INTELLIGENT SYSTEMS AND TECHNOLOGIES FOR MANUFACTURING

Patrick J. Eicker, David J. Miller, and David R. Strip

Patrick J. Eicker *is manager of the Computer Sciences Department at Sandia National Laboratories in Albuquerque, New Mexico, and* ***David J. Miller*** *and* ***David R. Strip*** *are senior members of technical staff in that department. Mr. Eicker's department develops intelligent systems and technologies for application to the DOE's manufacturing and hazardous materials handling operations, as well as advanced digital and optical computing technologies. He joined Sandia in 1969 after receiving a Ph.D. in mathematical statistics from Colorado State University (Fort Collins). He has worked in the areas of national-security- and energy-system analysis, and on solar-energy technologies. Mr. Miller's primary responsibilities as a computer scientist are software development and system integration for the Intelligent Machine*
(continued on page 22)

Issues as diverse as occupational safety, waste minimization, and product quality are forcing the Department of Energy (DOE) to increase the level of automation in its defense-production facilities. In this paper, we describe the development necessary to allow a specific automation technology—the standard industrial robot—to be used effectively in DOE manufacturing operations. We describe Sandia's research and development efforts on automated planning and programming, and on programming models and languages for "intelligent" manufacturing systems that use robots. We conclude by discussing the application of these concepts to machines and processes other than robots. We also mention how these technologies affect the more general manufacturing-technology base.

Introduction

Because of the convergence of many events, the U.S. Department of Energy's defense design and production complex is facing a period of great change. Today's shrinking defense budgets mandate that the DOE downsize this complex (without loss of capabilities), while Eastern Europe and the Soviet Union are reaching a new steady state of interaction with the West. In addition, standards that limit the exposure of production personnel to hazards in the workplace continue to become more stringent. Finally, environmental concerns require the development of benign production processes that generate smaller quantities of waste.

The DOE's approach to these issues consists of many components. One element of the DOE strategy is to increase the breadth and depth of penetration of automated production processes.

Sandia National Laboratories is heavily involved in the process of adapting standard, industrial-automation technologies to the specialized needs of the DOE's government-owned, contractor-operated production complex. This paper describes the research and development

Panel 1. Abbreviations, Acronyms, and Terms

AML — a manufacturing language; developed by IBM

AML/X — a manufacturing language for design and manufacturing; developed by IBM

ANS — American Nuclear Society

ANSI — American National Standards Institute

AUTOPASS — automatic programming system for computer controlled mechanical assembly; developed by IBM

C++ — an objected-oriented descendant of the C programming language; developed by AT&T Bell Laboratories

CAD — computer-aided design

DDCMP — Digital Equipment Corporation's proprietary digital data communications message protocol

DF1 — the American National Standards Institute protocol defined by ANSI publication X3.28–1976

DOE — U.S. Department of Energy

infeasibility — an invalid assembly sequence or assembly operation

LAMA — language for automatic mechanical assembly; a system developed by Tomas Lozano-Pérez at Massachusetts Institute of Technology

NASA — National Aeronautics and Space Administration

RCCL — Robot Control C Library; developed at the University of Pennsylvania and McGill University; currently maintained by McGill University

RIPE — robot-independent programming environment

RIPL — robot-independent programming language

VME — Versa Module European, a computer-bus standard

required because of the DOE's small-lot production environment and focuses on the work being done related to the application of industrial robots. (*Small lot* means that, typically, only a few hundred units are built over a span of many years.)

Standard Industrial Robots. Industrial robots are well suited to mass production. Once the robots have been programmed and provided with fixtures, production runs of tens or hundreds of thousands of parts are common. From an electromechanical standpoint, standard industrial robots can adequately execute a range of important DOE production operations, such as light machining, electronic and mechanical assembly, and welding. But because the DOE production environment involves small lots, methods must be developed to reduce the nonrecurring expense of developing programs and fixtures that often adds prohibitive costs to the overall production operation.

In the current state of the art, robot programs are developed in the following way:

1. Set up the workcell with a robot, fixtures, and piece parts.
2. Move the robot, typically by using a button box (more properly called a *teach pendant*), to each location where the robot is to perform an operation.
3. Collect the robot-joint-encoder readings for each point and store them in memory.
4. Write a program that links the points and operations (e.g., *move to point B, open gripper*).

While this approach is satisfactory for mass production, it does not meet the needs of small-lot production. It is like writing a new program for every new set of data in standard computer applications.

Intelligent Robot Systems. Sandia's approach to robot systems is to eliminate the entire manual process of teaching and programming. Instead, we develop algorithmic and software techniques that allow robot systems to generate their own plans and programs. Conceptually, this involves linking a domain-expert system for a particular type of production operation with a geometric model of the specific product. (A *domain-expert system* is equipped with "knowledge" and "rules" databases that enable it to interpret data and decide how to execute a specific task.)

As a specific example of automatic planning and programming, this paper uses a mechanical-assembly expert system that is coupled with computer-aided-design (CAD) models of the assemblies. This concept is being applied successfully at Sandia to a host of other operations, which are summarized in the conclusion to this paper.

The addition of sensor and model-based con-

Panel 2. Intelligent Manufacturing Systems

If the Department of Energy is to achieve its goal of reducing the exposure of workers to hazards, future production systems must exhibit a range of advanced behaviors. The following behavior can be expected of intelligent manufacturing systems that use computers, sensors, and software models of the working environment:

- The system can *automatically plan and program*:
  - — Trajectories and paths
  - — Machine instructions
  - — Optimum speeds
  - — Safe or "stay out" zones
  - — Product-inspection instructions
  - — Workspace reconfiguration
  - — Production of fixtures and molds.
- The system can *operate in less structured workspaces* because it can:
  - — Eliminate precision fixtures and jigs
  - — Accommodate variability from prior processing
  - — Compensate for tool wear and position error
  - — Accommodate workplace change
  - — Recover from errors.

trol allows us to reduce or eliminate the requirement for precise fixtures and jigs for the piece parts. In our mechanical-assembly example, the automatic planning and programming system develops computer-vision algorithms to "find" the piece parts in the workspace, and develops force-control algorithms to execute the assembly operations. These types of real-time sensory capabilities permit flexible production systems to have other desirable features—such as automated compensation for tool wear, in-process inspection, and automated error detection and correction.

In summary, these systems are able to generate plans of action, program the necessary devices, and then use sensors and instrumentation to react appropriately to changes in the workspace. We use *intelligent systems* and *intelligent machines* as short, descriptive nomenclature for these systems. Panel 2 identifies the behaviors to be expected from intelligent manufacturing systems.

Programming Intelligent Systems. In addition to discussing mechanical assembly as an example of automatic planning and programming, we also describe the programming environment used at Sandia to develop software for these intelligent systems. A common architecture, programming model, and programming language have been developed.

Intelligent systems integrate heterogeneous groups of functional elements: machines, sensors, real-time computers, communications, workpieces, and so on. From post-mortem analyses of the software developed for our earliest systems, we found that many algorithms and functions were being programmed again and again. For example, different applications often demand different robots, and robots from different manufacturers have different programming languages. When subprograms are common from application to application (and many are), the software must be rewritten completely to accommodate the language difference. This is an expensive proposition and is not limited just to the robotic parts of intelligent systems.

We developed the robot-independent programming environment (RIPE) and robot-independent programming language (RIPL) as an integrated solution to these issues. We will discuss the RIPE/RIPL system shortly.

Automated Planning and Programming

One of Sandia's projects on automated planning and programming for mechanical assembly is the Archimedes project.[1] Its goal is to provide the capability for automatically generating robot programs that do mechanical assemblies.

The Archimedes system uses the designer's CAD drawings as input and, from them, must deduce the feasible (or optimal) assembly orders (i.e., sequences) and part-mating operations. It then translates this information into a program that can be executed by a robotic workcell.

This task differs significantly from what appears to be a similar challenge in the assembly of electronics components, such as printed-circuit boards. Generally, electronics assembly is a problem that has two degrees of freedom, compared to the six degrees of freedom in the mechanical-assembly problem.

Furthermore, an electronics assembly's function has only limited relation to its physical design, which greatly simplifies one's ability to design easy-to-assemble products. Mechanical components typically display a much wider variety of shapes and require a greater variety of assembly operations.

Technical Description. The Archimedes approach is to accept a purely geometric CAD model as the description of the assembly. An alternative approach adopted by many others[2] assumes that the CAD model supplements the geometry with "features" or other information. Most often these features describe how parts mate. Although this alternative approach does simplify the challenge of building an automatic programming system, it also:

- Places an additional burden on the designer.
- Rules out application of the technique to existing designs that do not have the features incorporated in the model.
- Precludes new manufacturing techniques that are not supported by the features that become part of a design.

The Archimedes system executes the automated planning phase recursively to determine possible assembly orders. At each stage, geometric algorithms examine the current structure and identify the part or parts that are kinematically free to be removed. These parts are marked with the label of the current stage and deleted from the model. The procedure is then invoked recursively on the remaining parts until no parts are left. From the part labels, we build a minimal graph that describes the assembly precedence among parts. Any sequence that satisfies the relationships in this graph represents a kinematically feasible assembly order. The number of such sequences tends to grow quickly with the number of components in the assembly.

For this reason, other researchers have focused their attention on identifying subassemblies, i.e., groups of parts that may be assembled and then treated as a single unit.[3] It has been our experience that mechanical designers typically provide an adequate specification of subassemblies (often based on functional criteria or inspection requirements), so that automatic determination of subassemblies is not necessary.

A particular assembly order may not be feasible because of constraints other than kinematics. For example, if one part is supported by another, the supporting part may have to be assembled first to provide stability, even though kinematically the parts are unordered. The Archimedes framework uses a combination of a rule-based expert system and novel algorithmic techniques to identify invalid assembly orders (known as *infeasibilities*). As a result, the possible orders are pruned until we are left with only those that are feasible.

During this phase, the expert system also identifies the appropriate assembly operation (e.g., insert, mate, align) for joining the new part with the assembly. Some of the infeasibilities found in this phase of assembly planning can be eliminated by changes to either the design or the environment. For example, one part might not be mechanically stable until a subsequent part is added to the assembly. This infeasibility can sometimes be resolved by using fixtures.

The Archimedes framework includes rules and algorithms that recognize the need for fixtures and provide their design. This increases the set of feasible assembly orders, which is sometimes necessary if we are to find at least one feasible sequence. The increased choice of feasible orders is also beneficial when optimization of the sequence is a goal.

At the completion of the automatic planning phase, the output is a generic assembly plan; i.e., the sequence of operations on assembly parts, plus designs for fixtures. No existing robot controller could ever directly execute this *part-assembly plan* (although the

plan could serve as written instructions to a human assembler).

The automatic programming phase converts this plan into a program for the specific devices in the workcell. Archimedes contains a set of *skeletons*, or macros, for each assembly operation. These skeletons are program fragments written in the language of a specific robot and describe, in a generic way, how to execute a specific operation. The skeletons have blanks that are to be filled with data about the specific use, such as the coordinates of a pickup location or the weight of a part.

Archimedes instantiates a particular skeleton for each operation in the generic plan; i.e., it chooses the appropriate skeleton for each operation in sequence and begins to fill in the skeleton. It then expands these skeletons, or macros, using external information; i.e., information from the CAD model of the part, as well as from other sources, such as a CAD model of the robot's working environment or a path planner. The resulting expansion of the skeleton is a set of executable instructions in the robot's native programming language and a specification for the workcell layout.

Because flexible assembly often requires the use of sensors (for example, computer vision or force), a capability exists to include skeleton sensing programs as part of the skeletons for the assembly operations. Thus, an action skeleton may in turn contain a computer-vision skeleton, whose specifics about size and shape are automatically added at compilation time, based on the CAD model.

By separating the Archimedes system into a planning phase (which is generic) and a programming phase (which is task-specific), we have given the system great versatility. We can easily retarget the output from one workcell to another by replacing the robot-language-specific parts of the skeletons. We can even replace the robot-language skeleton with one written for a robot simulator. This allows us to carry out a simulation and watch an animation of the assembly process before parts are made, which makes Archimedes part of the design process and not just a manufacturing tool.

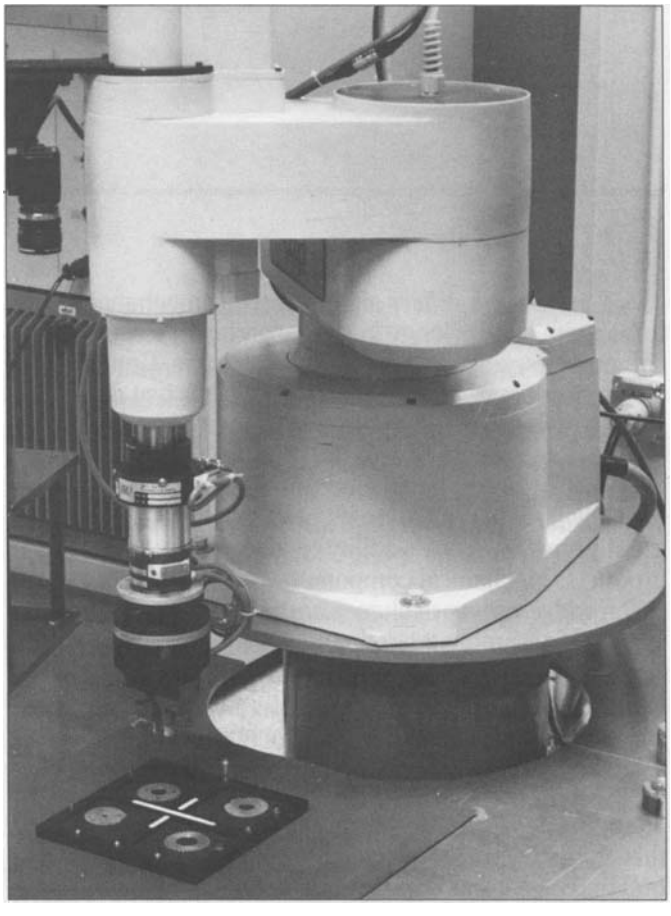


Figure 1. Sandia's automated-mechanical-assembly research platform on which Archimedes concepts are tested. These concepts have been demonstrated on unidirectional assemblies that are typical of a wide variety of assemblies built in the defense, automotive, aircraft, and appliance industries.

A truly flexible workcell will require constant reconfiguration to meet the demands of a broad product mix. The workcell itself is a mechanical assembly with a CAD model that is created, in part, through the compilation of the part-assembly plan. We can apply Archimedes to this CAD model and derive a plan for assembling and, with simple extensions, disassembling the workcell. This approach permits the use of the workcell in a wide variety of applications.

Archimedes has several precursors in its goal to simplify robot programming. We can trace task-level specification of programs and the use of skeletons at

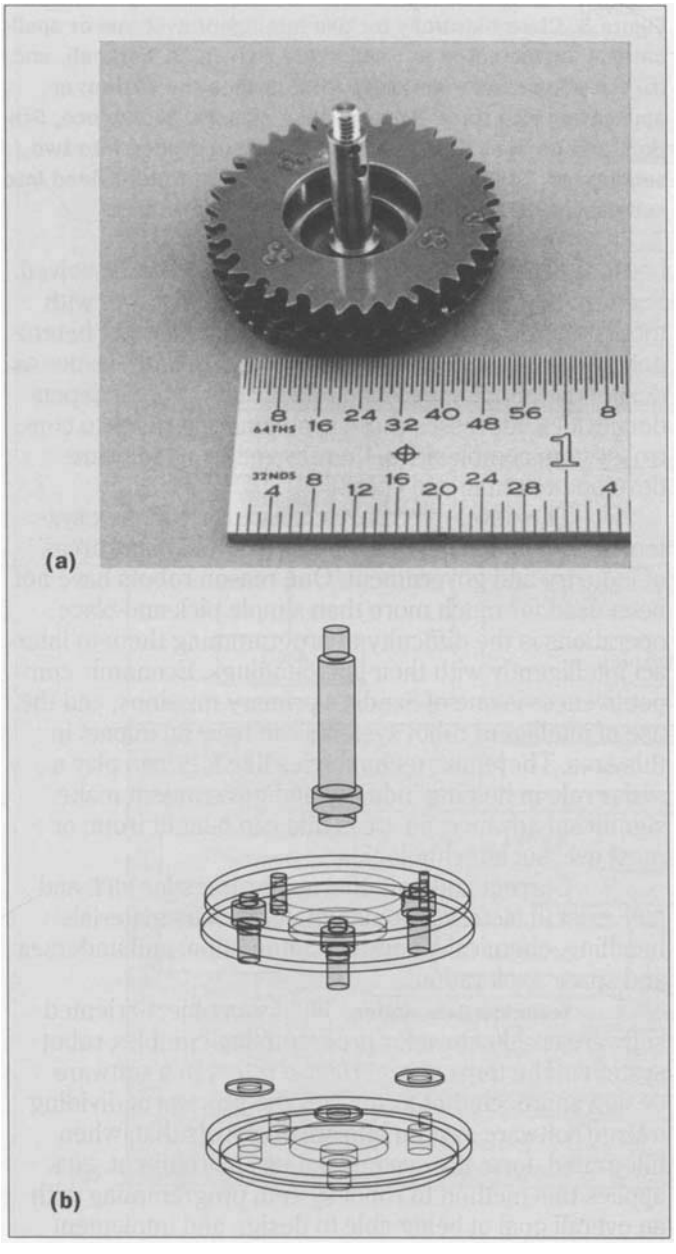


Figure 2. The prototype Archimedes system was used to plan the assembly operations, fixtures, and sensing programs for assembly of a Sandia-designed strong-link. (a) Pattern wheel of the strong-link. (b) Exploded diagram generated automatically by the Archimedes system.

least as far back as Lozano-Pérez's LAMA system.[4] Although this system contained several important ideas, it could not be implemented because it was really a conceptual design for a language. Other similar precursors (i.e., their goal was a higher level specification of a robot's assembly program) include IBM's AUTOPASS system [5] and AML language.[6] Several research groups are pursuing work that is similar to Archimedes.[7,8] But to the best of our knowledge, the Archimedes project is unique in that its goal is robot-executable code, which we have achieved for a limited modeling domain.

Prototype System. Figure 1 shows the prototype Archimedes system, which is based on a specialized solid modeler with a limited domain. Laboratory demonstrations of the system have included planning the assembly operations, fixtures, and sensing programs for assembly of a Sandia-designed strong-link. (This safety component is currently built by hand in lots of 5 to 10 units in Class–1000 clean rooms.) Figure 2 shows a photograph of the pattern-wheel subassembly of the strong-link and the exploded diagram generated by the Archimedes system's automatic planning phase.

The Archimedes system can be applied to a much wider variety of assemblies, including many from the automotive, aircraft, and appliance industries. We are continuing to evolve the system to broaden its domain of application.

Future Directions. The system described here is a prototype, proof-of-concept system. The work to date has concentrated on unidirectional assemblies—i.e., those in which all parts are placed in the assembly from a single direction—which covers a broad range of products both within and outside the DOE. Discussions under way with several U.S. manufacturers may result in the develop-

ment and transfer of full-scale industrial systems similar to those for which we have developed prototypes.

The extension of these capabilities to general, three-dimensional assemblies requires advances in the technologies for automated mechanical assembly. The advances required include developments in solid modeling (which underlies CAD), motion planning, and rule-based systems.

Figure 3. Class hierarchy for two intelligent systems or applications implemented at Sandia: (a) cask-head workcell, and (b) radiation-survey workcell. RIPE divides the system or application into three generic, base classes: Workpiece, Station, and Device. The Device class is then divided into two subclasses, Tool and Transport, that are, in turn, divided into subclasses for particular transport devices or tools.

Programming Environments for Intelligent Systems

The development of reliable, reusable software for large-scale, real-time systems using current software technologies is recognized as a key challenge for systems developers.

At Sandia, a robot-independent programming environment and a robot-independent programming language have been designed and implemented for use in building complex, intelligent machine systems.[9] The RIPE/RIPL technology addresses the complexity issues associated with such systems and provides an environment for rapid, cost-effective implementations.

Initially, two motives inspired RIPE's development: Sandia's diverse software needs and the cost of software development.

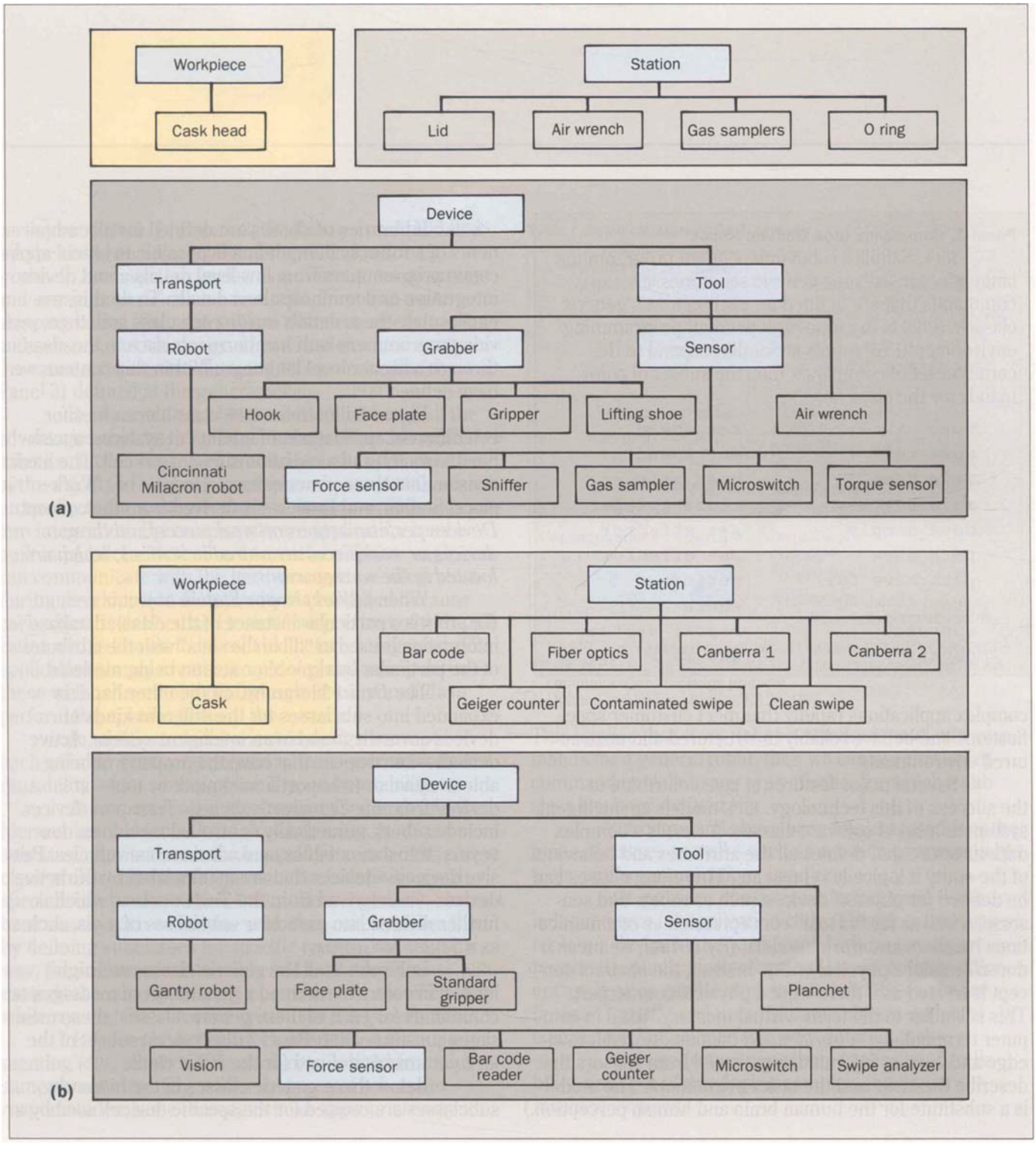
The research and development work done by Sandia's intelligent systems and technologies groups covers many different areas. These groups have built prototype systems for handling hazardous materials, automated assembly, edge finishing, and other applications. Within Sandia's robotics laboratories, the software needs include requirements for many different types of robots, sensors, and other devices, yet the staff has many different levels of programming expertise. We require a software environment that accommodates this diversity. In addition, it should allow the staff to concentrate on intelligent-machine research and development without having to be concerned about low-level programming details and software-system integration.

Software development represents a major part of the overall cost of building intelligent robot systems, because of the complex problems that need to be solved. Such problems include the integration of sensors with motion-control systems; real-time coordination of heterogeneous subsystems; and such typical software issues as design methodologies, code reuse, and device independence. RIPE addresses these problems and helps to control system complexities, thereby reducing software development time and costs.

The development of software for complex systems continues to be a difficult problem in many areas of industry and government. One reason robots have not been used for much more than simple pick-and-place operations is the difficulty of programming them to interact intelligently with their surroundings. Economic competitiveness is one of Sandia's primary missions, and the use of intelligent robot systems can have an impact in this area. Therefore, technologies like RIPE can play a major role in helping industry and government make significant advances in areas that can benefit from, or must use, such technologies.

Current and potential applications for RIPE and RIPL exist in factory automation, hazardous-materials handling, chemical-laboratory automation, and undersea and space exploration.

Technical Description. RIPE is an object-oriented software architecture for programming complex robot systems. The term *object oriented* refers to a software-design approach that formalizes the process of dividing a large software system into small objects that, when integrated, form the system and its environment. RIPE applies this method to robot-system programming with an overall goal of being able to design and implement


Workpiece
Cask head
Station
Lid
Air wrench
Gas samplers
O ring
Device
Transport
Tool
Robot
Grabber
Sensor
Hook
Face plate
Gripper
Lifting shoe
Air wrench
Cincinnati Milacron robot
Force sensor
Sniffer
Gas sampler
Microswitch
Torque sensor
(a)
Workpiece
Cask
Station
Bar code
Fiber optics
Canberra 1
Canberra 2
Geiger counter
Contaminated swipe
Clean swipe
Device
Transport
Tool
Robot
Grabber
Sensor
Gantry robot
Face plate
Standard gripper
Planchet
Vision
Force sensor
Bar code reader
Geiger counter
Microswitch
Swipe analyzer
(b)

Panel 3. Commands for a Generic Robot

RIPL, Sandia's robot-independent programming language, consists of a generic set of messages or commands that are defined for each of RIPE's generic classes. (RIPE is the robot-independent programming environment.) All robots at Sandia respond to this common set of commands (i.e., the subset of commands for the class *Robot*):

```
move                set_speed
move_rel            get_speed
move_home           open_gripper
move_react          close_gripper
move_comply         get_effector
path_move           put_effector
path_move_rel       perform
stop                where
approach            report_status
depart              who_am_i
```


complex applications rapidly that meet customer specifications and behave reliably in structured and unstructured environments.

Several major features in RIPE contribute to the success of this technology. RIPE models an intelligent system as a set of software classes. A *class* is a complex data structure that defines all the attributes and behaviors of the entity it logically represents. Therefore, classes can be defined for physical devices such as robots and sensors, as well as for "virtual" concepts such as communications handlers and world models. (By *virtual*, we mean it doesn't model a physical entity. Instead, the abstract concept is treated as if there were a physical counterpart. This is similar to the term "virtual memory" used in computer terminology. A *world model* contains stored knowledge and current information gathered from sensors that describe the tasks and the task environment. The model is a substitute for the human brain and human perception.)

If libraries of classes are defined for all components of a robot system, then it is possible to shield application programmers from low-level details about device integration or communications details. To do this, we encapsulate these details inside each class and, then, provide programmers with a uniform interface to the class through a higher level language. Within this context, we have defined RIPL.

Figure 3 illustrates RIPE class hierarchies for two different applications of intelligent systems: a caskhead workcell and a radiation-survey workcell. The basic division into three generic, base classes—i.e., Workpiece, Station, and Device—is derived from the concept: *Devices perform actions on workpieces; and these devices or workpieces are stored in stations, which are located in the workspace.*

When a *Workpiece* or *Station* object is created (i.e., this is a particular instance of the class), database information is used to "fill in the slots" with the attributes of the particular workpiece or station being modeled.

The *Device* hierarchy, on the other hand, is expanded into subclasses for the different kinds of devices normally found in an intelligent system. Active devices—i.e., devices that have the property of being able to move or transport a workpiece or tool—are derived from the *Transport* subclass. Transport devices include robots, numerically controlled machines, conveyors, translation tables, and autonomous vehicles. Passive devices—devices that are manipulated by the active devices—are derived from the *Tool* subclass, which is further divided into particular subclasses of tools, such as a *Sensor* or *Grabber*.

In Figure 3, all the generic classes are highlighted in color. We defined a generic set of messages or commands for each of these generic classes; these messages constitute RIPL. Panel 3 illustrates a subset of the RIPE commands defined for the *Robot* class.

Below these generic classes in the hierarchy, subclasses are created for the specific devices used by a

particular system. In Figure 3, the two systems modeled are the cask-head workcell and the radiation-survey workcell. Each system performs a different set of survey and sensing operations on mockups of nuclear-waste shipping casks. The systems use different robots, tools, and sensors, but are designed and implemented using the same software structure and RIPL commands (see Panel 3) defined at the generic levels.

Because RIPE is object-oriented, it shares all the advantages object-oriented technology offers, as applied to robotics. RIPE is organized around a class representation of the objects in a robot's workspace; hence, RIPE's structure reflects the system's physical structure. This aids system integration. Because a software application can be built in parallel with the hardware, the software engineers can communicate with the hardware-system integrators during development. Complexity is controlled because each object is defined and tested independent of the application and is known to be reliable before being used. The application simply creates, combines, and manipulates these well-behaved objects through RIPL commands to perform the specific tasks of the system.

In addition, object-oriented design concepts—such as inheritance and polymorphism—permit software reusability, extensibility, reliability, and portability.

Inheritance is the mechanism used to define the hierarchies of objects through subclassing. It means a subclass inherits the attributes and behaviors of its parent class while extending the parent class's definition with specialized characteristics. RIPE defines a generic, abstract base class—such as Robot—and, then, extrapolates this by defining subclasses for specific types of robots. This way, the software is extensible and reusable because RIPE extends the concept of a generic robot to a specific robot while reusing the software written for the generic robot.

Polymorphism, the mechanism used for implementing RIPL, allows different objects derived from the same parent class (e.g., different types of robots) to respond to the same messages in an appropriate way.

Table I. Some Libraries of RIPE Classes

Primary object	Base class	Subclasses defined
Device	Robot	Cincinnati Milacron model T3/786 Cimcorp gantry Gmfanuc model S–10, with the KAREL® programming environment*
	Sensor	JR3 force sensor LORD force sensor proximity sensors vision sensors
	Tool	torque wrench squeeze controller
Communications handler	Serial	no protocol DDCMP† DF1 protocol‡
	Parallel	Network client and servers for interprocess communications

* KAREL is a registered trademark of Gmfanuc Robotics Corporation.
† DDCMP is Digital Equipment Corporation's proprietary digital data communications message protocol.
‡ Defined in ANSI publication X3.28–1976.

This implies that, if we define a standard set of RIPL commands for a generic robot, then we can use that set of commands to talk to any type of robot for which a subclass has been defined.

RIPL commands have been implemented for other types of classes as well—such as sensors, programmable tools, and communications handlers.

RIPL provides consistent interfaces and device independence for application code. Therefore, one can replace a device in the workspace without having to rewrite the application software. Also, an entirely different application can be implemented that has a similar design structure and uses the same objects and RIPL commands, but will perform very different tasks.

All these factors contribute to the speed, reliability, and cost of developing complex, intelligent machine systems.

Status of RIPE and RIPL. At Sandia, RIPE is currently implemented and used in several application areas including two prototype systems for handling nuclear-waste shipping casks, a system for cleaning underground storage tanks, and a system for glove-box access. (A *glove box* is a sealed, protectively lined compartment that has holes to which gloves are attached for use in handling materials inside the compartment.)

These systems use libraries of RIPE classes that have been defined for the primary objects identified in Table I. These class libraries are implemented in AT&T's C++ programming language and are currently used on UNIX® system-based workstations and VME-based distributed microprocessors for real-time control. (The C++ language is an objected-oriented descendant of the C programming language; both languages were developed at AT&T Bell Laboratories. UNIX is a registered trademark of UNIX System Laboratories, Inc. VME stands for Versa Module European, a computer-bus standard.) We selected C++ because it offers all the features needed for object-oriented programming while still providing portability, real-time control, and compatibility with a large base of existing C code.

In addition to providing the capability to program these systems in C++, we have developed a RIPE client-server interface to a commercially available, graphics and animation-based robot-simulation system.[10] Thus, users have a graphical programming system for task simulation; off-line planning and programming; and safe, on-line control of devices by using the programming system's geometric world models.

The systems currently implemented (i.e., the cask handler, tank cleanup, and glove-box systems) use commercially available computing hardware, operating-system software, and device controllers. Because RIPE is based on "standards"—open systems such as the UNIX system, C++ language, and VME bus—and is not implemented as a proprietary system, developers who use it can put together a system. The environment was specifically designed to be portable and easily extensible for new devices and applications.

Prior Approaches. Before developing RIPE, we carefully surveyed the work that had been done to date, particularly in the area of robot-programming languages. Various approaches had been taken, and we tried to incorporate and expand on their best features and concepts when designing and implementing RIPE.

Vendor-supplied languages, such as Unimation's VAL–II® language and IBM's AML/X language, have evolved into general-purpose computer languages rather than just robot-programming languages.[11,12] However, they are not easily ported to other robots and computing platforms, and are not suitable for real-time, interrupt-driven coordination of multiple subsystems.

The Robot Control C Library (RCCL) is another robot-software system that uses UNIX system-based, C-language libraries of robot commands and sometimes replaces the robot controller to provide an environment for control-algorithm research.[13,14] Because RCCL requires expertise in programming an operating system and knowledge of complex trajectory planning, untrained people find that the system is difficult to use. (RCCL is now in the public domain and is currently maintained at McGill University, Montreal, Quebec.)

Small-scale efforts with the use of other conventional languages, such as the Ada® or Smalltalk® programming language, to program robot systems have been primarily for single-purpose, demonstration systems.[15,16] (Ada is a registered trademark of the U.S. Department of Defense, and Smalltalk is a registered trademark of Xerox Corporation.)

However, the underlying theme in all the efforts we surveyed stresses the need for a standard language that is:

- More powerful, flexible, portable, and readable
- Embedded in an environment that supports interactive development tools and facilities for simulation and graphical programming.

Future Directions. The development of version 1.0 of the RIPE/RIPL system is finished. Version 1.0 is now being documented for application by the DOE and industrial users.

Research and development have begun on enhancements to the system. The most critical areas of concern—error detection and recovery, and software safety—are particularly important for applications in hazardous environments.

In addition, future development work includes expansion of the class libraries to provide control for additional robots, sensors, other devices, and other communications modes. More detailed implementations of the Workpiece and Station classes are needed that define interfaces to CAD systems, simulation modeling systems, and object-oriented databases for storage and retrieval of object states and world-modeling information. A global *WorldModel* class must also be designed and implemented to tie together the information that allows device objects to interact with each other and the physical environment in a more intelligent way.

Finally, we need an icon-based, graphical programming environment that is based on another standard, such as X Window System™ software. (X Window System is a trademark of Massachusetts Institute of Technology.) This will make it easier for nonprogrammers to build applications using RIPE without the added expense and expertise needed for a separate simulation-system environment.

Conclusion

Sandia has demonstrated automated planning and programming concepts in a range of applications that are important to the DOE, and is moving these technologies from the research lab to the factory floor. A common technology base of hardware and software supports applications for both manufacturing and hazardous-materials handling.

Application-specific systems (such as the Archimedes project) use geometric models, physical models, and generic sensing or control software and hardware to plan, program, and implement operations automatically. These concepts have been used for:

- Mechanical assembly
- Robotic edge-finishing of machined and cast parts
- Health physics operations in radioactive waste handling; e.g., surveying for radioactive substances that may have escaped from confinement containers, such as special shipping casks.
- Robotic characterization and remediation (i.e., the removal of or proper containment of waste) of hazardous-waste storage tanks.

In addition, these concepts are being extended to:

- Automation of chemical-analysis laboratories
- Explosives assembly
- Disassembly of hazardous systems
- Mobile robot applications
- Chemical process control
- In-process inspection
- Adaptive machining on computer numerically controlled machines.

The RIPE/RIPL system enhances the speed and reliability with which we are able to develop new applications. Because so much of the hardware and software of intelligent systems can be used in widely differing applications, RIPE and RIPL were developed, in part, to facilitate reuse of existing, tested software. The RIPE/RIPL system was also developed to provide a common programming model for use by application programmers, as well as a common architecture for intelligent systems and a standard language for their implementation.

Transfer of these technologies to users within the DOE community has begun. In addition, cooperative research and development agreements have been developed with U.S. industrial partners. These transfers involve partners who are interested in developing more flexible production capability, or who require real-time, sensor-based control of production operations.

References

1. D. R. Strip and A. A. Maciejewski, "Archimedes: An Experiment in Automating Mechanical Assembly," *Proceedings of the 11th International Conference on Assembly Automation*, Dearborn, Michigan, November 1990, SME Technical Paper MS90–839, Society of Manufacturing Engineers, Detroit, Michigan, November 1990.
2. H. Ko and K. Lee, "Automatic Assembly Procedure Generation from Mating Conditions," *Computer-Aided Design*, Vol. 19, No. 1, January/February 1987, pp. 3–10.
3. L. S. Homem de Mello, "Task Sequence Planning for Robotic Assembly," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1989.
4. T. Lozano-Pérez and P. Winston, "LAMA: A Language for Automatic Mechanical Assembly," *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, August 22–25, 1977, William Kaufmann, Inc., 95 First Street, Los Altos, California 94022, 1977.
5. L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM Journal of Research and Development*, Vol. 24, No. 4, July 1977, pp. 321–333.
6. R. H. Taylor, P. D. Summers, and J. M. Meyer, "AML: A Manufacturing Language," *International Journal of Robotics Research*, Vol. 1, No. 3, Fall 1982, pp. 19–41.
7. A. DelChambre, "A Pragmatic Approach to Computer-Aided Assembly Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, May 13–18, 1990, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 1600–1605.
8. R. S. Mittikali, P. K. Khosla, and Y. Xu, "Subassembly Identification and Motion Generation for Assembly," *Proceedings of the IEEE International Conference on Systems Engineering*, Pittsburgh, Pennsylvania, August 9–11, 1990, IEEE, Piscataway, New Jersey, 1990, pp. 399–403.
9. D. J. Miller and R. C. Lennox, "An Object-Oriented Environment for Robot System Architectures," *Proceedings of the IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, May 13–18, 1990, Vol. 1, IEEE Computer Society Press, Los Alamitos, California, New York, 1990, pp. 352–361.
10. B. Christensen, W. Drotning, and S. Thunborg, "Model Based, Sensor Directed Remediation of Underground Storage Tanks," *Robotics and Remote Systems—Proceedings of the Fourth ANS Topical Meeting*, Albuquerque, New Mexico, February 25–28, 1991, Document No. 574–169/MO7389, U.S. Government Printing Office, Washington, D.C., 1991, pp. 227–237.
11. *User's Guide to VAL–II, Programming Manual*, Version 2.0, Document No. 398AGI, Unimation, Inc., Shelter Rock Lane, Danbury, Connecticut 06810, 1983.
12. L. R. Nackman, M. A. Lavin, R. H. Taylor, W. C. Dietrich, and D. D. Grossman, "AML/X: A Programming Language for Design and Manufacturing," *Proceedings of the 1986 Fall Joint Computer Conference*, Dallas, Texas, November 2–6, 1986, IEEE Computer Society Press, Los Alamitos, California, 1986, pp. 145–159.
13. V. Hayward and R. Paul, "Robot Manipulator Control under UNIX RCCL: A Robot Control 'C' Library," *International Journal of Robotics Research*, Vol. 5, No. 4, Winter 1986, pp. 94–111.
14. P. Backes, S. Hayati, V. Hayward, and K. Tso, "The KALI Multi-arm Robot Programming and Control Environment," *Proceedings of the NASA Conference on Space Telerobotics*, January 31 to February 2, 1989.
15. R. A. Volze and T. N. Mudge, "Robots Are (Nothing More Than) Abstract Data Types," *Robotics Research: The Next Five Years and Beyond*, Society of Manufacturing Engineers, Detroit, Michigan, 1984.
16. W. R. LaLonde, D. A. Thomas, and K. Johnson, "Smalltalk as a Programming Language for Robotics?," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina, March 31 to April 3, 1987, Vol. 3, IEEE Computer Society Press, Los Alamitos, California, pp. 1456–1461.

Biographies (continued)

Systems Division. He started working at Sandia in 1978 after receiving a B.S. in mathematics from Duquesne University (Pittsburgh, Pennsylvania) and an M.S. in computer and information science from The Ohio State University (Columbus). He has worked in robotics for five years. His interests include computer and robot programming languages, reusable software environments, and pattern recognition. Mr. Strip's current research interests are focused on automatic planning of robotic assembly and related issues in geometric modeling and geometric reasoning. He has a B.A. in mathematics from the University of Pennsylvania (Philadelphia) and an M.S. and Ph.D. in operations research from Cornell University (Ithaca, New York). He joined Sandia in 1978.

(Manuscript received May 13, 1991)