DESIGNING THE NEXT GENERATION OF ROBOTIC CONTROLLERS

David G. Goldstein, Ph.D.
Computer Science Department
North Carolina Agricultural & Technical State University
Greensboro, NC

Abstract

This paper describes the scenario-based, object-oriented approach used to specify the software architecture of the next generation of robotic controllers. We also discuss how we intend to implement a version of the controller via a multi-agent approach. We also describe our real-time, fault-tolerant, cooperative reasoning tools that we intend to use to facilitate developing the implementation of the controller. We also describe how we intend to interface existing applications and controller components to the tools so that they interact via objects detailed in the controller specification.

Introduction

Problem

The manufacturing industry of the United States has become increasingly less effective as other nations pour money into research. To regain world technological pre-eminance, the Advanced Research Project Agency and the National Center for Manufacturing Sciences have launched a number of programs. One of these, the Next Generation Controller program, attempts to develop a standard for robotic controllers for the post-1995 time frame. This paper will discuss the approach taken during our working for the program while staying within ethical boundaries concerning this crucial research program. The interested, authorized reader can obtain the actual specification document can be obtained from the National Center for Manufacturing Sciences.[1]

Most of the robotic controllers used in America today reflect programming concepts that are decades old. Change is required to remain competitive. However, manufacturers will not invest in equipment unless the return on investment promises to far exceed any risks. Therefore, the characteristics of any controller proposed must satisfy two different general goals: risk reduction and performance.

Robotic controllers must ensure several key items to facilitate risk reduction. First, current manufacturing practice must be supported; manufacturers are loath to shut down a working factory on a promise of efficiency. Second, current equipment must be supported. Finally, numerous existing applications and utilities must be supported. While each of these items will be discussed later in this paper, open systems largely address these concerns.

Innovative robotic controllers must also provide vast performance improvements for acceptance, where "performance" encompasses several aspects. Obviously, controllers almost always have real-time deadlines to meet and so should afford accurate results via efficient computational processes. Controllers should also afford previously unavailable capabilities, better user-interfaces, and promises of more efficient programming. Since product acceptance eventually depends upon economics, either flexibility (expanded product-line or higher quality product) or lower life cycle cost (through lower product development costs) must be offered.

Loss of market share and un-transferred technical innovations prompted the programs adoption of both risk reduction and increased performance. Current practice in the robotic control industry has hardly changed in decades (with the exception of a few, very well financed areas). The programs goals attempted to afford advanced features (such as art-to-part manufacturing) while facilitating a smooth transition from existing equipment and support software to the next generation of machinery.

Program History

This paper will try to explain the approach that was eventually, successfully used in developing a specification for the controller: scenario-based object-oriented analysis. This approach will be the central focus of the paper because traditional approaches failed dismally in attempting to gain enough support for developing a standard. Another key aspect of developing the standard was "cleaning the kitchen"; if too many cooks don't make good broth, then using fewer cooks might help. It did.

Organization of Paper

We first describe the scope of the controller's specification to provide context for the rest of the paper. We then describe the scenario-based methodology and how it can be employed to specify a controller. We then provide a description of the contents of such a specification. We finally analyze the effectiveness of our approach and draw some conclusions concerning the utility of the specification and the effectiveness of the proposed controller.

Scope of Controller Specification

Developing a national standard for robotic controllers necessitates employing a wide brush; painting a picture of robotic use and manufacturing needs in the near future requires comprehensive coverage of the domain while affording a great deal of flexibility. Comprehensive coverage is required to examine manufacturing from the level of controlling motors in actuators to analyzing throughput of factory lines. We try to specify interfaces to almost any type of information that a user of the system might want to analyze or modify.

Comprehensive coverage is also reflected in the very-varied concerns examined by the specification. Physical elements, such as sensors, effectors, payloads, conveyors, tools, and users are considered. Abstract physical elements, such as envelopes, schedules and enterprise expectations should also be considered. Abstract elements, such as those for configuration, are by far the most difficult, essential elements to include; different implementations employ different strategies for configuration, different kinds of elements requiring configuration, and different granularities of configuration. Given that the controller should address needs as diverse as composite baking, precision material removal, and many-axis (> 100) assembly, facilitating comprehensive coverage often required multiple, combinable representations.

Comprehensive coverage also includes covering current practice: in terms of specifying interfaces to existing machinery, programming techniques, and support software. Hence, interfaces to facilitate interacting with programs for solenoids and C++ programs were both supported.

The controller specification also reflected a great deal of flexibility. Interoperability and plug-replaceability are essential in new robotic architectures destined for commercialization. Such flexibility is facilitated by employing existing standards and an open, published architecture. With respect to this endeavor, the obvious standards to employ were EXPRESS and the Product Data Exchange Specification.[2,3]

The Product Data Exchange Specification (PDES) attempts to provide a standard mechanism for describing virtually any object that might be manufactured as a product. Hence, the numerous volumes of the specification are appropriate for addressing a variety of fields. The standard is hierarchical, building upon very primitive concepts such as Cartesian coordinates and measurement units to eventually describe features such as pockets and items such as resistors.

PDES was written in the specification language EXPRESS. EXPRESS is an ISO standard (but currently undergoing revision for its next version). EXPRESS is well-suited for describing object-oriented concepts (see Object-Oriented Methodology), since it supports both inheritance and abstract data types.

Inheritance is the notion that items described at a higher level in a hierarchy are subsumed in the structure of objects placed lower in a hierarchy (e.g., all mammals have mammary glands). Abstract data types allow new representations for new items (e.g., a car can be represented as a data type and used in representing a fleet of cars).

An example of EXPRESS code is provided in Figure 1. The first entity (object described) is a representation for a sine-wave. Such a type of line should inherit the characteristics of general Line's and is more abstract than lines described as Cartesian_sine_line and Spherical_sine_line.

A sinusoidal line has several attributes, to describe the phase, amplitude and compression factor of the wave. EXPRESS also facilitates constraining the attributes of an object. This example uses the functions all_in_0_to_2pi and periodic_phenomenon to describe relationships among values of attributes that must be present in valid objects.

The specification of the sine-wave is used in the second entity to describe a series of sinusoidal lines (such as might be generated by a

Fourier transform). The example allows for any positive number of sine curves to be combined to describe a Sine_series_line. The constraint expressed here states that there must be an offset for each curve specified.

Much of the work in the United States with respect to both EXPRESS and PDES stems from the National Institute for Standards and Technologies (NIST). The ISO acceptance of EXPRESS as a standard has prompted a large number of tool vendors to also support EXPRESS. Similarly, a great deal of funded research employs PDES to encourage its acceptance. NIST also has developed a public domain toolkit for manipulating EXPRESS models to facilitate working with PDES.

```
ENTITY Sine_line
  SUBTYPE OF (Line)
  SUPERTYPE OF (ONEOF(Cartesian_sine_line, Spherical_sine_line)) ;
    (*
    An individual line described in terms of
    Ampliture*sin(PeriodCompression*Angle+Phase)
    *)
     Compression_coefficient : REAL;
     Phase : REAL;
     Ampliture : REAL;
    WHERE
    all_in_0_to_2pi(Profile_element.Phase);
    periodic_phenomenon(Sine_line);
     END_ENTITY;

ENTITY Sine_series_line;
    (*
    Represents a function as a summation of sines of the form:
    f(w) = C1*sin(A1w+B1) + C2*sin(A2w+B2) + C3*sin(A3w+B3) + ....
    Offsets (t0, t1, ...) are also provided so that relative calculations can be performed, as in
    f(w) = C1*sin(A1(w-t0)+B1) + C2*sin(A2(w-t0)+B2) ...
    *)
    Offset: LIST[1:#] OF REAL;
    Component: LIST[1:#] OF REAL;
   WHERE
    SIZEOF(Offset) = SIZEOF(Component);
END_ENTITY;
```

Figure 1: Sample EXPRESS Model

Object-Oriented Methodology

Our goal in developing the robotic controller specification was to provide a set of standard interfaces by which various applications and equipment could communicate. We provided this interface by describing the set of objects that could be transmitted among applications and support software. We also specified the constraints placed upon these objects to ensure their validity. We also described some minimal performance requirements required of various classes of applications when generating and manipulating these standard objects.

The object-oriented methodology employed and developed novel concepts in Object-Oriented Analysis (OOA) and Design (OOD). Because public domain tools support automated translation of EXPRESS code into C++ (arguably the most popular language considered to be "object oriented"), Object-Oriented Programming (OOP) is also supported.

OOA strives to discern the essential objects for describing a domain. OOD attempts to organize and describe these objects, their behavior, and their interactions. We employed a specific object-oriented technique, Scenario-based Object-Oriented Analysis and Design, to derive the controller architecture. These techniques are particularly good at clearly expressing concepts in a domain and at providing an audit trail to the source of the

original object. To improve the clarity and utility of the our work, we employed Computer-Assisted Software Engineering (CASE) tools to express the products of our analysis and design.

Terminology and Procedures

Object-Oriented Software Engineering is a relatively new approach for developing software. The approach treats instances of data types as *objects*. The data types themselves are typically called *classes*. Walt Disney's favorite car, "Herbie", would be considered an object of the class "car".

Classes have attributes describing characteristics of objects. These attributes are assigned values to reflect the characteristics of particular objects. Hence, the color attribute of the class "car" with respect to the object "Herbie" might have the value "white".

Object-oriented techniques provide numerous benefits, a description of which would be beyond the scope of this paper. However, two of these advantages previously mentioned (abstract data types and inheritance) facilitate developing hierarchies of objects.

The *Composition Hierarchy* pictured in Figure 2 was described via EXPRESS LIST's in Figure 1; composition hierarchies express how multiple objects can be combined to describe more complex objects. Machines are excellent examples of composition hierarchies: a complex machine including tools, spindles, links, etc., can be succinctly expressed in a composition hierarchy.

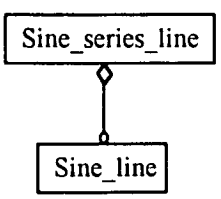

Figure 2: Composition Hierarchy

The *Generalization Hierarchy* expressed in Figure 3 was described via the SUPERTYPE and SUBTYPE relationships as part of the EXPRESS code in Figure 1. Generalization Hierarchies facilitate less abstract classes inheriting attributes from more abstract classes. A clearer example might be machines: Abbrassive_waterjet would be less abstract than Material_removal_machine that would be less abstract than Making_machine, which would be less abstract than Machine. Abbrassive_waterjet's inherit attributed from Material_removal_machine that inherit attributes from Making_machines that inherit from Machines.

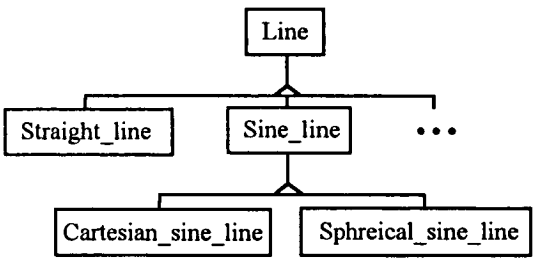

Figure 3: Generalization Hierarchy

Scenario-based Techniques

We employ *scenarios* to ascertain the various objects, their behavior, and their interactions in a given domain. Scenarios are timelines of events with respect to one or more objects. One can imagine a scenario as a movie depicting the existence of an object at some level of granularity.

Determining the level of granularity for examining an object is essential and often non-trivial. A machine can be described by its shape, or the shapes of its components, or by the shape of pieces of the components (such as screws on a jig).

Granularity is complicated by the fact that many aspects of an object might require descriptions in different measurement units and at multiple levels of abstraction. For example, if a planning application is going to reason about the behavior of a machine, it must know the machines' capabilities. The planner must know what the machine can do and with what precision. To ascertain the interactions among objects affecting the machine's precision, we may have to examine both a more precise granularity of composed objects (e.g., interactions of surfaces of the jig and tool) and a more precise granularity of time (e.g., microseconds as opposed to seconds). Hence, we use scenarios describing the same objects, but encompassing different time granularities.

Scenario-based analysis is particularly effective because it directly maps elements in the domain to models in the interface. Scenario-based analysis builds customer confidence

because he has a better concept of the mechanisms underlying any "black boxes". Scenario-based analysis also facilitates traceability; because classes originate from particular scenarios, questions that a customer might have can quickly and efficiently be addressed. We tend to involve the customers as much as possible in the analysis phase, hopefully obtaining the actual scenarios from them via interviewing techniques.

Implementing a Controller

We hope to implement a version of the specification using a multi-agent approach. Recent literature in artificial intelligence suggests that collections of simple agents are much easier to control than large, monolithic programs.

We intend to treat applications and portions of the implemented controller as intelligent agents (Figure 4). Many of these agents, e.g., planners, will be represented as knowledge-based applications. Other applications, e.g., machine executives, will be embedded in wrappers to communicate standard objects from the specification.

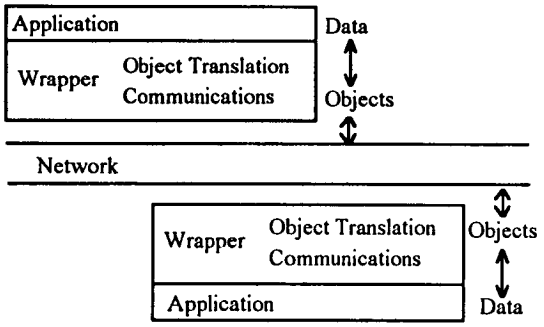

Figure 4: Controller Components as Agents

Crucial to our implementation of a controller will be the Distributed Artificial Intelligence Toolkit (DAIT).[4,5] DAIT provides distributed knowledge-based processing while affording transparent processor fault-tolerance. DAIT also includes predicates to facilitate real-time control. DAIT is based-upon NASA's C Language Integrated Production System.[6] DAIT includes tools for metering, configuration, interprocess communication, and user-interfaces. DAIT supports forward-chaining reasoning, procedural programming, functional programming, object-oriented programming, and deductive database queries. We expect to implement both Fuzzy inferencing and backward-chaining in the near future.

Conclusions

Object-oriented analysis and design provide an attractive mechanism for examining the domain of robotic control. Scenario-based software engineering techniques can often be used to achieve consensus among multiple customers concerning requirements. The Next Generation Controller program successfully used these techniques in developing a specification for standard robotic controllers in the U.S.

We hope to soon implement a version of the controller by employing multi-agent techniques. We will use internally developed tools specifically designed for cooperative knowledge-based processing in this endeavor. By embedding existing software in wrappers communicating objects found in the specification, we will facilitate interoperability and interchangeability of various components of our controller.

The next generation of robotic controllers must be innovative enough to support avant-garde research concepts such as art-to-part manufacturing. However, this specification of these new controllers will also have to address the needs of supporting current hardware and software. We feel that the specification adequately addresses these concerns. We also hope to soon realize a controller demonstrating interoperability and interchangeability of components while offering a very high degree of functionality.

References

[1] NATIONAL CENTER FOR MANUFACTURING SCIENCES,

[2] INTERNATIONAL STANDARDS ORGANIZATION, `Product Data Exchange Specification First Working Draft', ISO TC184/ SC4/WG1, Document N284, 1988.

[3] SPIBY, P.: `EXPRESS Language Reference Manual', Document Number N14, International Standards Organization, April, 1991.

[4] GOLDSTEIN, D.: `The Distributed Artificial Intelligence Toolkit', *AI Expert* (Miller-Freeman Publishing), San Francisco, January 1994.

[5] GOLDSTEIN, D.: `A Fault-tolerant Architecture for Distributed, Heterogeneous, Deductive Knowledge-based Applications', Ph.D. Dissertation, University of Texas at Arlington, Arlington, USA, August 1992.

[6] GIARRANTANO, J.: `CLIPS User's Guide' (NASA/JSC), Houston, TX, 1991

Acknowledgments

Aspect of this work were inspired while under contract on the Next Generation Controller program. The author has continued to investigate concepts presented here as part of North Carolina Agricultural & Technical State University's Manufacturing Initiative (internally funded) and Generic Object-Orioented Software Engineering laboratory (partially funded by the Army and IBM).