Asynchronous Dynamic Replanning in a Multiagent Planning Architecture

David E. Wilkins and Karen L. Myers
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
{myers, wilkins}@al.sri.com

Abstract
We briefly describe the history of ARPI projects at SRI International. Our work on agents situated in dynamic and unpredictable environments is described in more detail. Such agents require several capabilities for successful operation, such as monitoring the world, responding appropriately to important events, accepting goals, synthesizing plans for achieving those goals, and executing the plans while continuing to be responsive to changes in the world. In addition, the agents should be able to replan as changes in the world render their plans obsolete, and to reason about uncertain information. The Cypress system is a domain-independent framework for defining persistent agents with this full range of behavior and has been used for several demanding applications, including military operations, real-time tracking and fault diagnosis. The Cypress technology will be incorporated in the Multiagent Planning Architecture, currently under development, which will provide a new architecture for large planning problems that require the coordinated efforts of distributed human and computer experts.

SRI and ARPI
SRI International (SRI) has been involved in ARPI from its inception. Two SRI projects have already been completed during Phase 2 of ARPI. One, led by Marie Bienkowski, developed SOCAP (System for Operations Crisis Action Planning), using the SiPE-2 planning system as the reasoning engine. SOCAP was the central component of ARPI’s second Integrated Feasibility Demonstration. The second completed project, led by David Wilkins, developed software for creating persistent agents that can achieve complex tasks in dynamic and uncertain environments. This second project is described in this paper. Several Technology Integration Experiments (TIEs) were completed. For example, SiPE-2 has been integrated, at some level, with General Electric’s (GE) temporal reasoning system Tachyon, with GE’s case-based reasoner CAFS, and with Carnegie Mellon University’s (CMU) DITOPS scheduler.

Ongoing projects on guiding planning systems with high-level advice and machine learning of planning knowledge are described elsewhere in this volume. In this paper we briefly summarize a new project to develop a Multiagent Planning Architecture (MPA), and describe the Cypress technology that resulted from Phase 2 of ARPI. The MPA project began in late 1995 and is involved in five TIEs. Cypress provides some of the technology for defining agents in MPA. Our demonstration of Cypress shows multiple planning and execution agents doing runtime asynchronous replanning. In the future, MPA will facilitate the building of similar systems by providing the infrastructure for multiple agents. Cypress had to build its own infrastructure — MPA would make it possible to more easily explore and evaluate alternative architectures for such a system.

Multiagent Planning Architecture
The objective of MPA is to develop a new architecture for large, sophisticated planning problems that require the coordinated efforts of diverse, geographically distributed human and computer experts. MPA will facilitate incorporation of new tools and capabilities as they become available, and will allow planning systems to capitalize on the benefits of distributed computing architectures for efficiency and robustness.

MPA is a layered, agent-based architecture that groups planning agents (PAs), i.e., agents with planning-related capabilities, into planning cells. MPA will provide wrappers and agent libraries that provide support for asynchronicity (e.g., cuing incoming messages, or interrupting execution for priority requests) and interagent communication. MPA intends to have PAs communicate using the common plan representation and the agent communication languages being developed by other ARPI projects, but other alternatives will be explored as well.

One of the defining characteristics of MPA that distinguishes it from some other agent architectures is the ability to compose a planning cell of agents committed to one particular planning process. Planning coordinator agents are a type of meta-PA that hierarchically decompose a planning task and distribute it to the PAs of a planning cell. A planning coordinator may distribute tasks to both PAs and meta-PAs. The planning cell is
Taskable, Reactive Agents

Our Phase 2 research concerned developing persistent agents that can achieve complex tasks in dynamic and uncertain environments. We refer to such agents as taskable, reactive agents. An agent of this type requires a number of capabilities. The ability to execute complex tasks necessitates the use of strategic plans that provide an outline for accomplishing tasks; hence, the agent must be able to synthesize new plans at runtime. The dynamic nature of the environment requires that the agent be able to deal with unpredictable changes in its world. As such, agents must be able to react to unanticipated events by taking appropriate actions in a timely manner, while continuing activities that support current goals. The unpredictability of the world could lead to failure of plans generated for individual tasks. Thus, agents must have the ability to recover from failures by adapting their activities to the new situation, or even replanning should the world change sufficiently. Finally, the agent should be able to perform all of these operations even in the face of uncertainty about the world state.

The notion of a taskable, reactive agent does not correspond to a PA, but rather is an abstract description of the capabilities of a type of high-level agent. In the MPA, such an agent would be implemented as a meta-PA that controlled several different PAs.

Many domains of interest require agents with the capabilities we have described. Military operations provide a good example. Certainly, one would not engage in an undertaking such as Desert Storm without first formulating a strategic mission plan. Reactive response and failure recovery are necessary because unexpected equipment failures, weather conditions, and enemy actions (among others) may require changes to the overall strategic plan. Reasoning in the face of uncertainty is critical, since complete knowledge for a given scenario is unlikely.

Cypress provides a framework in which to create taskable, reactive agents. Several features distinguish our approach: (1) the generation and execution of complex plans with parallel actions, (2) the integration of goal-driven and event-driven activities during execution, (3) the use of evidential reasoning for dealing with uncertainty, and (4) the use of asynchronous replanning to handle runtime execution problems.

Asynchronous replanning contrasts with synchronous replanning, in which plan execution is halted while an alternative plan is generated. Asynchronous replanning is critical in domains such as military operations or robot control, where it is infeasible to halt all execution activities while replanning some portion of the overall plan. Cypress is the first system of which we are aware that supports asynchronous runtime replanning with a general-purpose generative planner. When problems arise during execution of strategic plans, Cypress can invoke a planning agent to produce a new plan while an execution agent continues to execute unaffected portions of the plan. This multiagent replanning/exeuction architecture will eventually operate in MPA, and lessons learned in its development will be incorporated into MPA.

The motivation for Cypress was to build a heuristically adequate system that would be useful in practical applications. To this end, Cypress relies on mature, powerful planning and execution technologies, namely the SIPE-2 generative planner, the PRS-CL execution system, and the Gister-CL system for reasoning about uncertainty.\(^1\) We have applied Cypress to a number of ARPI researchers. The results of this survey will help make MPA be of as much use as possible to the broader ARPI community, and identifies technologies that are sufficiently far along in their development to be considered for inclusion as PAs.

\(^1\)In particular, Cypress = SIPE + PRS. Sipe-2, PRS-CL, Gister-CL, Act-Editor and Cypress are trademarks of SRI International.
of demanding problems, including real-time tracking, fault diagnosis, and military operations (Wilkins & Desimone 1994).

While there have been many efforts to develop the component technologies (planners, replanners, reactive controllers, uncertain reasoners) required for the kind of agent described here, and even to integrate some of these technologies, we are unaware of any current systems that provide the full functionality of Cypress. This paper focuses on the integration of plan generation, execution, and replanning in Cypress. Detailed technical discussions of the subsystems are provided in other documents.

Cypress Abstract Agent Model

We begin by presenting an abstract characterization of our model for taskable, reactive agents, as shown in Figure 1. The model contains two main components, an executor and a planner. The two components share a library of possible actions that the system can take. The library encompasses a full range of action representations, including plans, planning operators, and executable procedures. Together, these are called Acts in Cypress. The executable procedures correspond to predefined standard operating procedures for satisfying individual goals. These three classes of actions span multiple levels of abstraction.

The relationship of this agent model to MPA is that a taskable, reactive agent would be a meta-PA composed of several PAs. For example, in the Cypress demonstration described below, the planner is one agent in its own process, while the executor is implemented as two agents, each with its own process.

The executor is always active, constantly monitoring the world for goals to be achieved or events that require immediate action. In accord with its current beliefs and goals, the executor takes actions in response to these goals and events. Appropriate responses include applying predefined executable procedures stored in the action library, invoking the planner to produce a new plan for achieving a goal, or requesting that the planner modify a previous plan for which problems have developed during execution. The planner should be capable of synthesizing sophisticated action sequences that include parallel actions, conditional actions, and resource assignments.

The planner plans only to a certain level of detail, with the executor taking that plan and expanding it at runtime by applying appropriate library actions at lower levels of abstraction. Planning to the lowest level of detail is often undesirable because of the resultant combinatorics of deep searches. Furthermore, it makes sense to plan down only to abstraction levels at which actions can be reasoned about ahead of time. For example, it is undesirable to plan large military operations down to the most minute detail since many decisions are conditioned on information that is not available until runtime. Rather, it is the responsibility of the executor to further adapt the plan to the actual state of the world during execution. Similarly, it is often undesirable for the execution system to respond to high-level goals without a plan; for instance, a reactive system should not attempt to implement a Desert Storm-sized operation by applying procedures blindly.

There is an additional benefit to having an executor that can take plans at varying levels of abstraction and expand them at runtime, namely, it enables the executor to begin taking actions toward meeting a goal without having to wait for a completely finished plan.

The Cypress System

Cypress constitutes a particular framework in which to define taskable, reactive agents based on the model presented in the previous section. The architecture of Cypress is depicted in Figure 2.

Cypress is built on top of several mature AI systems that have been tested in a number of real-world applications. SIPS-2 (System for Interactive Planning and Execution) is a classical planning system capable of generating plans hierarchically (Wilkins 1988). PRS-CL (the Procedural Reasoning System) is a reactive execution system that integrates goal-oriented and event-driven activity in a flexible, uniform framework (Georgeff & Ingrand 1988). The Gister-CL system implements a suite of evidential reasoning techniques that can be used during both planning and execution to analyze uncertain information about the world and possible actions (Strat & Lowrance 1989). For example, Gister-CL can be used to reason about uncertain information in order to choose among multiple planning operators for a goal by the planner, and to choose among suitable procedures by the executor at runtime.

PRS-CL and SIPS-2 employ their own internal representations for plans and actions. For this reason, Cypress supports the use of an interlingua called the ACT representation (Wilkins & Myers 1985) that en-
Figure 2: The Architecture of Cypress

ables these two systems to share information. ACT provides a language for specifying actions and plans for both planners and executors. Cypress includes translators that can automatically map Acts onto SIPE-2 and PRS-CL structures, along with a translator that can map SIPE-2 operators and plans into Acts. Using the ACT interlingua, PRS-CL can execute plans produced by SIPE-2 and can invoke SIPE-2 in situations where runtime replanning is required.

The Act-Editor was implemented as a subsystem of Cypress. The Act-Editor is a graphical knowledge editor for creating, displaying, and manipulating Acts. It provides knowledge-editing capabilities for both SIPE-2 and PRS-CL.

In contrast to many other agent architectures, planning and execution operate asynchronously in Cypress, in a loosely coupled fashion. This approach makes it possible for the two systems to run in parallel, even on different machines, without interfering with the actions of each other. In particular, PRS-CL remains responsive to its environment during plan synthesis. The subsystems of Cypress can function independently of each. However, Cypress is used most advantageously as an integrated planning framework that supports a wide range of planning and execution activities.

PRS

PRS-CL is a framework for constructing persistent, real-time controllers that perform complex tasks in dynamic environments while responding in a timely fashion to unexpected events. PRS-CL has proven useful in developing several demanding applications that required integration of reactive and goal-oriented behavior, including a monitoring and control system for the Reaction Control System of the NASA Space Shuttle (Georgeff & Ingrand 1988) and a control system for naval battle management aboard a Grumman E-2C (Ingrand, Goldberg, & Lee 1989).

Individual instantiations of a PRS-CL system are referred to as PRS application agents. A PRS-CL application agent consists of a database containing current beliefs or facts about the world, a set of current goals, a set of procedures describing how sequences of actions and tests may be performed to achieve certain goals or to react to particular situations, and intentions that keep track of the current procedures being executed by the agent. Multiple PRS agents can be active simultaneously. Each PRS agent has its own local goals, intentions, and database and runs asynchronously in the overall framework.

A PRS-CL agent interacts with its environment
through its database (which acquires new beliefs in response to changes in the environment) and through the actions that it performs as it carries out its intentions. While the system is running, it constantly monitors incoming information and goals. Activity is triggered in response to the adoption of an explicit goal or to some change in the world. This combination of goal- and data-driven activity yields a flexible, adaptive plan execution framework. In particular, any intention can be interrupted and reconsidered in the light of new information about the world. The monitoring method used guarantees that any new fact or goal is noticed in a bounded time, thus providing rapid response to new events.

PRS-CL has the properties necessary for the executor component of taskable reactive agents: it is reactive, integrates goal-driven and event-driven activities uniformly, and has proven effective in numerous applications. The ability to define multiple PRS agents supports the simultaneous use of multiple instantiations of our abstract agent model.

SIPE-2

SIPE–2 is a partial-order AI planning system that supports planning at multiple levels of abstraction. It has the properties required by our agent model, including the ability to generate action sequences that include parallel actions, conditional actions, and resource assignments, and the ability to perform temporal reasoning. SIPE–2 can also be used to modify previously generated plans, an essential capability for runtime replanning. In contrast to most AI planning research, heuristic adequacy has been one of the primary goals in the design of SIPE–2. Example applications include planning the actions of a mobile robot, planning the movement of aircraft on a carrier deck, travel planning, construction tasks, the problem of producing products from raw materials on process lines, and joint military operations (Wilkins 1990; Wilkins & Desimone 1994).

Gister

Gister-CL implements a suite of evidential reasoning techniques that can be used during plan generation and plan execution to analyze uncertain information about the world and possible actions (Strat & Lowrance 1989). The techniques are based upon the mathematics of belief functions developed by Dempster and Shafer and have been successfully applied to a variety of problem domains. Gister-CL supports the construction, modification, and interrogation of evidential structures, thereby allowing a domain expert to quickly and flexibly develop an argument (i.e., a line of reasoning) specific to a given situation.

The ACT Language

ACT was developed as a domain-independent language for representing the kinds of knowledge about activity used by both planners and executors. A full presentation of the ACT language can be found in (Wilkins & Myers 1995). Here, we summarize some of the issues that motivated the design of ACT.

Planners and executors share many representational requirements. Both require a language for expressing goals, a means of ordering activities, and mechanisms for encoding information about when a particular procedure can be applied. Concepts such as protection intervals and resources are useful for both types of systems. Nevertheless, there are important representational differences for the two classes of systems due to their different functionalities. The planner is looking ahead to the future consequences of courses of action, while the executor is sensing the world and responding to incoming information and goals. These differences affect action representation in several ways. For example, execution systems frequently use conditionals and loops, and must provide the means to express reactive behaviors for operating on the world. Generative planners stress subgoaling, parallel actions, and reasoning to predict the result of taking certain actions. The ACT language is sufficiently expressive to support all of these concepts.

The development of the ACT language is similar in motivation to the KRSL language (Lehrer 1993). In fact, the ACT formalism has been one of the formalisms driving the design of the specification for plans in KRSL. The KRSL effort is very ambitious, trying to develop a common language for many types of systems, including systems for planning, execution, scheduling, simulation, temporal reasoning, database management, and other tasks. In contrast, the ACT formalism is focused more narrowly on a common language for planning and execution systems. We view this narrow focus as critical to achieving success in near-term use in software tools, since a common representation that tries to cover too broad an area faces many serious problems (Ginsberg 1991).

It is always difficult to evaluate the merit of a representation language. However, we believe that the design of the ACT formalism has been validated by its successful use in Cypress applications ranging from military operations planning to real-time tracking and fault diagnosis. The ACT formalism should be viewed as an evolving entity though, whose expressiveness will undoubtedly increase in the future to support additional applications.

Asynchronous Runtime Replanning

Cypress contains a domain-independent runtime replanning capability, which allows PRS-CL to invoke SIPE–2 to perform replanning for failures that cannot be remediated locally by the application of predefined Acts. The replanning framework supports the following behavior:

1. PRS-CL detects an irrecoverable failure during plan execution.
2. **PRS-CL** communicates the current state of execution to **SIPE-2**, and then *continues executing those parts* of the plan that are unaffected by the failure.

3. **SIPE-2** invokes its replanner to produce an alternative plan.

4. The new plan is translated to an Act and forwarded to **PRS-CL**.

5. **PRS-CL** merges the new plan with its current activities and continues execution.

Step 2 highlights an important characteristic of our replanning framework, namely, its *asynchronous* mode of operation. For asynchronous replanning, plan execution continues on those branches of the plan that are not affected by the failure. This mode of operation contrasts with *synchronous replanning*, in which plan execution is halted while an alternative plan is generated. Asynchronous replanning presents greater technical challenges, the most critical of which is to reconcile the state to which plan execution has progressed during generation of a new plan with the new plan itself. Asynchronous replanning is critical in many domains, since it is infeasible to halt execution while replanning occurs for some parts of the plan.

**Replanning Architecture**

Our basic model for replanning is bottom-up in nature, being driven by the activities of the executor. The executor is responsible for recognizing replannable failure situations, requesting new plans from the planner, and finally implementing the new plan.

We developed a *transformational approach* to replanning where the activities in the original plan are left unmodified when possible. In particular, the planner modifies the failed plan during replanning rather than generating a completely new plan, and the executor continues execution of original-plan threads that are unaffected by the failure while replanning takes place. This approach contrasts with many existing methods that simply abort execution of the old plan upon failure, and then begin execution of a new plan. The transformational approach has the advantage of preserving undisturbed those activities in progress that remain part of the new plan. This property is essential in domains where it is infeasible to halt all execution activities while replanning a portion of the overall plan.

An application **PRS** agent initiates the replanning process upon detection of a failure that it recognizes as replannable (Wilkins *et al.* 1995). All requests for replanning are forwarded to a **PRS** agent named **Replanner** that performs all necessary communication with **SIPE-2**, thus enabling the agent that requested replanning to continue execution of those parts of the plan that are unaffected by this failure. The **Replanner** agent is identical to other **PRS** agents: its specialized behavior results from the specific set of Acts that it executes. The architecture for the replanning framework is depicted in Figure 3.

The message sent to the **Replanner** agent indicates the name of the application agent requesting the replanning, a unique identifier for this particular failure episode, the plan being executed, the failed goal, and the *execution front*. The execution front is a list of the nodes last successfully executed on each parallel thread of the plan. The **Replanner** agent then sends **SIPE-2** relevant information from its database that characterizes the current world state. Without such updates, **SIPE-2** could generate plans only for the original state rather than the state in which the failure occurred, since it does not monitor the world during execution. The **Replanner** agent then awaits a response. In general, **SIPE-2** will reply with a new plan that addresses the failure. The new plan is forwarded to the **PRS** agent that requested the replanning, which then integrates it with its current activities. If no new plan can be generated for the overall goal, the **Replanner** notifies the requesting **PRS** agent, which in turn terminates its execution of the original plan.

When issued a replanning request, **SIPE-2** responds by trying to modify the failed plan rather than producing a new plan from scratch. Doing so is often more efficient, since only limited changes are generally needed to fix the original plan. **SIPE-2** begins by simulating the original plan through the execution front, and then comparing its expected world state with the actual state provided by **PRS-CL**. **SIPE-2** collects those formulae not expected to be true and determines how they affect the failed plan. The plan is modified to eliminate any problems by deleting unnecessary subplans and inserting new goals (as described in (Wilkins 1988)). The planner ensures that the future consequences of the new plan do not interfere with the still-active execution threads.

One of the key technical difficulties in developing the replanning capability was integrating the revised plan with the current activities of the executor. This problem is further complicated for asynchronous planning, since execution of the original plan continues while the new plan is generated. To this end, **SIPE-2** provides **PRS-CL** with both a new plan and a *node map*. The node map is a partial mapping from nodes in the original plan that have been changed or removed to nodes in the new plan. The node map encodes sufficient information for **PRS-CL** to transform its current activities during transfer of control to the new plan. Actions/subgoals being executed from nodes that are not in the node map simply continue execution. For mapped nodes, **PRS-CL** aborts their associated activities, and begins execution at the node in the new plan specified by the node map. The transformational approach required the development of a number of complex routines for manipulating the runtime activities of **PRS-CL**.
Planning under Uncertainty

Agents that operate in dynamic and unpredictable environments must be able to cope with uncertain information. There are several different sources of uncertainty. First of all, it is often impossible to characterize the current state of the world with complete accuracy. Imperfections in the domain knowledge can compound this situation, by adding inaccuracies to the expected outcome of actions. As well, external events may alter the world state in unanticipated ways.

There are several ways in which explicit reasoning about uncertainty can be valuable for taskable, reactive agents, including initial situation assessment, Act selection by both the planner and executor, Act parameterization, and plan evaluation. We have used Gister-CL within Cypress to aid in Act parameterization and plan evaluation (Wilkins et al. 1995). In particular, an in-depth evidential analysis can be used to select particular military forces to carry out specific missions in support of larger military objectives. Gister-CL could also be applied to reasoning about the likelihood of success in applying Acts, although we have not yet made use of that ability.

Cypress Application

An implemented example illustrates the use of Cypress within a military operations domain. This domain is an extended version of the one used for ARPI's second Integrated Feasibility Demonstration. The domain knowledge includes approximately 100 plan operators, 500 objects with 15 to 20 properties per object, and 2200 initial predicate instances. Plans range in size from several dozen to 200 actions, usually spread among numerous parallel threads of activity (Wilkins & Desimone 1994).

The scenario begins with a goal request for deterring several military threats. SIPE-2 is invoked to produce a plan using a set of Acts previously input to the system. During the planning process, Gister-CL is used to assist SIPE-2 in choosing appropriate military forces for particular missions.

The plan produced by SIPE-2 contains four main threads of parallel activities: two threads for deterrence using ground forces, one for deterrence using naval forces, and the fourth using air forces. Throughout the planning process, a PRS-CL application agent monitors the world for additional goals and events that might require immediate action. The execution agent executes the plan by applying appropriate Acts to refine the plan to lower levels of abstraction, eventually bottoming out in actions that are executable in the world. The execution agent remains responsive to new goals and events throughout.

As part of the air deterrence operations, aircraft are moved among various air bases. The use of an air base requires explicit transit approval, which is granted initially for all bases in the domain. An execution failure can be triggered by rescinding transit approval for any of the bases used in the plan. The execution agent records any such changes, and detects a failure when execution reaches the stage where the rescinded approval is required. No Acts are defined for repairing such a failure locally; thus, execution would completely fail at this point without replanning. When replanning is enabled, the execution agent will notify the Replanner agent, which in turn will issue a replanning request to SIPE-2. Meanwhile, execution of the remaining branches of the original plan continues without disruption.

Replanning for this situation produces a modified plan in which an alternative mobilization strategy is employed. One of our test cases results in the removal of a dozen actions from the plan, replacing them with a new subplan of similar length. The operations in the new plan are selected so as not to interfere with the continuing execution of actions on other parallel threads in the original plan. The new plan is sent to the PRS application agent, which integrates the new plan with its current activities and continues.

Conclusion

Cypress is a powerful framework in which to define agents that must accomplish complex goals in dynamic and unpredictable environments. The application of
Cypress to the military operations domain attests to the system’s usefulness.

The development of Cypress involved more than simply engineering the integration of existing systems. Multiagent, asynchronous replanning constitutes one important technological advancement, providing flexible plan execution that can adapt to significant unexpected changes in the world. In addition, interesting technical problems had to be solved to bring the system into being. Of greatest significance was the design of the ACT language as a common representation for both executors and planners. In developing ACT, subtle differences between these two classes of systems arose. PRS-CL had to be extended in numerous ways to support the execution of plans produced by SIPK-2: the kinds of knowledge employed in automatically generated plans involved constructs not found in the domain procedures defined for previous PRS-CL applications.

Several characteristics distinguish Cypress from other systems that provide both planning and reactive execution. Many systems do not use general-purpose planning and so cannot generate plans of sufficient complexity for many application domains of interest. Previous work in runtime replanning has either been limited to synchronous approaches (Laird 1990; Washington & Hayes-Roth 1992) or focuses on local, adaptive modifications to rule sets, rather than employing the full look-ahead reasoning of a generative planner (Lyons & Hendricks 1992; Firby 1987). The ability to modify a complex, parallel plan at runtime and adapt execution activity to the new plan is, to our knowledge, a new accomplishment.

Finally, developing multiagent systems like Cypress will be greatly facilitated by the results of the MPA project. Cypress employed three agents in the demonstration described above. This required hardwiring specific communication protocols into both the planner and executor. There is no meta-PA in Cypress that concisely documents the interaction of these agents. Using MPA, the communication capabilities would be readily available in the form of a wrapper, and the overall architecture of the three agents would be clearly defined in a meta-PA. The development, exploration and evaluation of different agents and alternative agent architectures should be much easier with MPA.

Acknowledgments

Except for the first two sections, most of this paper was previously published as part of “Planning and Reacting in Uncertain and Dynamic Environments” in the Journal of Experimental and Theoretical AI, volume 7(1), 1995. pp. 197-227 (reprinted with permission). The research described in this paper was supported by the ARPA/Rome Laboratory Planning Initiative under Contracts F30602-90-C-0086 and F30602-95-C-0235.

References


