Evaluating Manufacturing Machine Control Language Standards: An Implementer's View

Thomas R. Kramer
National Institute of Standards and Technology
MS8230
Gaithersburg, MD 20817, USA
kramer@cme.nist.gov

Abstract —The focus of this paper is: how can standards for manufacturing machine control languages be evaluated? What is required of a standard defining one of these languages so that implementations will interoperate? The paper provides a set of specific questions to ask about a control language standard. Reasons why the questions should be asked are given. Four machine control languages are used as examples:
EIA-274-D, BCL, DMIS, and STEP-NC.

Keywords: *control, language, machine, standard, AP 238, BCL, DMIS, EIA-274-D, ISO 10303, ISO 14649, STEP-NC*

I. INTRODUCTION

The focus of this paper is: how can standards for manufacturing machine control languages be evaluated? What is required of a standard defining one of these languages so that implementations will interoperate?

The paper deals only with languages intended to be used in control program files. There are also manufacturing machine control languages (such as the I++ DME Interface Specification and DMIS Part 2) intended to be used for transmitting individual commands one at a time. The issues for those languages are similar, but they are not addressed here.

In this paper, "manufacturing machines" means things such as machining centers, turning centers, and coordinate measuring machines that are run by computer numerical control systems.

The interfaces served by the standards are those between program generators and program execution systems. The programs that travel over these interfaces need to be capable of exercising the full functionality required by the task at hand (in general, all or almost all of the functionality of the receiving system). Hence, a manufacturing machine control language must provide a suite of program statements or commands that exercise that functionality.

The systems on the two sides of the interface must interoperate in the sense that a program passing across the interface must be executable by the receiving system and must do what the generator of the program intended it to do. Moreover, the intended meaning must be as described in the standard.

A standard for a language should describe how program statements should be executed in enough detail that if the standard is implemented in a number of sending systems and a number of receiving systems and all systems conform to the standard, any receiving system will do what is intended in any program generated by any sending system.

II. WHY AN IMPLEMENTER'S VIEW?

An implementer is a person who programs the software on one side or the other of an interface so that the software either generates programs or reads and executes programs. Because building an implementation requires understanding all the details of a language, even the most minute, implementers are the people in the best position to judge whether a standard is complete and unambiguous. Standards are usually written with the implementer as the primary audience. One World Wide Web Consortium web page, for example, says "Specifications are aimed at people writing software to implement them" [15].

The implementer's view of a control language standard is different from the standard writer's view in the same way that a file reader is different from a file generator. The standard writer has concepts in mind and puts down statements in the formal language used to define the standard, statements in natural language, and diagrams to represent those concepts. The implementer reads the formal language, natural language, and diagrams and forms concepts from them. Just as parsing from characters into structures is much more difficult than generating character strings from structures, so implementing a standard is much more difficult than writing it.

In implementing a standard, the implementer is continually asking, "Is there more than one way in which this can reasonably be interpreted?", and "Do I have to make some assumption about the meaning in order to implement it?" If the answer to either question is yes, the implementer can be confident that some other implementer will make the other choice or assumption, so that the other implementation will not interoperate.

III. EXAMPLES OF STANDARDS FOR MANUFACTURING MACHINE CONTROL LANGUAGES

The following manufacturing machine control language standards will be used as examples. The author has direct experience with all of them by having built an implementation and/or studied the standard and submitted detailed comments to the committee responsible for the standard.

A. EIA-274-D

EIA-274-D, dated February 1979, [4] is a standard of the Electronics Industry Association (EIA) and is a low-level language designed for execution on the controller of a machining center or turning center. Another name for it is RS-274-D, since it is also an ANSI standard under that name. It is informally called a "G and M code" standard, since it consists primarily of codes starting with G or M. In this respect and in meaning, it is similar to ISO 6983.

B. BCL

BCL [13] is a low-level language designed for execution on the controller of a machining center or turning center. It is a language whose acronym outlived its original name. The name started in 1983 as "Binary CL" (EIA 494 A). The "CL" probably stood for Cutter Location. The standard did not say what CL stood for. By the February 1997 proposal for EIA 494 C, BCL had changed into "Basic Control Language". The language itself changed also over that period from a terse gobbledegook of letters and digits into human-readable abbreviated English command names accompanied by parameter values having primitive data types (keyword, string, number, etc.), all represented using ASCII (American Standard Code for Information Interchange) characters.

C. 3. The STEP-NC milling family: ISO 14649 and STEP AP238

ISO is the International Organization for Standardization. STEP is the STandard for the Exchange of Product model data. ISO 14649 and STEP AP 238 are high-level languages for various types of numerically controlled machines. These standards are still being developed. The most mature parts (subdivisions of a standard) are applicable to machining, specifically Parts 10 and 11 of ISO 14649 [8], [9]. Only those two parts are discussed in this paper.

AP 238 [7] is largely a recasting of ISO 14649 semantics into the terms of the STEP "integrated generic resources" so that machine control programs may be processed (to a certain extent) by any system that can handle the STEP integrated generic resources. STEP itself is a series of several dozen parts designed for product data representation and exchange. All the STEP parts are part of ISO 10303. AP 238 encompasses multiple parts of ISO 14649. Only the portion of AP 238 relevant to machining centers is covered here.

These languages are "high-level" in the sense that they are designed to communicate geometry, machining operation data, and machining strategies and to leave other decisions (the generation of toolpaths, in particular) to the controller. However, they also include facilities for sending low-level commands that give tool paths in detail.

D. DMIS (Part 1)

DMIS (Dimensional Measuring Interface Standard) is a mid-level language for programs for coordinate measuring machines (CMMs) and other dimensional measuring equipment [2]. Since it must do numerical data analysis while it executes, DMIS requires much more complex execution software than do the EIA-274-D and BCL languages, but it does not require the use of strategies that STEP-NC requires. DMIS has been updated through five versions, starting in 1986. DMIS 5.0 is the most recent standard.

DMIS defines both a programming language (the input format) and a format for data reporting (the output format). Only the programming language is covered in this paper. There is a DMIS Part 2, which is an object interface specification. It is not further discussed here.

IV. QUESTIONS FOR EVALUATING A MANUFACTURING MACHINE CONTROL LANGUAGE STANDARD

To evaluate a manufacturing machine control language standard, the following questions should be asked.

A. Is the standard complete for the intended use?

B. Is the standard clear and unambiguous?

C. Is the standard defined using a high-level information modeling language for which processors (readers and code generators) are readily available?

D. If the standard is defined using a high-level information modeling language, is there a well-defined file representation that works with the high-level language?

E. If the standard is defined directly as a file format (i.e. not by using high-level language), is the method used to define the file format clear and unambiguous?

F. Is special software required for reading and writing program files or for assembling the file data into meaningful structures? If so, is it widely available, and is it free or costly?

G. Has the standard been tested? How? What were the results?

H. Is there a continuing committee devoted to maintaining the standard? What is the committee's track record of dealing with proposed changes?

I. Are there intellectual property issues that may make using the standard impossible or expensive in the future?

J. Is there a critical mass of conforming implementations of the standard? Does it appear there will be a critical mass in the future?

K. Does the standard have conformance classes? Are they part of the standard?

L. Is it necessary to follow a set of usage rules additional to the standard in order to build an interoperating implementation?

M. Are there mechanisms in place (formal or "natural") to insure that implementations conform to the standard?

V. DISCUSSION OF THE QUESTIONS

How much weight to give to the various questions depends on one's point of view. Most of the people to whom the performance of a manufacturing machine control language standard is important are in one of two groups: (A) end users trying to decide whether to acquire and use a system that implements the standard, or (B) systems developers trying to decide whether to implement the standard (particularly those working for systems vendors or for users building their own systems). End users who want to buy a system rather than to build one, for example, may not care how hard it is to build an implementation. A person building an implementation, as another example, may not care whether there are other implementations of the same sort.

The discussion of examples in this section reflects the opinion of only the author.

A. Is the standard complete for the intended use?

From a user's point of view, in order to determine whether a standard is complete enough, the user should make a list of the required machine functions and then determine whether the standard supports those functions. This may be done by studying the standard, by getting information from users of the standard, by observing conforming implementations in action, or by some combination of those.

There is no global answer to this question because the meaning of "the intended use" is dependent on who is doing the intending. There are no fixed boundaries on the set of people who might be users of a standard.

All of the examples are complete for simple use on machines of the sort for which they were originally developed, but all of them could be extended for more advanced functionality or for control of similar but different machines.

As an example of additional functionality on a target machine, EIA-274-D, which was designed to be used on machining centers, specifies that codes G36 to G39 are "permanently unassigned" (meaning revisions of the standard should not use them) and available for "individual use". Kearney and Trecker built a machining center with a broken tool detector, and extended EIA-274-D by using G38 to operate the detector [11].

As an example of a machine that falls a little out of the ballpark from the original target machines, DMIS was designed for doing dimensional measurements on a coordinate measuring machine using a touch probe, so DMIS has a "measure a point" command. Dimensional measurement can be done using a theodolite, but a point cannot be located all at once with a theodolite. It is necessary to take at least two measurements of angles to the same point and then calculate its location. DMIS does not provide a command to do that.

B. Is the standard clear and unambiguous?

For clarity and unambiguousness, there are two largely separable areas requiring attention: syntax and semantics. Syntax covers what tokens (i.e words, numbers, and special symbols), statements, and sequences of statements can legally be written. Semantics covers what a token, statement, or sequence of statements means.

Modern formal languages exist (EBNF, for example) that make it possible to specify the syntax of a control language very precisely. It is also possible to be precise about syntax in natural language, but that is more difficult.

There are no formal languages that make it possible to specify semantics. Only natural language and diagrams are available for conveying the meaning of a standard, and it is very difficult to specify semantics by these methods.

The level of being clear and unambiguous of most of the machine control standards the author has seen is not very high.

Ambiguity in a standard may be intentional or unintentional, but in either case ambiguity defeats interoperability. EIA-274-D, for example, is filled with intentional ambiguity. Appendixes A.1, A.2, and A.3 provide that each implementer can specify a host of things (such as how numbers can be written and whether dimension values are absolute or incremental) that need to be agreed between a program generator and a program executor.

STEP NC contains many instances of unintentional ambiguity (for example, the location of most open profiles is undefined). NIST has submitted 153 suggestions for technical changes in Part 10 of ISO14649 and 70 for Part 11. Many of these suggestions aim to eliminate ambiguity.

C. Is the standard defined using a high-level information modeling language for which processors (readers and code generators) are readily available?

Standards that are defined using a high-level information modeling language have a large advantage over those that are not. Examples of high-level information modeling languages include:

• EXPRESS (not an acronym) — developed as part of STEP [5],

• XML Schema (Extensible Markup Language Schema) developed by the World Wide Web Consortium [14].

These languages all contain primitive data types (integers, strings, lists, etc.) and provide for defining the sorts of interlinked data structures that are needed in a machine control language.

When a control language standard is written using one of these high-level languages, a certain amount of automatic processing may be done. Software is available that will read the file defining the control language, check its syntax, and generate source code in a computer language. The source code defines computer language structures corresponding to the structures in the control language and contains functions for accessing (extracting and inserting) the data in those structures. If a machine control language standard is written using a high-level language, the standard will probably have been checked for good syntax using an automatic checker, and a potential user of the standard who has a checker can use it to check the standard.

If a machine control language standard is written using a standard lower-level formal language such as EBNF (Extended Backus Naur Form) [10], it can be checked for syntax automatically, but utilities for generating computer code are not readily available. Moreover, since the lower level formal languages do not define structures, there is not enough information in the control language definition file to produce code useful for building an implementation. Only structures that mirror the syntax can readily be built by an automatic system working from EBNF, and the syntax structure is not likely to be the structure an implementer would like to use.

If a machine control language is written using an ad hoc description method, neither automatic syntax checking nor automatic code generation is feasible.

Since the semantics of a machine control language standard cannot currently be described in a formal language, it is never possible to generate an implementation automatically that does anything more than read program files, rewrite program files, allow browsing, and generate statistics, even if a high-level language has been used to define the control language.

D. If the standard is defined using a high-level information modeling language, is there a well-defined file representation that works with the high-level language?

With a high-level information modeling language, it is feasible to define a generic file format that combines with any information model defined in the language so that a specific file format exists for the model. Thus, when a manufacturing machine control language is modeled using the high-level information modeling language, general-purpose software designed to be used with the high-level language will read or write a file containing an executable machine control program without any work on the part of the implementer other than making a single library function call in a program. This saves an enormous amount of work an implementer would otherwise have to do and makes it much less likely that the reader or writer will not conform to the standard. Of course, when writing, a model of the program must be built before a "write this model to a file" function can be called.

EXPRESS and XML Schema have well-defined generic file formats of the sort just described. EXPRESS works with the STEP Part 21 format [6], while XML Schema works with XML [1]. A high-level language can work with more than one file format. An EXPRESS model can be used with XML, for example.

High-level information modeling languages generally are also built to support implementations that use databases (or persistent objects) and application programming interfaces rather than files for exchanges across an interface, but this paper does not deal with that. It is not common for stored machine control programs to be implemented that way.

The STEP-NC standards are all modeled using EXPRESS. None of the other three examples uses a high-level information modeling language.

E. If the standard is defined directly as a file format (i.e. not by using high-level language), is the method used to define the file format clear and unambiguous?

EIA-274-D uses English to define the file format. The English descriptions are generally hard to follow. There are several unintentional ambiguities and many intentional ones.

Section 3 of BCL (as proposed for EIA 494 C) defines overall file structure in English. Section 4 defines in English what the fields of a BCL record (a single statement) may be and what characters constitute a valid field (such as a text field, parameter separator field, or numerical field). Most of the descriptions in Sections 3 and 4 are clear and unambiguous, but the definition of "numerical field" is ambiguous. Sections 8.0.1 and 8.0.2 of BCL define a higher level syntax notation for defining what sequences of fields constitute valid commands. The higher level notation is clear and unambiguous and is used consistently in the succeeding parts of the standard.

DMIS 5.0 defines its file format two ways. First, much of Section 5 describes syntax in English, and a syntax notation defined briefly at the beginning of Section 6 is used in the remainder of Section 6 (over 400 pages) to define what sequences of fields constitute valid commands. Second, Annex C gives a definition of DMIS syntax in EBNF (although the lowest level, such as what sequence of characters makes a real number, is omitted). The file format of DMIS is thus generally clear and unambiguous. The use of EBNF has enabled the automatic construction of DMIS input file syntax checkers [12].

F. Is special software required for reading and writing program files or for assembling the file data into meaningful structures? If so, is it widely available, and is it free or costly?

In all four examples, there are at least two levels of encoding. All of the examples use ASCII code at the lower level to interpret bits as characters. Managing this level takes no special software. Every common programming language reads and writes ASCII. At the upper level (where reading means to convert a stream of ASCII characters into meaningful structures and writing means to convert structures into a character stream) all the examples except STEP-NC require special software for reading and writing. Typically:

• data structures must be designed,

• a parser must be built to receive a character stream and build and populate a hierarchy of structures, and

• a writer must be built to traverse a hierarchy of structures and generate a character stream.

In STEP-NC, ISO 14649 EXPRESS models can be used directly with STEP Part 21, and no special software is needed for reading and writing files beyond that which can be generated automatically. AP 238, however introduces a third level of encoding. Special software is needed not for reading and writing but for dealing with this third level of encoding. Most of the data in AP 238 is encoded at a level between a character stream and structures meaningful to an application programmer. The middle level is built in terms of entities from the STEP integrated resources, and Part 21 files contain representations of these structures. The structures in this middle level are utterly unintelligible to programmers conversant with machine control. In order to use AP 238 it is currently necessary to have special software that either (1) converts the integrated resources structures into structures like those that may be created directly from ISO 14649 and provides access functions for the 14649-like structures or (2) provides access functions for the integrated resources structures with semantics similar to those that may be created directly from ISO 14649. Currently, only the second method has been implemented, and there is only one provider of this type of special software. Building software of this sort has a high and steep learning curve.

G. Has the standard been tested? How? What were the results?

A manufacturing standard is like a piece of complex software. As with software, mistakes may be made in syntax or logic, and the functionality may not be what is intended by the authors. There is no chance that complex software will be bug free as it comes from the programmer. It must be compiled, tested, and debugged before release. There may be bugs in syntax, bugs in operation (writing beyond the end of an array, for example), and bugs in what the program does. This is universally acknowledged. Commercial software houses always have testing procedures in place. As with software, there is no chance that a complex manufacturing machine control language standard will be bug free as it comes from the authors. It should be implemented, tested, and debugged before final release. This is rarely acknowledged. Most standards development organizations do not have standards testing methods in place that must be applied before a proposed standard may be approved. STEP has testing and conformance procedures, but they are too little and too late to insure that a standard is of high quality at the time of first release.

Computer languages have compilers that make executables that can be tested. Standards do not. The best that can currently be done automatically with a standard, if it is defined using a high-level language, is to build a system that can read program files, rewrite program files, allow browsing, and generate statistics.

EIA-274-D allows so many choices (i.e. is so ambiguous) that only testing one of the many billions of legal variants is feasible. The extent to which a variant has been tested is dependent on the creator or vendor of the variant.

BCL appears to have been well-tested by the large organizations that used it, in close collaboration with the vendor that provided the implementations. If it had been widely implemented (correctly), it could have provided a high degree of interoperability.

There has been no formal testing program for DMIS. Some vendors appear to have implemented DMIS in conformance with the standard and tested carefully. Other vendors have not.

STEP-NC has been tested to a modest extent, but it is far from being fully tested. There are enough ambiguities in the standard that the notion of “conforming implementation” is tenuous. Further implementation tests are under way.

H. Is there a continuing committee devoted to maintaining the standard? What is the committee’s track record of dealing with proposed changes?

Technology advances rapidly so that additional functionality is needed periodically in any standard dealing with machine control. If a standard is not updated when new technology appears, implementers will extend the language in non-standard ways in order to use the technology.

Even if new technology does not appear, machine control language standards are so complex that it takes years to eliminate all the ambiguities and bugs.

To accomplish updating a standard, it is necessary to have a group in place that understands the standard and can judge the merits of proposed changes. Determining what updates are needed works most smoothly if there is an established, well documented process for updating the standard. The process should include consideration of requests from anyone for changes.

Of the examples, DMIS has a very good method of handling updates, STEP-NC is just getting started on its first

round of updates, and EIA-274-D and BCL appear to have no currently active group.

In the DMIS system, a web site is open for Standard Improvement Requests (SIRs) from anyone [3]. Each request is logged and proceeds through several status states until consideration is complete. The web site shows the disposition of all of the hundreds of SIRs proposed since 1996. Once a SIR is entered in the system, anyone can submit a statement for or against the suggested change. No formal system will force the group in control of a standard be open to suggestions for change. In the case of DMIS, the group (now the DMIS Standards Committee) has been open to change and appears to treat all suggestions fairly. Other groups for other standards are often said to be less open and fair.

I. Are there intellectual property issues that may make using the standard impossible or expensive in the future?

Intellectual property right problems related to standards are not unusual. Potential users of a standard should look out for existing and foreseeable problems.

It is possible to get intellectual property rights (patents and copyrights) related to standards. If rights are granted, the owner may try to prevent others from using a standard or try to charge a fee for using it. A ploy the owner might use is to allow inexpensive usage at first and later, once the user has a substantial investment in using the standard, increase the fees. With patents, the owners rights are likely to be unclear, so that users may become involved in expensive litigation.

In 1993, U. S. Patent 5198990 was issued in which a claim was allowed for executing DMIS directly on a control system. The point of having a control language is to execute it, the idea of doing so directly is completely obvious, and patenting the obvious is not supposed to occur. Thus, it is disheartening that the U.S. Patent Office allowed the claim. The effect of the patent is said to have been that no one implemented DMIS for a few years. Eventually, it is said, an agreement was reached that the patent rights would not be used, and DMIS came into common use.

In 2004, U.S. Patent 6795749 was issued for a method of using ISO 14649. It is possible that this may have a chilling effect on the implementation of ISO 14649.

J. Is there a critical mass of conforming implementations of the standard? Does it appear there will be a critical mass in the future?

There need to be enough systems on each side of the interface that useful work can be done.

With EIA-274-D the very notion of conformance fails because the standard is so ambiguous. There are dozens of dialects of the language. Most computer aided manufacturing (CAM) systems have many different post-processors so that they will produce files in most dialects. Thus, it is feasible to use EIA-274-D, but programs are not portable from one machine to another except in some cases when the same company built both controllers. To be fair, EIA-274-D apparently never intended to support interoperability. EIA-274-D is analogous to the concept of "romance language" in that a speaker of one romance language will have a much easier time learning another romance language than will a speaker of Chinese or Swahili.

BCL is, perhaps, the saddest case. BCL is the clearest and least ambiguous of the standard languages for milling machines. Its usefulness, including the portability of programs, was proven by implementations in a few large installations (Rock Island Arsenal, in particular) but there are currently no known commercially available implementations. It seems to have died out. The better mousetrap did not make it in the marketplace.

STEP-NC - Commercially available implementations do not yet exist. There may or may not be a critical mass of implementations in the future.

DMIS - There are said to be several commercially available conforming implementations, but there are also said to be several commercial implementations that purport to implement DMIS but do not conform. There are also commercially available packages that include "DMIS" in their names but are not DMIS and do not claim to be. It is a "buyer beware" situation.

K. Does the standard have conformance classes? Are they part of the standard?

A conformance class is a subset of the specifications of a standard that is approved in some way for some type of use. For example, DMIS has prismatic and thin walled conformance classes. A conformance class may be defined by specifying which commands must be implemented and for each command, which parameters must be implemented.

There are at least three reasons to have conformance classes. First, for large languages, implementing the entire language may be beyond the capability of a vendor or the vendor may decide that it is not economically justified. Second, there may be some class of jobs which requires only a subset of the capabilities of the language. Third, there may be some set of machines which share a subset of the capabilities for which the language has commands.

If conformance classes have been defined for a standard but not incorporated in the standard itself, the status of the classes is in doubt (for example, it may not be clear under what circumstances the definition of the classes might change).

EIA-274-D does not define conformance classes.

DMIS defines two main conformance classes (prismatic and thin walled — meaning sheet metal) plus seven addenda for special capabilities such as rotary table and contact scanning. Moreover, there are three levels for each class and addendum. The DMIS conformance classes are not yet part of the standard.

BCL divides its commands into 32 groups called function sets. This is done in the standard. The intended use of the function sets is not described in text, but Appendix F, which suggests what the contents of machining process plans should be, says that a machining process plan should include a list of required function sets.

STEP-NC defines conformance classes for milling in section 5 of ISO 14649-11 and in section 6 of AP 238. These, however, are not the same. ISO 14649-11 defines six conformance classes by first dividing its entities into eight "data sets" (same idea as BCL's function sets) and then saying which combination of data sets must be included in which conformance class. AP 238 section 6 defines four conformance classes by providing a checklist of entities with a column for each class.

L. Is it necessary to follow a set of usage rules additional to the standard in order to build an interoperating implementation?

There may be communities of users of a standard that agree to follow a set of usage rules. In such communities, it is usually expected that if both the standard and the usage rules are followed, implementations will interoperate, but if the usage rules are not followed, implementations will not interoperate even if they conform to the standard. In some cases there is a fee to join the group, and the usage rules are not publicly available. Potential users of a standard should look out for this situation.

Usage rules may be desirable in several circumstances, such as:

• The standard is large and conformance classes have not been defined, so the rules serve to define a de facto conformance class.

• The standard is ambiguous.

It is much more desirable, however, to fix the standard so as to formalize the conformance classes and fix the bugs.

Of the examples, the author is aware of usage rules only in the case of AP 238 testing, and these rules do not seem to be intended to continue in the long run.

M. Are there mechanisms in place (formal or "natural") to insure that implementations conform to the standard?

Where there are many vendors on each side of a data interface and many users on only one side of a data exchange (readers and writers of HTML, for example), there is a "natural" mechanism for insuring that implementations conform. Any product that does not conform will not be used. No company can produce its own non-conforming flavor of HTML and coerce customers into using it.

Machine control languages never have the benefit of natural pressure for conformance. The markets are too small. Also, both sides of a machine control language interface (i.e., the programmer and the machine controller) are usually in the same customer company, so that the writer is able to adjust to whatever the reader expects. The customer rarely is able to insist on conformance to a standard in this situation. Vendors want to be able to claim to use a standard, but also want users to be unable to use products from other vendors. Thus, vendors may claim to implement a standard without actually conforming to the standard.

Thus, for machine control languages, formal procedures for ensuring conformance are needed in order to get conformance.

For machine control languages, conformance tests strict enough to ensure interoperability if passed are extremely difficult and time-consuming to devise. Once devised, they are difficult and time-consuming to apply. The details of this are enough to fill another paper.

The author is aware of no conformance mechanisms for EIA-274-D, and close conformance to the standard appears to be rare or non-existent.

BCL did not seem to have formal conformance mechanisms, but conformance (in the late 1990's) appeared to be excellent for two reasons. First, there was one primary vendor for BCL controllers. Second, the users were mostly large organizations (including the Rock Island Arsenal) with many machining centers who made the same parts many times and wanted to be able to use the same program on different machines.

For many years, DMIS did not have conformance requirements in the standard or conformance tests or conformance testing services. All manner of non-conforming implementations that claimed to use the standard came into existence. Commercial systems that were only generally similar to DMIS were built. Programs called DMIS programs were rarely interoperable between vendors. It became clear that something needed to be done to help achieve interoperability. In 2001, conformance requirements were included in DMIS 4.0. However, no conformance classes, conformance tests, or conformance testing services were defined at that time. Since then, conformance classes have been defined as discussed earlier, and modest conformance tests have been provided [12]. There is still no conformance testing service.

VI. CONCLUSION

This paper has presented thirteen questions the potential user of a machine control language standard might want to ask in order to decide whether to use the standard. Reasons why the questions should be asked have been given. As examples, partial answers to the questions have been provided for four machine control language standards.

REFERENCES

[1] Bray, T., et al., (editors), “Extensible Markup Language (XML) 1.0 Fourth Edition”, World Wide Web Consortium, http://www.w3.org/TR/2006/REC-xml-20060816, 2006.

[2] Consortium for Advanced Manufacturing - International, “Dimensional Measuring Interface Standard Part I, Revision 05.0”, Consortium for Advanced Manufacturing - International, 2004.

[3] Dimensional Metrology Standards Consortium, http:www.dmisstandard.org/content/blogsection/6/55, 2007.

[4] Electronic Industries Association, “EIA Standard EIA-274-D Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines”, EIA, 1979.

[5] International Organization for Standardization, “ISO International Standard 10303-11, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description method: The EXPRESS language reference manual”, International Organization for Standardization, 2003.

[6] International Organization for Standardization, “ISO International Standard 10303-21, Industrial automation systems and integration — Product data representation and exchange — Part 21: Clear text encoding of the exchange structure”, International Organization for Standardization, 2002.

[7] International Organization for Standardization, “ISO International Standard 10303-238, Industrial automation systems and integration — Product data representation and exchange — Part 238: Application protocol: Application interpreted model for computerized numerical controllers”, International Organization for Standardization, 2007.

[8] International Organization for Standardization, “ISO International Standard 14649-10, Industrial automation systems and integration — Physical device control — Data model for computerized numerical controllers — Part 10: General process data, second edition”, International Organization for Standardization, 2004.

[9] International Organization for Standardization, “International Standard ISO 14649-11, Industrial automation systems and integration — Physical device control — Data model for computerized numerical controllers — Part 11: Process data for milling, second edition”, International Organization for Standardization, 2004.

[10] International Organization for Standardization, “International Standard ISO/IEC 14977, Information technology — Syntactic metalanguage — Extended BNF”, International Organization for Standardization, 1996.

[11] Kearney & Trecker Corporation, “Part Programming and Operating Manual, KT/CNC Control Type C”, Pub 687D, Kearney & Trecker, 1979.

[12] National Institute of Standards and Technology, “DMIS Test Suite”, http://www.isd.mel.nist.gov/projects/metrology_interoperability/dmis_test_suite.htm, 2007.

[13] Numerical Control BCL Standards Association, “NCBSA Standard Proposal for EIA 494 C, Basic Control Language (BCL) An ASCII Data Exchange Specification for Computer Numerical Control Manufacturing”, Numerical Control BCL Standards Association, 1996.

[14] Walmsley, P. (editor), “XML Schema Part 0: Primer Second Edition”, World Wide Web Consortium, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028, 2004.

[15] World Wide Web Consortium, http://www.w3.org/XML/Core/#IPR, 2007.