A Knowledge-Based Inspection Workstation

E. Messina, J. Horst, T. Kramer, H. Huang, T. Tsai, and E. Amatucci
National Institute of Standards and Technology
{messina,horst,kramer,huang,tsai,amatucci}@cme.nist.gov

Abstract

We are building an inspection workstation development environment to use as a testbed for understanding what types of knowledge, e.g., data, algorithms, and processes, can increase the productivity of inspection operations. Inspection can be more efficient through reducing the need for fixturing, integrating the generation of process plans and their execution within the controller, and reducing the errors or data losses that occur by translating the models to different formats. Initial configuration of inspection systems can be less costly through the use of open architectures that are constructed from components. Key elements of our work include in situ feature-based planning, vision-driven part pose estimation, and software methods to facilitate construction of manufacturing controllers. These provide a rich environment in which to study the categories of knowledge that are useful in intelligent control of inspection workstations. This paper describes our vision, approach, and preliminary results.

1. Introduction

Manufacturers are subject to ever increasing profitability and quality requirements. This leads them to seek out greater flexibility and capability in their manufacturing equipment and systems. These requirements have been documented under the Integrated Manufacturing Technology Roadmapping (IMTR) Initiative, which gathered inputs from industry and government to determine the most pressing technology needs and associated barriers that cut across multiple sectors. The IMTR reports [8] [9] listed as high priority technical needs intelligent control systems that are based on open, modular architectures, have model-based process control, and function as components within the larger enterprise information systems.

We are building an inspection testbed in order to study the knowledge aspects required to address the manufacturers' stated needs. In our context, knowledge refers to the information, data, processes, and other capabilities that are encoded or used by a controller and enable it to achieve its goals. We have chosen an inspection workstation (IWS) as our principal testbed for investigating the knowledge requirements for intelligent controllers for several reasons. Inspection is a costly, yet critical function in any manufacturing operation. We seek to help reduce the costs of inspection setup and execution. Inspection controllers today are closed, black box systems, which are difficult and costly to enhance or customize. If manufacturing systems are to become more capable and autonomous, they have to be able to sense and respond to their environment. We are developing a component-based approach to building controllers that can lead to open architecture frameworks that can be more easily configured, enhanced, and maintained. Inspection provides a sensory-rich environment for our experiments. We will briefly describe our inspection testbed, and the Real-Time Control System (RCS) reference architecture, on which it is based. We then present the feature-based process planning and fixtureless inspection capabilities of the testbed. The software engineering aspects of the testbed are then described, followed by the knowledge aspects of the testbed. We conclude with some observations about the approach and future directions.

2. The Inspection Workstation

The inspection workstation consists of a Cordax[‡] three-axis coordinate measurement machine (CMM) with a touch-trigger probe. A downward-looking camera is mounted on the arm of the CMM. A typical usage scenario would be the following:

1. The CMM operator puts the part to be measured onto the work surface without fixturing it. The part's solid model is downloaded to the controller. The model is in terms of manufacturing features, such as holes and pockets.
2. Manufacturing features from the solid model of the part are used by the Feature-Based Inspection and

[‡] **Disclaimer:** Certain commercial products are identified in this article to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by NIST, nor does it imply the materials or equipment identified are necessarily the best for the purpose..

Control System (FBICS) to generate plans for the necessary setups and inspection procedures. The FBICS process will be described below. The final output of the process is a series of Dimensional Measuring Interface Standard (DMIS) files (described below), which are stored on the controller.

3. The operator initiates the inspection through the user interface by instructing the IWS to compute the position of the part on the measurement table. The CMM moves the arm to a position that enables the camera to view the measurement table and the part thereon. An intensity image of the part is captured. The image is processed in order to extract 2D edge features, which are compared with 2D features extracted from a synthetic image of the solid model. The edge features from the two sources are compared, through a Pose Estimation and Part Recognition (PEPR) algorithm, and an estimate of the part's location is produced. The results are displayed to the operator, who confirms that the location has been correctly computed.
4. The location computed by PEPR is for coarse localization. For more accurate positioning, the probe is used to measure two or more known features on the part. The part location is determined from the feature locations.
5. The actual part coordinates are used by the FBICS system to adjust the inspection plans. The DMIS files are sequentially interpreted and executed. Each DMIS file corresponds to a feature that will be inspected. Measurement results are captured in a DMIS output file. DMIS is a language that provides commands for measuring higher-level geometric entities, such as radii of circles, directions of straight edges, and planarity of surfaces. The CMM itself may only be able to measure discrete points. Hence a method must be available to convert the desired measurement (e.g. radius of a bore hole) to a series of measurement points using the probe. Conversely, the measured set of points must be used to compute the resulting higher-level measurement, which can then be compared to the desired value for quality decisions.
6. The measurement results are analyzed and compared to the desired dimensions. An analysis program flags out-of-tolerance conditions. Problem areas may be highlighted graphically, as shown in Figure 1. Inspection results are overlaid onto VRML models with visual cues, such as deviation lines, to illustrate where and how the part is out of tolerance. Inspection results are collected by a product information database that captures statistical trends and can be used to predict the need for interventions in the manufacturing cells. For example, it can determine the maintenance intervals or flag excessive tool wear based on the inspection results.

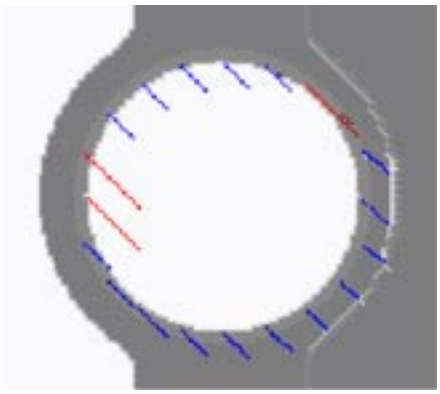

Figure 1: Visualization of Inspection Results

The scenario just described provides several advantages. The current state of the practice in inspection requires numerous steps that cost time and resources, and can lead to errors through sequences of interpretations and transformations of data and intentions. Inspection data and visualization may be used by both manufacturing engineers and design engineers to exchange data for better Design for Manufacture (DFM). Some of the steps and potential errors that are eliminated or reduced are:

- Design and production of fixturing for the particular part to be inspected.
- Fixturing of the part by the operator.
- Creation of the inspection program, through a series of "teach" motions on the equipment or through manual selection of points on a graphical representation of the part.
- Errors introduced by conversion of model geometry between different systems and representations as data has to be transferred to various software tools, for example, from the original CAD system where it was designed to a dimensioning and tolerancing package, or to a tool for creating the DMIS program.
- Clear and quick visual references for inspection technicians to understand where parts are out of tolerance or are within tolerance. With this information, the manufacturers may be able to identify tool wear trends, weaknesses in machine tool operation, and deficiencies in the initial design for manufacturability. The visual cues may be simple CAD models with whiskers or shaded areas corresponding to tolerance data with out of tolerance areas highlighted to quickly warn of problems. Whiskers are lines representing visually the distance and direction from the measured point to the desired, nominal point on the part.

3. Software Architecture

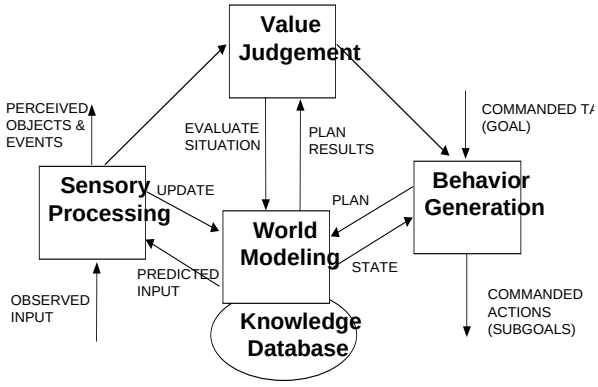

Figure 2: An RCS control node

The controller software is built following the Real-Time Control System (RCS) reference architecture [1] [2]. Developed over the course of two decades at NIST and elsewhere, RCS provides engineering guidelines for the design and implementation of intelligent manufacturing control systems. RCS is geared towards the problem domain where machines are designed to do useful work while employing *in situ* knowledge (from sensors) and *a priori* knowledge, tactics and strategy. These intelligent machines utilize feedback from the physical environment to adapt and modify their plans in order to accomplish the given goals. Based on control theory principles, RCS partitions the problem of control into four basic elements that together comprise a *control node*: behavior generation (BG), sensory processing (SP), world modeling (WM), and value judgement (VJ). Figure 2 shows the RCS control node and the connections between its constituent components. The BG module plans and executes actions to be performed by the equipment. The SP module gathers input from the world. The WM element contains the dynamic and transient data that is computed by the system as it processes its sensory input and estimates its state and the state of the world. World modeling also comprises the longer-term, more stable information contained in external knowledge bases, such as CAD models. The VJ module is used to evaluate the "goodness" of results that would be outcomes of plans generated by BG. BG may generate several tentative alternative plans, which are evaluated by VJ in order to select the best one.

Each individual control node receives commands, generates commanded actions (actuation) and receives inputs from the external world. Each self-contained node is a microcosm of a closed loop control system, producing command signals to what it considers actuators and receiving feedback from what it considers sensors. The actuators and sensors may be virtual, as described in [3]. Thus, actuation commands can be high-level, for example, "Inspect pocket feature." The sensory feedback may have been processed through various levels of abstraction and may take the form of a set of geometric features. Control nodes containing these fundamental capabilities are assembled into a system, following a process of task and temporal decomposition. For more details on the methodology see [14] and [7]. [4] describes the RCS version used in this project, which is referred to as the Intelligent Systems Architecture for Manufacturing (ISAM).

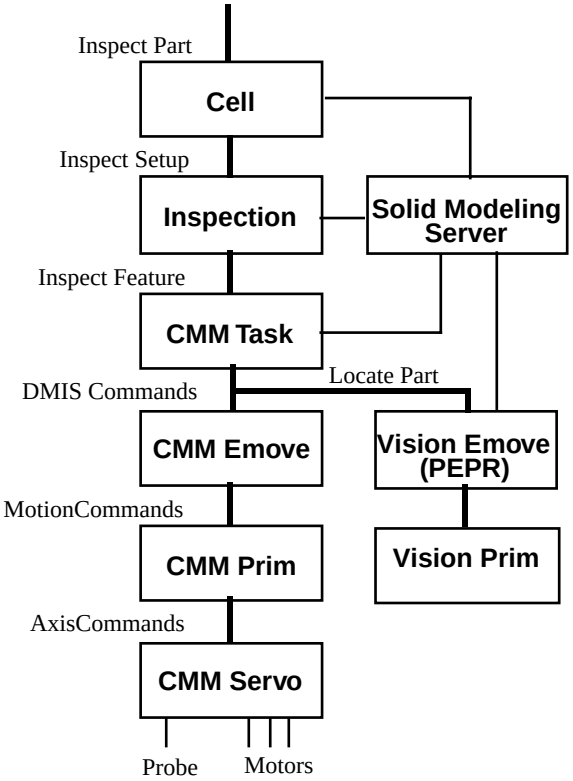

Figure 3: The inspection workstation control hierarchy

Guidelines for system design include organizing the control hierarchy around tasks that the system is to perform in a top-down manner and the physical equipment in a bottom-up manner. The levels of the hierarchy are determined by following the rule of thumb for control system stability that states that control loops should be separated by an order of magnitude. The goal is to manage the complexity at each level of the hierarchy. The span of control at a higher level is greater, both in terms of the breadth of the planning space and the time that is to be considered. However, the resolution of the information or knowledge that a higher level has to deal with is coarser.

For example, the Workstation level produces plans for an entire setup, which may take an hour to inspect and comprises at least one entire face of a part. However, it does not have to deal with entities lower than at the feature level. Contrast this with the Task level, which plans for a single feature that may take minutes to inspect. The Task level planner has to take into consideration higher-resolution knowledge, such as edges, vertices, and their parameters and dimensions.

Figure 3 shows a simplified view of the Inspection Workstation controller hierarchy, including some of the commands that are passed between levels. The nodes typically are run as separate processes on processors running Solaris or VxWorks operating systems. The configuration of hardware is flexible, with nodes able to be distributed across heterogeneous platforms.

4. Feature-Based Inspection

The feature-based process planner hierarchically creates a set of inspection plans for a part. Each of the three top-level controllers (Cell, Workstation, and Task) may receive a command from its superior of the form "inspect X" — in the case of the Cell controller, the superior is the system user. At successively lower hierarchical levels, X is a smaller item, but more such commands are received. To carry out "inspect X," each of the three controllers makes a plan for itself and executes that plan.

The highest level, Cell, receives an entire part design as its input with a command to "inspect the part". The Cell level planner analyzes the geometry of the part and determines the number of setups that are required, creating a process plan consisting of a number of run-setup steps. In executing this process plan, the Cell controller sends the Inspection Workstation controller a series of "inspect setup" commands.

For each such command, the Inspection Workstation controller analyzes the individual features that are to be inspected in that setup and generates a process plan consisting of a number of inspect feature steps. In executing this plan, the Inspection Workstation controller sends the CMM Task controller a number of "inspect feature" commands.

For each such command, the CMM Task controller analyzes the feature and generates a process plan consisting of various types of inspection steps, such as change a probe or measure a point. The plan is encoded in the DMIS format. A single DMIS file is generated for each feature. This plan is executed by the CMM Task controller using a DMIS interpreter, which generates commands to give to the Elemental Move (Emove) controller. These commands are movement commands for the CMM arm. The lower levels of the hierarchy, Prim and Servo, accomplish the movements and record the coordinates when the probe contacts the part surface. The coordinates are fed back up the chain up to the Task level, where they are used to compute the feature parameter that was being measured. The resulting feature measurement is incorporated into the output DMIS file, to be reviewed by the operator or by an automated program that flags out of tolerance conditions.

The planning language used for the Cell and Inspection Workstation controllers is A Language for Process Specification (ALPS) [5] augmented with types of commands appropriate for those controllers. The language is modeled in EXPRESS [10]. Plan files for those controllers are prepared in the STEP Part 21 format [11], a standard format devised for use with EXPRESS.

5. Pose Estimation

The Pose Estimation and Part Recognition (PEPR) subsystem produces a coarse-resolution estimate of the location of the part on the inspection workstation table. The current implementation supports laminar prismatic parts and is based on an algorithm that performs an efficient search based on a priori ordering of model feature sets into “words.” A 2D set of edge features is extracted from the camera image of the part. The model of the part is used by the solid modeling engine to produce a synthetic 2D image and its corresponding edge features. The model edge features are formed offline into feature sets (words) with attributes (letters). These letters are ordered canonically within each word and then the words are ordered in a dictionary of model feature sets. At runtime, sensed features sets (words) are formed, ordered, and matched to the model dictionary words. Pose estimates are generated and a pose clustering setup completes the algorithm.

6. Software Development

In addition to building an example of RCS-based control by using the RCS architecture for our inspection workstation, we are investigating ways to facilitate the use of RCS in construction of subsystems or entire controllers by other system builders. A corollary benefit to being able to develop a robust and usable software development framework is the potential for software reuse. We are experimenting with a component-based, modular approach that follows the RCS model for control nodes. Software tools that support this development effort include internally developed ones and commercially available systems.

The most basic element of the IWS software engineering environment is an RCS template. An RCS template is defined as a set of C++ classes that define the components that comprise an RCS control node: BG, SP, WM, and VJ. The operations, interconnections, and

timing mechanisms appropriate for each of these modules are built into the templates. For instance, the executor (EX) portion of Behavior Generation takes care of parsing the plan generated by the BG Planner (PL) and manages the communication of portions of this plan to the appropriate subordinate, monitors the feedback from the subordinate and handles exceptions. This IWS testbed implementation combines the SP and WM into a single module and splits out the BG into the PL and EX. SP and WM are combined into a single SPWM template due to computational considerations; there can be large quantities of data flowing from the cameras. The PL and EX have been separated to better support the more detailed and complex interactions within and between them. The templates we are using are based on the ISAM variant of RCS [15].

To build a system, the engineers instantiate the generic modules into their system, populate them with their algorithms, attach the appropriate knowledge bases, and make the connections between the nodes. Although tools are not required in order to follow this process, we have used two different tools in our development. One was developed in-house [16] and was especially built to support an earlier version of RCS, in which the individual modules within the control nodes have not been fully fleshed out and are not separated out. This Java-based tool provides graphical means of defining the hierarchy, automatically creating the command and status connections between nodes. Code is generated from this tool, to which developers add their specific algorithms or data structures. Similarly, we are working with ControlShell, a tool from Real-Time Innovations Inc. aimed at computer-aided control system design. ControlShell is a general-purpose tool, so we were able to use it to build our own BG, SPWM, and VJ template components. These were put into a repository and can be accessed by developers. We were also able to develop and reuse functioning components and interfaces in this environment. This system is graphical and enables the user to design the hierarchy, its interconnections, and to generate code, which is subsequently filled in with the specific algorithms using a text editor.

Another area of investigation is that of component specifications. To have truly open and interoperable control systems, all of the principal computational elements need to be well defined within a framework. This enables incorporation of third-party components that adhere to the specification requirements. RCS provides a framework in which to specify the main processing units, as defined by the templates. The specific algorithms that perform the sensory processing, planning, and value judgement must also be well-defined in order to facilitate being able to plug and play or to reuse existing ones. Previous work has led to a comprehensive set of descriptors for the part pose estimation class of algorithms [6][12]. Given a set of categories, an algorithm's provider can fill them in and publish the specifications. Those interested in employing an available algorithm rather than writing their own can locate an existing component and assess its applicability from the specifications, similarly to how a hardware engineer selects semincondutor chips from a catalog. The ControlShell environment provides a shared component repository with links to HTML files containing component specifications.

7. Knowledge in the Inspection Workstation

There are three main categories of knowledge utilized by the inspection workstation. The categorization of the knowledge is useful in helping evaluate the implementation of the testbed. Understanding how and when knowledge is used, as well as its longevity, can help formalize the definition of the architectural elements. This definition guides the requirements for software development tools and environments. Once the definition is clear, designers and implementers know where they need to add their specific items of interest, be it novel planning algorithms or interfaces to their solid modeling servers. The categories in the IWS and examples within each are described:

1. externally-received (*a priori*) models that are invariant
   - —the part to be measured
   - —the workstation kinematics and dynamics
2. *in situ* world models generated from sensory inputs and processing by the system
   - —intermediate representations of the part or its features for use by planning and evaluation algorithms
   - —point, edge, and surface features extracted from camera images
   - —estimation of the state of the system
   - —feature dimensions computed from measured points
3. procedural knowledge that enables the system to accomplish the tasks it is given
   - —How to localize the part on the work surface
   - —How to generate inspection plans for the given part features
   - —How to perform the inspection instructions
   - —How to perform motion commands
   - —How to stay within the error bounds during execution of any motion or action
   - —How to handle emergency or out-of-range situations

Procedural knowledge is considered part of the knowledge that is added to the RCS templates as an application system is built. Certain types of task-specific knowledge can be captured in Finite State Machines

(FSMs) that are easy to understand, develop, and maintain. Tools such as ControlShell facilitate creation and maintenance of FSMs. Typically, FSMs can be used to control high level behaviors or states of the system. Software that implements algorithmic knowledge performs most of the actual planning, sensory processing, simulation, and execution. FSMs can be used to organize high-level behavior and hide the complex algorithmic code. In our Inspection System, the FSM complexity increases as you go up the hierarchy. There are no FSMs at the servo level, whereas the motion-related FSM at the Emove level has 37 states. The Prim level FSM has 18 states. Generally speaking, algorithmic complexity increases as you go down the hierarchy. One exception is at the servo level: the Prim and Emove trajectory generators are both more complex than the servo level trajectory generator. One rough measure of complexity is number of lines of code. Emove has 190 lines, Prim has 340, and Servo has only 40.

With a knowledge base that is centered on the part, the control system can associate derived and computed data to the part model and its features and attributes. This can organize the knowledge around the end product of the enterprise, i.e., the part that is being manufactured. The relationships between derived data and the parent entity on the part model can be traced and maintained. For example, for some of the sensory processing algorithms, it is useful to know the "parent" feature or geometry that a synthetically generated edge feature was derived from. That is, knowing that a linear edge came from a certain pocket feature, the algorithm can use adjacency information to find nearby or related geometries in order to aid its matching between sensed (vision) features and those generated from the model.

Table 1: Example world model entities and commands per hierarchy level

Hierarchy Level	World Model Entities	Command Vocabulary
Cell	Starting Workpiece, Final Part	Inspect_Part
Workstation	Setup, access volumes	Inspect_Setup
Task	Manufacturing Features (e.g., pockets, holes, bosses)	Inspect_Feature
	surfaces, edges	Locate_Part
EMove	Target Point (x,y,z and tolerance)	Measure_Point
Prim	Way Point	Goto (x,y,z, Vel)

Because of the top-down decomposition of the IWS software, the knowledge has a corresponding hierarchy. At each level, control nodes contain, create, and utilize knowledge that is appropriate to their temporal and spatial scope. The node may need to integrate data from a subordinate level in order to make the information useful for its purposes. Similarly, it must decompose and translate the plans that it generates into the vocabulary and granularity that is appropriate for its subordinate nodes and is within the scope of that node's knowledge and ability. Table 1 contains examples of world model entities and command vocabulary for the top five levels of the hierarchy.

8. Conclusions and Future Directions

We have described the main elements of an inspection testbed being developed to study knowledge issues for intelligent control. Our approach integrates capabilities that work from a single model of the manufactured part and its attributes. The inspection plan is generated at the CMM from the manufacturing features of the part, using a solid modeling server for intermediate geometric calculations. The part's location on the inspection workstation's table is computed using vision and image features computed from a synthetic image of the solid model. The world model and knowledge base are being developed in order to study ways to represent the part, its attributes, manufacturing processes, and sensed data in order to enable more efficient and autonomous inspections and to facilitate the design and implementation of advanced inspection architectures. Our preliminary conclusions lead us to categorize knowledge in terms of its permanence and whether it is procedural or declarative. We also find that it is advantageous to leverage a single model within the entire workstation hierarchy and use it as the anchor for other sorts of knowledge, both permanent and derived.

We plan to evolve our testbed to include more advanced capabilities and further our studies. We will integrate different part pose estimation algorithms and validate our component specifications for this class of algorithms. We are working to develop more robust visualization techniques for inspection results using either commercial packages or custom web-based inspection displays. We plan to integrate vision-guided inspection, which is research being done by another team in the Intelligent Systems Division [13]. This work will extend the use of the sensed and CAD models. Future work will investigate more fully the knowledge aspects of integration with the greater enterprise. For instance, what kind of measurements and data should be transmitted to the shop floor controllers? We ultimately intend to use the results from this testbed to propose open architecture component specifications that would help in the construction of inspection systems. This work can be applied to other intelligent manufacturing control applications.

9. References

[1] Albus, J.S., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 3, pp. 473-509, May/June, 1991.

[2] Albus, J.S., "A Reference Model Architecture for Intelligent Systems Design," In *An Introduction to Intelligent and Autonomous Control*, Antsalakis, P.J., and Passino, K.M. eds., 1993.

[3] Albus, J.S., and Meystel, A., Behavior Generation in Intelligent Systems, *NISTIR 6083*, National Institute of Standards and Technology, Gaithersburg, MD, 1997.

[4] Albus, J.S., "An Intelligent Systems Architecture for Manufacturing," *Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective*, Gaithersburg, MD, October 20-23, 1996.

[5] Catron, B., Ray, S. R., "ALPS — A Language for Process Specification", *International Journal of Computer Integrated Manufacturing*, Vol. 4, No. 2, pp 105 -113, 1991.

[6] Horst, J. A., et al., "Precise Definition of Software Component Specifications," *Proceedings of the 7th Symposium on Computer-Aided Control System Design (CACSD '97)*, Gent, Belgium, April 28-30, 1997.

[7] Huang, H.M., et al., "Intelligent System Control: A Unified Approach and Applications," Chapter in *Gordon and Breach International Series in Engineering, Technology and Applied Science, Volumes on "Expert Systems Techniques and Applications,"* To be published in 1999.

[8] Integrated Manufacturing Technology Roadmapping Initiative, *"Intelligent Controls for Discrete Manufacturing Draft Roadmap,"* Oak Ridge, TN, December,1998.

[9] Integrated Manufacturing Technology Roadmapping Initiative, *"Manufacturing Processes and Equipment Draft Roadmap,"* Oak Ridge, TN, December, 1998.

[10] ISO 10303-11:1994, *"Industrial automation systems and integration - Product data representation and exchange - Part 11: The EXPRESS Language Reference Manual"*, ISO, Geneva, Switzerland, 1994.

[11] ISO 10303-21:1994, *"Industrial automation systems and integration - Product data representation and exchange - Part 21: Clear Text Encoding of the Exchange Structure"*, ISO, Geneva, Switzerland, 1994.

[12] Messina, E., et al., "Component Specifications for Robotics Integration," *Autonomous Robots*, Vol. 6, No. 3, June 1999.

[13] Nashman, M. et al., "A Unique Sensor Fusion System for Coordinate Measuring Machine Tasks," *Proceedings of the SPIE International Symposium on Intelligent Systems and Advanced Manufacturing*, Vol. 3209, Pittsburgh, PA, October, 1997.

[14] Quintero, R. and Barbera, A. J., "A Real-Time Control System Methodology for Developing Intelligent Control Systems," *NISTIR 4936*, National Institute of Standards and Technology, Gaithersburg, MD, 1992.

[15] Scott, H.A., "The Inspection Workstation-based Testbed Application for the Intelligent Systems Architecture for Manufacturing," *Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective*, Gaithersburg, MD, October 20-23, 1996.

[16] Shackleford, W., Proctor, F.M., "JAVA-Based Tools for Development and Diagnosis of Real-Time Control Systems," *Proceedings of the ASME: Computers in Engineering Conference*, Atlanta, GA, September 16-18, 1998.