Software Engineering of Machine Control Systems: An Approach to Lifecycle Economics

Sushil Birla Kang Shin

Real-time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, U.S.A.

Abstract

We describe a domain modeling process and framework for use in lifecycle engineering of software to monitor and control agile machining equipment. We focus on an approach of defining initial requirements in a way that facilitates evolution. The process has been used in an industrial case study.

1 Introduction

Software for high-performance programmable manufacturing equipment has become very costly and difficult to develop, integrate, extend, and maintain. On the one hand, there are tremendous benefits from increasing sophistication, intelligence, and versatility in the functions of computer-integrated control, monitoring, diagnostics and maintenance. On the other hand, development uncertainties, integration costs, and operational risks have become an obstacle to further innovation in real-time control software for manufacturing automation. We explore an approach for agility in software development and integration at the conceptual foundation level, organizing initial knowledge about the manufacturing automation domain in a way that is easier to reuse, extend and integrate incrementally, as and when resources and economics justify enhancement of the capability of the manufacturing system.

Current state of controls software: Currently available commercial programmable controllers, numerical controllers, coordinate-measurement-machine controllers, etc., do not provide adequate agility. In the development of new control software, prior experience is mainly leveraged through people or ad hoc use of code fragments or library code that is specific to target environments and supplier organizations. It is difficult, very costly, and often impossible to integrate computer-controlled subsystems procured from different suppliers into a manufacturing cell (Figure 1). Enhancements and cross-vendor integration in the field are even more difficult. Knowledge for agility in creating innovative and intelligent controls does not exist in an organized, easily deployable, standardizable form.

Manufacturing automation domain model: The critical task of organizing knowledge of the (manufacturing) application domain is known as domain analysis [22], and its work product is a conceptual model of the domain. We have organized knowledge in ontologies of *processes* and *resources*. This organization is in an object-oriented form [4, 23]. In our context, an *object* is an individual identifiable item, unit, or entity—either real or abstract—with a well-defined role in the manufacturing automation domain. Our *model* is an abstraction of pertinent properties and characteristics of manufacturing equipment and its elements. In this approach, we form *classes* of objects or instances having the same features or characteristics. The characteristics of a class were organized in terms of orthogonal attributes or *instance variables* or state variables and *operations* that may be performed on them. Classes were organized in a generalization-specialization hierarchy where the specialized class, i.e., *subclass* is a *kind of* its generic class. For example, one specialization path is: *resource* → *processing-resource*,...→ *processing-equipment*,...→ *controlled-processing-equipment*,...→ .... Controlled-processing-equipment may be a basic-machine, or such peripheral-mechanisms as tool-changers and work-changers, or such auxiliary equipment as hydraulic-power-units and oil-coolers, or global sensors. Thus, the taxonomy accommodates equipment not only for material-processing, but also for associated handling and inspection. We have also used an aggregation-constituent hierarchy which describes a *part of* relationship between classes. For example, one constituent-path is *cell* → *workstation*,...→ *basic-machine*,...→ *axis-group*,...→ *joint*,...→ *rotational-joint-pair*....

Issues: There are several economic and technical issues in the creation and maintenance of a domain model. The initial cost [24] to create such a reusable resource is much higher than the cost of a single-use solution. There are many uncertainties in amortizing this cost over (future) applications that are not fully specified at the beginning. As machines become more easily configurable for agility, the

IEEE International Conference on Robotics and Automation
0-7803-1965-6/95 $4.00 ©1995 IEEE

quantity of units over which specific software can be amortized will reduce. To further add to the complexity of the economics, the process of defining the "right abstractions" for the "right domains" is iterative by its very nature [4, 22]. We considered two approaches to deal with these difficulties. The first approach was to search for ways to bound the domain to obtain stability in the requirements of the included application. The second approach was to find a modeling technique that makes evolution easy. Both approaches are open research questions [22]. We have found it easier to pursue a combined approach—mostly bottom-up—in which we incrementally extend the domain.

Relation to prior work: Economic benefits from reusing software assets have only been validated through common experience in a few non-real-time applications where the domains have been very narrowly defined [15,22], e.g., application-generators. There is no published empirical proof that the programming technique of systematic software reuse reduces program development time, duration, cost, skill-requirements, or defect-density on any practical-scale project [8, 10, 11, 21]. To make software evolution easier, Dijkstra [9] and Parnas [18] recommended that any particular program be developed as though it is a member of a family of potential programs that share some common properties, facilitated through appropriate abstraction of these commonalities. The concept of program families evolved into the notion that reusable assets focused on a well-defined domain, in the context of a domain-specific architecture, show more promise in reducing development time [2, 6, 22]. The *Synthesis Process* [6] is a domain-oriented approach that encompasses all the work products of software development. The Software Productivity Consortium reports [25] that the process is at an exploratory level, i.e., it is immature and incompletely developed, recommended only for pilot and low-risk projects, under the guidance of specialists in this process. In a recent pilot project [7], the *Synthesis Process* was applied to a subdomain of a command-and-control application once only. The pilot project was declared a success, although no metrics were collected. The object orientation (OO) paradigm facilitated Parnas' concept of abstracting commonalities across program families [19]. There is strong analytical support and wide intuitive belief that the OO paradigm facilitates reuse and evolution of software, especially the higher-level work products [5]. However, economics issues, reusability, and extensibility of real-time software have not been studied systematically—related work in that field has focused on formal specification methods. Our work focuses on the software economics issues. We have defined a framework and domain analysis process (Section 2) and tested it in an industrial case study. The organization of domain knowledge for the production and maintenance of real-time control software is novel.

Organization of paper: In Section 2, we describe the approach followed in this study. In Section 3, we describe how we bound the domain and its subdomains in an extensible manner. In turn, it becomes appropriate to define a reference architecture, as done in Section 4 by gleaning, adapting and synthesizing from existing architectural models. In Section 5, we define subdomains for each class of components in manufacturing equipment. In Section 6, we describe how we model subsystems and the equipment-system as compositions of their constituents. In Section 8, we summarize the contributions and direction of further research. The whole process is reported more completely in [3].

Conceptual framework: Since models are only approximations of reality, they can be dependable and useful only within some context. Our conceptual framework (Figure 2) defines that context in a layered arrangement. First, the abstractions have to be directed to some defined purpose—the outermost or global part of the framework. Different software engineering methods (the second layer in Figure 2) exist for capturing and organizing these abstractions. Unfortunately, the respective models are not easily inter-convertible. We chose the object-oriented method, since it is emerging as the most comprehensive and unifying method across the fields of databases, artificial intelligence, and software engineering. The object model also provides us independence from the implementation level of programming languages. The domain-definition and control architecture layers in our context-setting framework are treated in Sections 3 and 4.

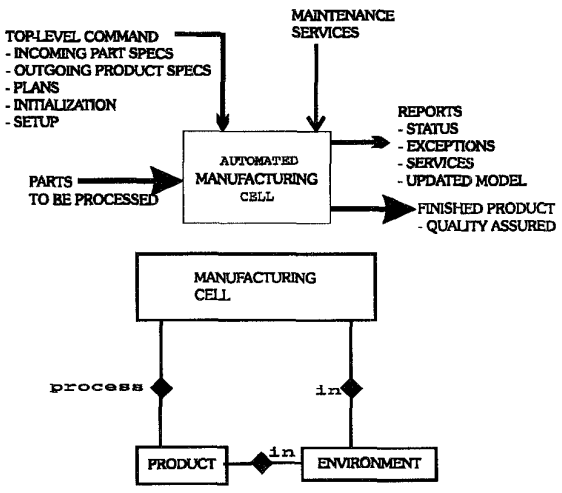

Figure 1: A manufacturing cell

2 Domain modeling process

We devised the following procedure for analyzing the domain of programmably controlled servo-actuated manufac-

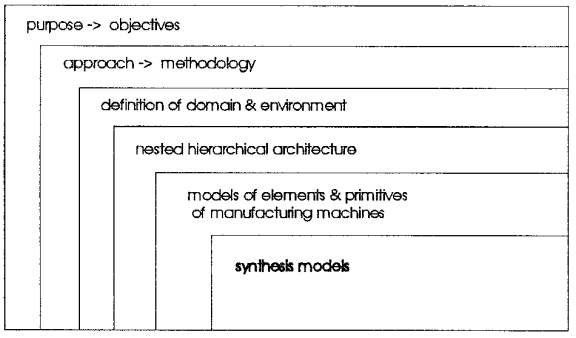

Figure 2: Conceptual framework for cell model

turing equipment, assimilating and adapting ideas from different sources [12, 20].

Procedure for modeling equipment elements:

S1: Consider common equipment configurations.
S2: Identify their functions.
S3: Identify classes of physical components from which these functions can be composed.
S4: Identify elemental functions from which these functions can be composed.
S5: Identify the conceptual primitives from which these elemental functions can be composed.
S6: Model the conceptual primitives.
S7: Model common patterns of inter-relationships.
S8: Build models of component functions and classes successively from the conceptual primitives, employing generalized relationships at each level.

Steps **S3**—**S8** yield classes of objects (Section 5), from which many more machines can be composed than the number in the initial collection. The process will be incremental and iterative [4].

Approach taken for extensibility:

S1: Envision scope of extensions early.
S2: Conceive a top-level architectural model over this scope.
S3: Selectively start modeling well-known concepts "bottom up".
S4: Consult domain experts for additional concepts.
S5: Take chances in foundation being larger than necessary initially.

The cost of **S5** is low relative to cost of discovering inadequacy later. We believe that the conceptual framework, thus established, will reduce the disadvantages of iterative and incremental growth, namely, difficulty of maintaining data integrity and the conceptual integrity of the whole system.

3 Bounding the domain

We are systematically and incrementally approximating the definition of an ontology and semantics [2] for the domain. We analyzed the case of equipment for machining automotive cylinder blocks, cylinder heads, transmission cases, and other parts that fit within the requirements envelope of these parts. However, as these products evolve and their manufacturing processes change, we want our equipment models to be reusable and extensible at minimum life-cycle-cost. It requires a balance between too much and too little initial engineering. Prolonged initial engineering increases the initial investment, delays the start of benefits, delays the start of validation (increasing risk), and increases exposure to changes. Inadequate initial engineering may reduce reusability, especially if future iterations are not performed by the original engineering team. We approached the tradeoff by starting with *readily available* manufacturing engineering knowledge and organizing it in an extensible framework.

3.1 Scope of equipment

Figure 1 depicts the scope in terms of a cell that performs a family of processes on a family of products in some given environment, meeting specified goals of quality, quantity, timing, and cost. By family, we mean that the domain of potential members is well-defined and bounded, although the specific members may not be known ahead of time. To facilitate quick, inexpensive, easy extension, i.e., agility, we organized specification knowledge about products, processes, environment, and equipment in orthogonal dimensions. We bounded the system by bounding the scope in these three dimensions [3]. We characterized the equipment in terms of design intent parameters and configurations, e.g., the property of linearity within the designed operating range. We limited kinematic configurations to non-redundant mechanisms that can be modeled in terms of the ISO standard for representing kinematics [13]. Our scope is limited to machines which will provide an unambiguous correspondence between a monitored or controlled function and a constituent unit. We use this constraint to advantage in architectural simplification, as shown in Section 4.

3.2 Scope of monitoring and control

The purpose is limited to manufacturing parts safely, correctly, and productively and to stop operation, if the equipment is not able to provide this service. Producitivity goals include minimizing the execution time, and the parasitic losses, e.g., consequences of failures.

Task hierarchy: Physical processes are organized in a task hierarchy corresponding to the levels in the control architecture (Section 4). We decomposed control system tasks into the following *monitoring, control, and cognitive tasks*, from which a wide range of procedures can be composed.

Monitoring tasks:

1. Acquire value of some sensed variable.
2. Collect a prescribed time-history of such values.
3. Reduce this time history to some meaningful parameter, using a prescribed procedure, which may use equipment models.
4. Compute the expected value of the sensed or derived parameter, possibly using equipment models.
5. Compare the sensed value or the derived parameter with its expected value.
6. Compare the deviation with allowable limit.
7. Trigger prescribed action upon reaching the limit.
8. Store intermediate computational results as prescribed for later use, possibly for further reduction, possibly using equipment models.

Control tasks: They build on monitoring tasks:

1. Acquire value of some controlled variable or parameter from prescribed plan of execution (typically decomposed from a program for processing workpieces).
2. Decompose or transform this acquired value to values of variables or parameters to be controlled by execution agents. These transformations would use equipment models.
3. Distribute or transfer the values to these execution agents. The ultimate resulting values are set as outputs to some controlled actuators.

Cognitive tasks: Here we describe less structured monitoring and control tasks that acquire knowledge, or refine it. One type of cognitive tasks is *machine learning*. Our scope of machine learning is limited to the fitting of parameter values in previously prescribed models, using prescribed model-fitting procedures. The purpose is to support the tasks of monitoring, control, prognostics, preventive maintenance, diagnostics, corrective maintenance, and enhancement or engineering improvements. The data for machine learning may come from operational data or from controlled calibration tests. Our domain model provides the needed structures. Modeled causal laws also allow estimation of the reaction time needed for each physical function to be serviced and the response time needed by the physical serving mechanism. The domain model provides an organization for tracking these estimates. Similarly, we support learning about perception. Most of the time the perception is not at the point of interest, but at some remote location. Thus, feedback is correspondingly distorted and contains systemic errors, uncertainty and noise of measurement, in addition to similar deviations from the monitored process. Established causal laws and quantitative information do not allow clear isolation of these factors.

4 Architectural model

The domain knowledge is organized around architectural structures, which, in turn, are also a reusable resource. By architecture, we mean the structure(s)—fixed for some time-period—from which specific applications can be composed with minimum additional engineering time and cost. We resort to a reference model [1] architecture, for the application domain, for further organization and partitioning of information in the system to improve timeliness, effectiveness, and computational efficiency. The architecture is based on deploying hierarchies of generalization-specialization, constituents or aggregation, tasks, control, and resolution-relevance (spatial, temporal) [14]. The spatial span and resolution at the innermost nested level have the smallest values. Spatial span increases and resolution gets coarser at each successive outer level. Figure 3 shows the overall information architecture.

Timing: Each level up to level **m** performs an execution cycle within a guaranteed time limit, i.e., these levels are *real-time* subsystems. Efficient and timely communication is made possible by organizing world models into objects corresponding to the levels—information is kept closest to where it is most needed. For example, servo-sensor information is most detailed at level 1, and as we go upward it is successively reduced or abstracted.

Component models for different contexts: The model provides for abstraction of monitored information into the levels in the control architecture shown in Figure 3. The domain model provides for specialization of the attributes within the same system, depending on the context of usage. For example, even though the dynamics-model may be complicated over the whole operating range, it can often be simplified over the narrow range in which a servo-loop is operating in any particular context. The nested hierarchical architecture sets up these contexts. A higher level in the hierarchy may give a different simplified model at different times to different lower levels nested within it. The proposed architecture provides for categorizing goals [16] which help determine which models and which approximations and simplifications are appropriate when in pursuit of a goal of a particular category. This modeling approach, including the nested hierarchy, provides scalability to cover different cost-performance tradeoff aspects, versatility to accommodate different kinds of devices, and extensibility beyond the initially known applications. The resulting knowledge architecture provides a foundation for incorporating artificial intelligence in real time.

5 Modeling equipment elements

Our primary purpose of modeling is to represent real-world machines in a way that captures their characteristics essential to the applications (Section 3). Secondly, we want to

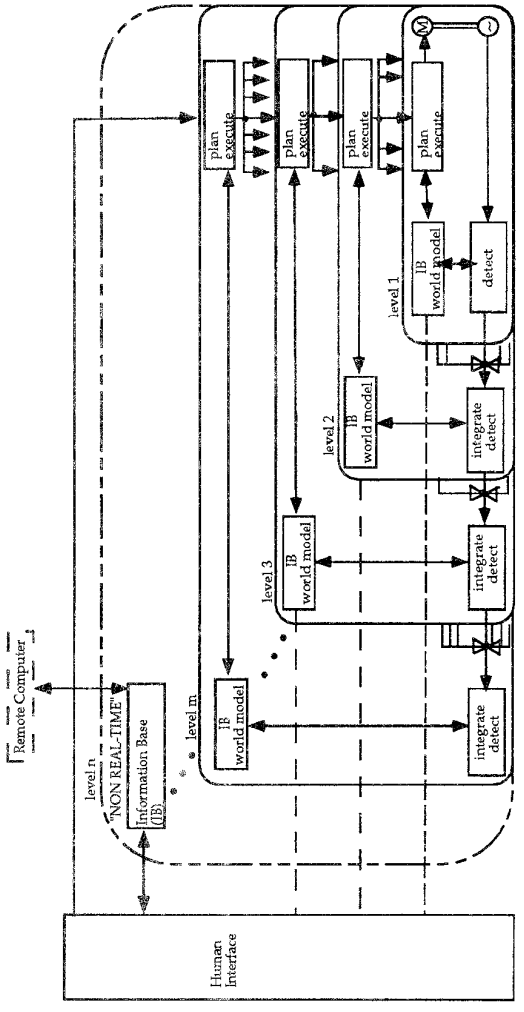

Figure 3: Control hierarchy for a cell.

represent this knowledge in a way that maximizes reusability.

We abstract a single joint or axis of motion as an aggregation of joint-pair (revolute or prismatic), drivetrain, drive, and feedback-sensor. We abstract their common properties in a generalization or superclass *component*. The generality of the approach includes abstractions of input and output, building from conceptual primitives at the level of basic physical quantities. To continue the example of hierarchies introduced in Section 1, class *drive* is modeled as an aggregation of classes *actuator* and *power-regulator*. An example specialization path is: *actuator* → *electrical-actuator*,... → *d.c.-brush-type servo-motor*,... → *rotary d.c.-brush-type servo-motor*.... The structure makes provision for a wide range of actuators, e.g., an air cylinder, through different specialization branches. Similarly, the power-regulator may be a pulse-width-modulated electronic amplifier or an fluid-supply on-off control valve.

Modeling joint dynamics: A joint is modeled as a network of nominally linear system components. The composed property of interest is the dynamics between the input at an actuator and the output motion at some reference point on the joint. We use well-known mathematical transformations to derive the dynamics model for joints of the type used in production machine tools. However, we can improve accuracy of the information, knowing the operating point.

Usage and performance history: The generic component model includes attributes that are useful in prognosis, preventive maintenance, diagnosis, and future specification and design for maintainability. It captures usage history factors affecting life-consumption that are easily available during operation, e.g., duration for which the monitored element is subject to some load, speed, or temperature. However, typical controllers of manufacturing machinery do not provide on-line facilities to capture this data. Typically, separate maintenance-management systems are installed to monitor the equipment, but without the benefit of the crucial operational data, e.g., the time history of load, speed, and temperature to which the component is subjected. The model also has attributes concerning component life, degradation, and life-consumption, so that application software can update this information, based on the usage history factors captured in the model. Other application software can use the computed results. Our model also provides for capturing and tracking dynamic model parameters, so that monitoring software can detect deviations from designed operational limits, e.g., significant nonlinearity.

6 Modeling machines

Ideally, previously-modeled knowledge about components should be reusable in obtaining models of their compositions, e.g., joints. However, the state of the art is far from this ideal. Often properties of an assembly are modeled as a simple union of the properties of its constituents. This approach is not adequate for modeling a servo-controlled joint, axis-group, or machine. The assembly may contain new properties and conditions that do not exist in its components. For example, take a 3-axis machine where each axis individually has a certain range of travel. However, when the three axes are considered together, certain positions of one axis may prevent interference-free travel of another axis. More such interferences arise as a result of other attachments to the machine (e.g., fixture, tool). Our model provides for a more general specification of constraints to be defined explicitly by the user.

Modeling machine kinematics: Our kinematic model is based on an ISO standard [13]. We have extended it to include connectivity of axis-groups or substructures to pro-

vide for the inclusion of kinematic models for fixtures, workpieces, and tooling, and to to include kinematic errors of motion. The composed property of interest is the motion of the work-point as a result of motions of the joints (or vice versa). The kinematics-model of a machine is a composition of the models of the joints in the specified connectivity-order. From relatively few building blocks, a much larger number of combinations can be generated. A model of a machine organized in this manner can be applied to lathes, milling machines, drilling machines, machining centers, grinders, coordinate measurement machines, and robotic mechanisms.

7 Implementation issues

We described agility and intelligence attributes of automation, envisioning a factory environment of distributed intelligence. Manufacturing cells (Figure 1) will be autonomous entities, with self-sufficient computing resources. The cell-level of the control system (Figure 3) includes an object-oriented information base. New engineering information could be brought through portable media or through a communication network.

Open architecture needed: The economics of our proposed software process depends upon its adoption by a community of its users and their acceptance of a common domain model and architecture. The manufacturing automation community in the U.S.A. has invested significant effort in developing a next generation controller specification for an open systems architecture standard [17]. However, it does not propose abstractions for modeling the kinematics and dynamics of a machine or its elements (Sections 5 and 6). Its abstractions (shown in its domain model) for sensors, axis, and machine are not sufficiently generic. For example, it lacks a machine classification independent of processes. Thus, it is limited in its flexibility and extensibility. Other projects known to us are not defining a domain model and architecture at the level of sensor-servo interactions, nor are these projects providing support for capturing and abstracting sensor-servo interaction history.

8 Conclusion

We have performed a case study of agile machining systems in the automotive industry to develop reusable models of manufacturing equipment. We developed and applied a process for modeling the domain of programmable servo-controlled equipment through this case study. Our process is a novel synthesis of ideas from many sources.

We have proposed a nested hierarchy of model-specializations to suit the purpose and timing-constraints of lower control levels. To our knowledge, such systematic specialization in the same system has not been reported elsewhere.

Future work: The major technical limitation is the iterative and incremental nature of the modeling process. To facilitate rapid iteration and evolution, we are building a testbed for control systems that will enable manufacturing automation researchers to experiment with various configurations, functionality, and performance levels.

Controller performance validation: To validate a servo-control loop after a new configuration is statically created, the technology base exists, design and simulation tools are commercially available, but they have to be integrated. However, there is weakness in the technology base to analyze performance of the control software in a working controller. The technology base for dynamic reconfiguration is also weak. For example, if a model is switched while a machine is running, its effect on control stability has to be checked, but there are no tools to support on-line validation. Our contribution is to make more of the necessary knowledge about the equipment available and accessible when and where it is needed. This information will make it easier to test and validate control ideas and their implementation in software.

Software process effectiveness: We plan to document effort required and difficulties encountered at each modification to the control system, the software library, and the conceptual model. The effectiveness of our process will be evaluated in terms of effort saved in creating control software for a specific machine and the number of such application-development cycles required to recover the investment in building the domain model.

References

[1] J. S. Albus, "RCS: A reference model architecture for intelligent control," *Computer*, vol. 25, no. 5, pp. 56–59, May 1992.

[2] G. Arango, "From art form to engineering discipline," in *Proceedings of the 5th International Workshop on Software Specifications and Design*, pp. 152–159, 1989.

[3] S. K. Birla, "Conceptual modeling of manufacturing automation," Technical report, University of Michigan, Ann Arbor, Michigan, 1994.

[4] G. Booch, *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, 390 Bridge Parkway, Redwood City, California 94065, 1991.

[5] G. Booch, *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, 390 Bridge Parkway, Redwood City, California 94065, 1993.

[6] G. Campbell, S. Faulk, and D. Weiss, "Introduction to synthesis," Technical Report Intro_Synthesis_Process_90019_N, Software Productivity Consortium, 2214 Rock Hill Road, Herndon, VA 22070, 1990.

[7] J. O. Connor, C. Mansour, J. Turner-Harris, and G. Campbell, "Reuse in command-and-control systems," *IEEE Software*, vol. 11, no. 5, pp. 70–79, Sept 1994.

[8] S. Conte, A. Dunsmore, and V. Shen, *Software engineering metrics and models*, Benjamin Cummings, Reading, Mass, 1986.

[9] E. Dijkstra, O. Dahl, and C. Hoare, "Notes on structured programming," in *Structured Programming*, pp. 1–82, Academic Press, 1972.

[10] W. Frakes, "An empirical framework for software reuse research," Technical Report 9014, Syracuse University CASE Center, 1990. Proceedings of the Third Workshop on Methods and Tools for Reuse.

[11] W. Frakes, "Software reuse empirical studies," in *Software reusability*, chapter 6, Ellis Horwood, 1994.

[12] P. Freeman, editor, *Tutorial: Software Reusability*, IEEE Computer Society Press, Washington, D.C., 1987.

[13] *ISO CD 10303-105 Product Model Data Representation and Exchange: Part 105—Kinematics*, International Standards Organization, july 1991.

[14] C. Isik and A. Meystel, "Pilot level of a hierarchical controller for an unmanned mobile robot," *IEEE Journal of Robotics and Automation*, vol. 4, no. 3, pp. 241–255, 1988.

[15] C. Krueger, "Software reuse," in *acm computing surveys*, S. March, editor, pp. 131–183, Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036, June 1992.

[16] S. Lytinen, "Conceptual dependency and its descendants," *Computer Math. Applic.*, vol. 23, no. 2–5, pp. 51–73, 1992.

[17] *Next Generation Controller Specification for an Open Systems Architecture Standard*, Manufacturing Technology Directorate Wright Laboratory, September 1994. WI-TR-94-8033.

[18] D. L. Parnas, "On the design and development of program families," *IEEE Trans. Software Engineering*, vol. SE-2, pp. 1–9, March 1976.

[19] D. L. Parnas, "Designing software for ease of extension and contraction," *IEEE Trans. Software Engineering*, vol. SE-5, no. 2, pp. 128–138, March 1979.

[20] R. Prieto-Diaz, "Domain analysis for reusability," in *Proceedings of COMPSAC '87*, pp. 23–29. IEEE, 1987.

[21] R. Prieto-Diaz, "Historical overview," in *Software resuability*, chapter 1, Ellis Horwood, 1994.

[22] R. Prieto-Diaz and G. Arango, *Domain Analysis and Software Systems Modeling*, The IEEE Computer Society Press, Los Alamitos, California, 1991.

[23] J. Rumbaugh et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[24] M. A. Simos, "The domain-oriented software life cycle: Towards an extended process model for reusability," in *Proceedings of the Workshop on Software Reusability and Maintainability*. The National Institute of Software Quality and Productivity, October 1987.

[25] *Reuse-driven software process guidebook*, Software Productivity Consortium, 1993.