Twenty-Third Annual International Computer Software and Applications Conference

A Design Pattern for Autonomous Vehicle Software Control Architectures

Michael L. Nelson

A Design Pattern for Autonomous Vehicle Software Control Architectures

Michael L. Nelson
Department of Computer Science
University of Texas - Pan American
Edinburg, TX 78539
m.nelson@computer.org

Abstract

Design patterns represent a generalized approach to solving a related set of problems. Typically, a design pattern does not** provide **reusable code. Instead it provides a common vocabulary and a generalized approach in an application-independent manner. We have developed a design pattern for use in controlling autonomous vehicles. The control of individual components within an autonomous vehicle will obviously differ from one vehicle to another. However, all of the components and subsystems must work together as a whole, This overall control is carried out by a Software Control Architecture, and includes concepts from artificial intelligence, computer vision, vehicle navigation, and graph theory. The Strategic-Tactical-Execution Software Control Architecture (STESCA) was developed to serve as a design pattern for autonomous vehicle control systems. The STESCA approach is currently being used to control both an autonomous underwater vehicle and a land-based wheeled autonomous vehicle in simulation.

1. Introduction

This paper presents a design pattern used to create software control systems for autonomous or robotic vehicles. It has been used to develop control systems for two different types of vehicles (one an underwater vehicle, the other a wheeled land-based vehicle) in simulation, and is currently being implemented on a 'real world' land-based robot. We begin with a discussion of design patterns, followed by an overview of autonomous vehicles. We then present the Strategic-Tactical-Execution Software Control Architecture (STESCA) as a design pattern for use in controlling robotic vehicles.

2. Design patterns

The simplest definition of a design pattern is that it represents a generalized approach to solving a related set of problems. In teaching an introductory course in computer programming [1], Figure 1 is presented as a "common approach" to program design. These are almost insultingly simple, but they are commonly used in solving problems on a computer. Fortunately, design patterns get quite a bit more involved (and interesting) than this, or it would not make for a very interesting topic. And design patterns are currently a very hot topic, with a lot of interest within the community.

One of the reasons that design patterns are such a hot topic is that they provide a form of reusability in computer software. Hardware designers have long known the benefits of reusability; the computer itself is rarely built from "the ground up" (i.e., by designing and building each individual component). Instead, many off-the-shelf components are used. And when a new component is needed, it is usually not designed in a vacuum. Rather, it is designed using ideas, principles, and approaches learned in creating previous similar components.

Reusability in computer software began in earnest with libraries of modules (typically functions and procedures) that could be used in many different applications. These modules are often referred to as low-level routines. The object-oriented (00) approach helped to extend the concept of reusability into higher-level code by providing libraries of classes that could be used in various applications. In either case, however, the basic idea is the same: provide actual code that can be used by several applications.

Design patterns offer an even higher level of reusability for computer software: reuse at the design level. That is, much like the design of a new hardware component, the design proceeds using ideas, principles, and approaches learned in creating previous designs. Wegner used the analogy of a template in describing the class definition [2]; that is, the class definition serves as a template in creating new objects. Similarly, the design pattern is like a template during the design stage in that it serves as a template for the new design.

Of course, there are defmitions for design patterns that are much more involved than our simple definition. The 'gospel' for the 00 community comes from Gamma et al [3], in which design patterns are described as "simple and

0-7695-0368-3/99 $10.00 © 1999 IEEE

Welcome/Header Message Get Dam Calculate Results Output Results	Welcome/Header Message Loop Until Done Get Data Calculate Interim Results Output Interim Results Calculate Final Results Output Final Results
Figure 1 a: 'Common' approach	Figure 1 b: 'Calculate as you go' approach

Figure 1. Very simple design patterns

elegant solutions to specific problems in object-oriented software design" [3, p. xi]. They then go on to state that design patterns "capture solutions that have developed and evolved over time. . . They reflect untold redesign and recoding as developers have struggled for greater reuse and flexibility in their software." The authors then describe several design patterns, and propose a way to organize them. It should be noted, however, that while Gamma et al discuss 00 design patterns, there is nothing in their basic definition that makes it an exclusive concept of the 00 community.

The organization of design patterns is also a very important topic, and has been discussed in several recent books and papers [3, 4]. However, the organization of reusable components is really nothing new, and is mainly a numbers problem. That is, as the number of components (design patterns, in this case) grows too large for a person to grasp and understand, how can we organize (or classify) them so that we can find the one that we need? This same problem has been addressed to varying levels of success for libraries of software components [5].[1]

3. Autonomous vehicles

An autonomous vehicle (AV) is both unmanned and untethered. Unmanned means that there is no 'person-in-the-loop' to make decisions should unexpected problems arise. Untethered means that there is no communication with the vehicle once it is underway, so it cannot request new or additional instructions from a human operator.* Thus, an AV must be able to recognize potential problems and respond to them independently of human intervention.

[1] The organization of components is also a problem for hardware - leafing through any electronics catalog shows a multitude of components available. In the same way that the software community is looking to the hardware community for ideas and analogies concerning reusable components and design, perhaps we should look also to them for ideas on organizing the components...

[2] Actually, untethered only implies no communication to the vehicle. It could be sending information back that was collected while underway (i.e., one-way communication from the AV).

Since an AV is both unmanned and untethered, it must have a fully specified mission that it can carry out under normal circumstances. It must also be able to handle any problems that can be 'reasonably' anticipated, such as obstacle avoidance and path replanning. For those problems that cannot be handled while underway, it must also have some 'fall-back' positions, such as skipping some portion(s) of the mission or even aborting the entire remainder of the mission.

Controlling an Autonomous Vehicle (AV) is a highly complex task. The control of individual components within an AV generally falls under the realm of classical control theory. As such, it has a solid mathematical foundation, is well understood, and is relatively straightforward to implement. However, for an AV to function properly, all of the components and subsystems must work together as a whole. This overall vehicle control goes far beyond classical control theory. Concepts from artificial intelligence, computer vision, vehicle navigation, and graph theory may all be used together to implement 'human-like' reasoning as applied to problems such as motion control, path planning, and obstacle avoidance. [6]

There are three main problems must be addressed in successfully controlling an AV. First, there is the control of individual components within the AV. Second, there must be a way of expressing a mission for the vehicle. Third, the appropriate vehicle component commands must be created from the stated mission, handling as many unforseen circumstances as possible.

This overall control system can be carried out by a Software Control Architecture (SCA). While many such systems have been developed, none have been general enough to be used on a variety of vehicles. STESCA, a tri-level approach to vehicle control, has been used to successfully control both an Autonomous Underwater Vehicle (AUV) and a land-based wheeled AV. STESCA was intended to serve as the basic design for control of any type of autonomous vehicle. In moving from an underwater vehicle to a land-based vehicle, this basic design is proving itself as a design pattern.

A software control architecture (SCA) provides a framework upon which a complete software control system

can be built. It must be capable of representing and responding to events and obstacles in the real world. It should also help to manage the complexity associated with the various vehicle subsystems, both hardware and software. In other words, it must be able to handle the three main problems previously discussed: mission specification, individual component control, and conversion of the stated mission into appropriate component commands taking into account objects and events in the real world (A discussion of alternative approaches is considered to be beyond the scope of this paper, but may be found in [7]).

3. STESCA: the Strategic-Tactical-Execution Software Control Architecture

An object-oriented (00) approach was used in creating STESCA. This has been beneficial for many reasons. 00 modeling allows for 'real world' objects to be modeled directly on the computer. Thus vehicle components are clearly identifiable, they are not 'hidden' or 'spread out' between various routines and dam structures. 00 design (OOD) encourages solving problems in a computer in the same way as we solve problems in the 'real world.' Thus the AUV version of STESCA is based upon the way that a human crew controls a manned submarine. This provided a working model to use as the basis for our approach.

There were two primary goals in developing STESCA. First was to create a 'generic' framework to simplify the process of creating an SCA for autonomous vehicles. Second was to allow mission specification by 'anyone' with a 'minimal' level of training (while it takes a team of scientists and engineers to design and build an autonomous vehicle, it should not take such a team to operate it). The first goal led to a design pattern for an AV SCA that is the subject of this paper (the second goal concerning mission specification is discussed further in [7, 8]).

STESCA is a tri-level approach to vehicle control. Mission specification occurs in the top Strategic level. The middle Tactical level converts the mission specification into actual vehicle component control commands, maintains data collected during the mission, and is responsible for mission and path replanning (should it be necessary) while the vehicle is underway. The bottom Execution level consists of the software interfaces to the individual vehicle components.

Consider the following story that presents a human analogy for the three levels of STESCA. Suppose that your spouse tells you to go to the store to get some milk. The mission, expressed in the Strategic level, is simple: go to the store, get some milk, and return home (note that path planning is done 'off-line' before the mission begins). The Tactical level accomplishes this by starting the car, backing out of the driveway, driving to the store, etc. However, the Tactical level has no working knowledge of how the vehicle actually works, only how to turn the key, shift gears, step on the brakes, and so on. It is the Execution level that translates these commands into the actual electrical / mechanical activities that control the vehicle, causing it to move.

3.1. The design of STESCA

The emphasis in STESCA is on *what* each level should do, not how it is accomplished. This emphasis on what rather than how maps directly into the object-oriented approach, where the emphasis is on what the various objects do rather than how they accomplish their various activities. Figure 2 shows the major components of STESCA.

The Strategic level consists of the mission specification system. The mission consists of a series of mission phases [8, 9]. A mission phase may be as simple as a single action, such as transit or change depth. It may also be a collection of these simple actions, or a collection of collections and simple actions. These collections may be named and stored in /retrieved from secondary storage. They may also be used to create a set of 'standard operating procedures' (SOP's). As each phase is sent to the Tactical level, the success ('status') is returned. If the phase successfully completes, the next phase is sent. If not, then an alternative phase may be chosen. The mission specification system was designed using inheritance, and was implemented via a linked list (with the design details, of course, hidden from the user). The class MissionPhase includes all of the pointers and methods necessary for constructing the list, and serves as the root of an inheritance hierarchy for all possible phases.

The Tactical level (see Figure 2) has many components. It is responsible for carrying out the various mission phases, controlling and coordinating the various vehicle components. The main components include the Vehicle Commander, Navigator, Engineer, Mission Specialist, Engineer, and three data stores: the World Model, Mission Model, and Data Recorder.

The Vehicle Commander, which receives the mission phases from the Strategic level, is responsible for carrying out each phase. It coordinates the activities of the other components of the Tactical level. The Navigator is responsible for computing the location of the vehicle. The Command Sender sends actual commands to the various components of the Execution level, receiving back the success ('status') of each command. The Engineer maintains the status of each component in the Execution level, and may check the status of the various components as necessary. The Mission Specialist, which is optional, is responsible for other mission dependent activities.

The three data stores in the Tactical level are the World Model, the Mission Model, and the Data Recorder. The World Model contains all the pertinent information known ahead of time about the area(s) of operation. This information may be used if mission or path re-planning becomes

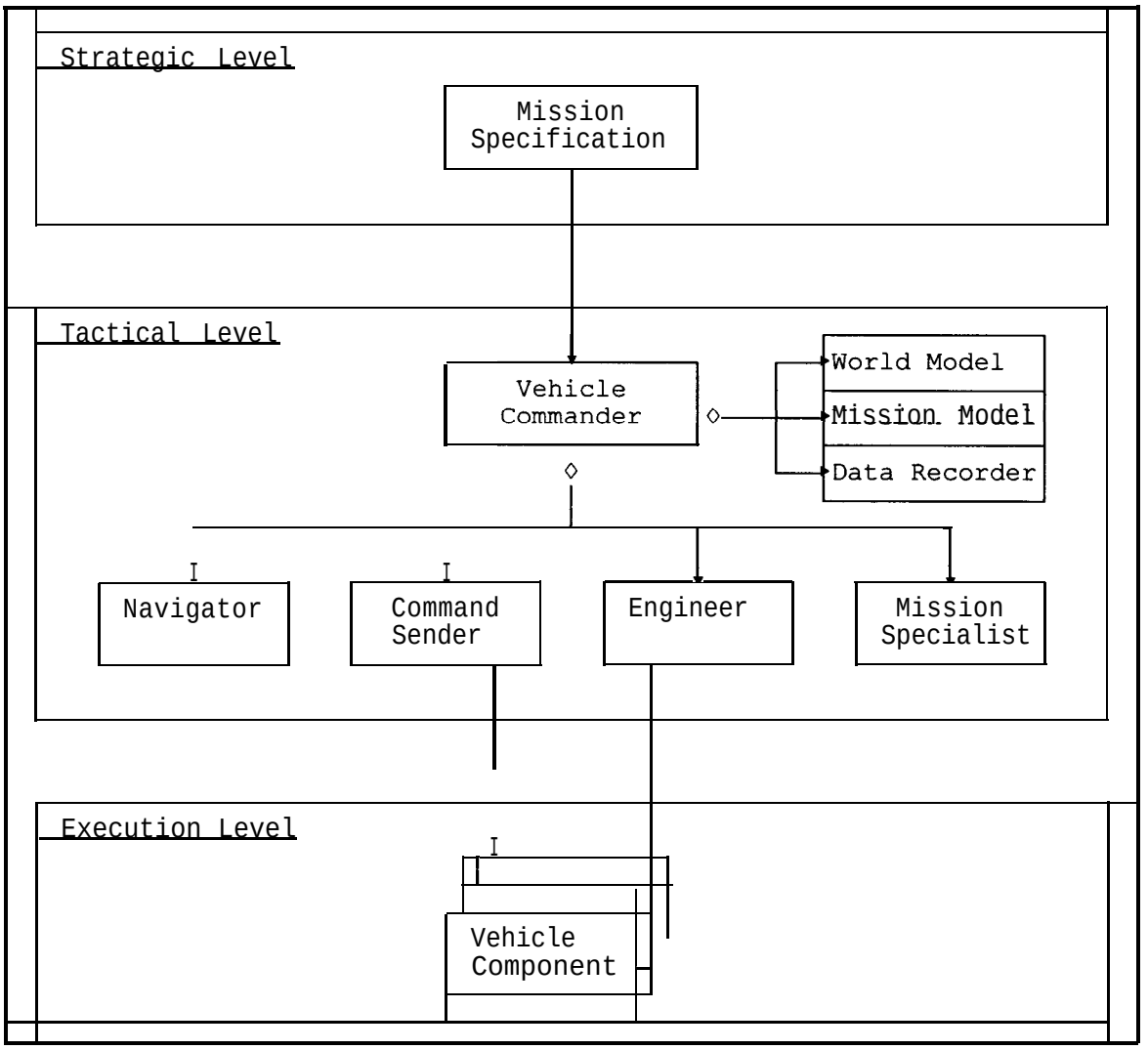

Figure 2: STESCA overview

necessary. The Mission Model is a log of all component commands and status checks sent to the Exectuion level. This information may be used after a mission is complete to assist in analyzing the success / failure of a mission. The Data Recorder is used to store information collected during a mission.

The Execution level contains the interfaces to the various vehicle components. This allows STESCA to be implemented on virtually any type of vehicle. As previously discussed, the control of individual components generally falls under the realm of classical control theory. As components are added to the vehicle, it is the responsibility of the Tactical level to control and coordinate their actions in carrying out the various mission phases.

The tri-level approach allows STESCA to be implemented for virtually any robotic system. The Execution level consists of the control code developed for each vehicle component. The Strategic and Tactical levels are then built on top of this code. The mission specification system (in the Strategic level) is designed so that only minimal familiarity with the vehicle is necessary. The Tactical level converts the stated mission into appropriate component commands, which are then sent to the Execution level.

3.2. STESCA as a design pattern

In describing STESCA as a design pattern, we generally follow the description proposed in [3]. STESCA would be classified as structural and applying to objects. It is structural in that it deals with the composition of and collaboration between classes and objects. The intent is to provide abstractions for the complex task of AV control. The motivation is that all AV's are faced with the same types of problems (high-level mission specification,

individual component control, and mapping the mission into appropriate component control commands). STESCA provides a layered approach to encapsulate these responsibilities in a consistent fashion when moving from one type of vehicle to another.

The applicability is primarily to autonomous vehicles, but as previously discussed it should also apply to robotic systems in general. It might even be possible to generalize it even more to make it applicable in nearly any type of control system application. The structure is as shown in Figure 2, with the primary participants being the Mission Specification, the Vehicle Commander which is composed of the various Vehicle Components.

Collaborations between components are minimized as much as possible. The end-user of the vehicle communicates only with the Mission Specification. The Mission Specification then communicates with the Vehicle Commander by sending requests to execute mission phase. The Command Sender communicates with the Vehicle Components by sending component commands. The Engineer also communicates with the Vehicle Components, but is only allowed to check their current status.

The consequences of using the pattern are fairly typical of the 00 approach. The overall complexity of the system is reduced to a more manageable level, but the increased overhead of message passing makes the system less efficient from a hardware point of view (obviously, however, this becomes less and less of a concern due to the continually increasing speed and decreasing cost of CPU and memory). However this is the typical tradeoff between human efficiency and machine efficiency. While there are defmitely more efficient ways to control a robot from a hardware point of view, we believe that STESCA represents a highly efficient approach from a human point of view. The layered approach, as well as the composition within the Tactical and Execution levels, breaks the control problem into understandable components. As needs change, the components can easily be replaced (or modified) as long as the new component maintains the appropriate interface (for example, there are many forms of navigation which can be used, and it is also possible to switch between forms of navigation while the vehicle is underway).

Before continuing, it is also worthwhile to compare STESCA with design patterns which have already been identified and catalogued. Consider, for examples, the Layers category [4]. Here we have layers of software that provide interface to one another in a hierarchical fashion. In STESCA there are three layers, but the user sees only the mission specification and execution portions of the top Strategic level (layer). The mission is executed by the Strategic level sending appropriate messages to the Tactical level, which in turn sends appropriate messages to the Execution level. The end user of the vehicle need know nothing about the Tactical or Execution levels, and the Strategic level need know nothing about the Execution level. It is the interface between the levels that is all important. This is also very similar to the Adaptor pattern [3].

Event Notification [4] is also very important within the Tactical and Execution levels, particularly between the Tactical level's Engineer and the various components in the Execution level. While the Engineer can check on the status of any component at any time, it is also up to the components themselves to notify the Engineer (if possible) of any problems that they experience. As such, the Engineer is an Observer [3] of the Execution level components.

STESCA most likely exhibits properties of other design patterns as well, either as a whole or within one of its three levels. Therefore, it is most appropriate to identify STESCA as a compound pattern (i.e., a design pattern which is composed of other design patterns). This should not be too surprising since the intended domain of applications (robotic vehicles) is a specific and highly involved problem area.

While class diagrams are often used in describing design patterns, it has been suggested that role diagrams are better suited for describing object collaboration based patterns [4]. Referring once again to Figure 2, an overview of STESCA, we see one form of a role diagram. This is especially true within the Tactical level, which makes perfect sense given its roots. The Tactical level in particular was modeled after the way that a manned vehicle operates. Each member of the crew (Vehicle Commander, Navigator, Engineer, Mission Specialist, etc) is defined in terms of what that member does (i.e., in terms of that member's role).[3]

3.3. Applying the STESCA design pattern

Implementing STESCA on a land-based vehicle was relatively straight-forward, and the initial prototype [10] was developed in less than half the time that it took to create the initial AUV prototype [8, 93, with both implementations being done in C++. One primary consideration is determining what type of high-level commands will be needed by the vehicle user. For instance, do we need to be able to tell the vehicle to "move forward 50 meters," "move forward 3 minutes," and/or "move to the design lab."

Another consideration that we are still grappling with is mission phase failure [11]. That is, if the vehicle fails to achieve a particular mission phase, what should happen then? The Navigator interface is also important, since it is quite often desirable to be able to switch between different

[3] It should be noted that although the roles are similar to those found in a human crew, they are not identical. Interaction between human crew membee are frequent and necessary. In STESCA, communication between the various crew members only occurs via the Vehicle Commander. This means that the crew members are only aware of themselves and the Vehicle Commander. If they need any information, they get it from the Vehicle Commander and do not care how or where that information comes from.

forms of navigation (e.g., dead reckoning or GPS-based). At this time we cannot really address the Mission Specialist as this exists as a 'stub' in current implementations (for now we are primarily concerned with controlling the vehicle, putting it to good use is a subject of future research). For our implementation, their were virtually no issues to consider at the Execution level. That is because we are essentially users of our vehicles, using the vehicle manufacturer's interface as the Execution level.

4. Conclusions

In summary, STESCA provides a way of handling the complexity involved in controlling autonomous vehicles. It serves as a design pattern in that it provides an approach which can be used to control virtually any type of vehicle. So far, it has been successfully used in simulation to control both the Phoenix AUV [12] and the Pioneer ATRV [13], and development has just begun for the Pioneer ATRV vehicle itself. We have shown that STESCA is indeed a design pattern for controlling robotic vehicles, and that since it exhibits and includes other design patterns to various degrees, it is more appropriately called a compound design pattern.

In closing, however, we would like to revisit Figure la. In using STESCA to create a mission and control an AV, we find that four things occur: first we get a "welcome to STESCA" screen; second we enter the desired mission; third the high-level mission is converted to individual vehicle component commands; and fourth the component commands are 'output' (not to a printer but to the vehicle components). Maybe we haven't come as far as we thought, as we are still using that basic approach to problem solving. Then again, maybe this is simply a classic case in which the beauty of the design is in its simplicity.

5. Acknowledgments

This work was supported in part by a Faculty Award for Research (FAR) from NASA. Many faculty and students at the University of Texas - Pan American have assisted in the overall creation and testing of STESCA. The faculty, staff, and students at the Naval Postgraduate School (NPS - Monterey, CA) have also been most helpful in this effort.

6. References

[1] M. Nelson, *CSCI* 1300: *Computers and Society,* lecture notes, University of Texas - Pan American, Edinburg, TX, 1999.

[2] P. Wegner, "Dimensions of Object-Based Language Design," *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA '87),* Orlando, FL, Oct 1987, pp 168-182.

[3] Gamma, E., R. Helm, R. Johnson, & J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, MA, 1995.

[4] W.F. Tichy, "A Catalogue of General-Purpose Software Design Patterns," *Technology of Object-Oriented Languages & Systems (TOOLS 23),* Santa Barbara, CA, Jul 1997, pp 330-339.

[5] T. Poulis, *Using an Object-Oriented Database Management System to Enhance the Reusability of Class Definitions,* Naval Postgraduate School, Monterey, CA, Sep 1992.

[6] M.L. Nelson, R.B. Bymes, S-H. Kwak, R.B. McGhee, "Putting Object-Oriented Technology to Work in Autonomous Vehicles," *Technology of Object-Oriented Languages and Systems (TOOLS), Santa* Barbara, CA, Aug 1993, pp 279-288.

[7] M.L. Nelson, "A Software Control Architecture for Autonomous Vehicles," *31st Hawaii International Conference on System Sciences (HICSS-31),* Kohala Coast, HI, Jan 1998, pp 226-232.

[8] M.L. Nelson and V. Rohn, "Mission Specification for Autonomous Underwater Vehicles," *Oceans '96,* Fort Lauderdale, FL, Sep 1996, pp 407-4 10.

[9] M.L. Nelson and V. Garcia, "An Object-Oriented Approach to Autonomous Underwater Vehicle Control," *10th International Symposium on Unmanned Untethered Submersible Technology,* Durham, NH, Sep 1997, pp 385-393.

[10] M.L. Nelson, J.R. DeAnda, R. Fox, and X. Meng, "A Software Control Architecture for Autonomous Vehicles," *Aerospace / Defense Sensing, Simulation and Controls (AeroSense),* Orlando, FL, Apr 1999 (not yet published).

[11] M.L. Nelson and G. Flores Jr., "Failure Path Planning in STESCA," *23rd International Computer Software and Applications Conference (COMPSAC '99),* Phoenix, AZ, Oct 1999.

[12] Naval Postgraduate School Center for Autonomous Underwater Vehicle Research Web Site, http://www.cs.nps.navy.mil/research/auv/, Monterey, CA, 1999.

[13] Real World Interface (IS Robotics) web site, http://www.rwii.com/, Jaffrey, NH, 1999.