the Expresso [5] and the *Parser* components of the 2CIDS translator in a single step. One possible enhancement would be to write a Lisp script to handle the second data transformation. This would essentially place both data transformations within the Expresso program itself. Alternative methods might include some sort of batch file that will execute both stages of the transformation in sequence.

One potential additional capability that has been suggested for future addition to the 2CIDS translator is the ability to handle incremental updates. At the moment, it is unclear to which format these updates will be made. However, it is clear that these updates will need to be added to any previously translated data dictionaries. It might be necessary to make these updates having only the updates and the previously generated CIDS SGML output. If this turns out to be the case, then the 2CIDS translator may need to incorporate an SGML parser in order to read the previously generated CIDS output.

5.2 Possible Future Related Work

It is interesting to note that the Extensible Markup Language (XML) is emerging as a very important standard for electronic commerce, because it allows the creation of World Wide Web (WWW) pages that are much more intelligent than their Hyper Text Markup Language (HTML) predecessors. SGML was considered too complex to be easily implemented on the WWW, and so a subset of it is being defined as the XML standard. Because of this close relationship between SGML and XML, the CIDS and PCIS standards are poised to quickly take advantage of XML browsers that are currently in the development stages. A future version of 2CIDS might include the ability to output an XML version of a CIDS dictionary.

Another step forward in technology regarding the CIDS concerns tools to compare differing terminology. Even with dictionaries converted into the CIDS format, comparing two similar terms requires human analysis. While this method is useful in identifying differences between same term definitions, it is also time consuming. In the future, it will be useful to have software tools that can do basic comparisons on dictionary terms in CIDS. Beyond a simple match comparison, which can identify if the two terms are defined in the same dictionary, these tools should be able to do a similarity rating. It could be very helpful for a tool to give a rating of how similar two term definitions are, based on things like topic match, word placement, word frequency, etc. Such tools could greatly simplify the process of finding similar electronic components.

6. References

1. IEC 61360-2/FDIS:1997, *Standard data element types with associated classification scheme for electronic components – Part 2: EXPRESS Dictionary Schema.*

2. ISO 10303-11:1994, *Industrial automation systems and integration—Product data representation exchange—Part 11: Description methods: The EXPRESS language reference manual.*

3. ISO 10303-21:1994, *Industrial automation systems and integration—Product data representation exchange—Part 21: Implementation methods: Clear text encoding of the exchange structure.*

4. ISO 8879:1986, *Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML).*

5. NIST Expresso Home Page: URL: *http://www.mel.nist.gov/div826/staff/denno/nist-expresso.html*

6. The NIST STEP Class Library (SCL) URL: *http://www.mel.nist.gov/msidstaff/sauder/SCL.htm*