Software Platforms for Unmanned Systems

Software platform characteristics and developer advantages

Rajive Joshi, Ph.D., Arnab Bose, Ph.D. and Scott Breneman, Ph.D.
Real-Time Innovations, Inc.
Sunnyvale, CA

ABSTRACT
To simplify system development and software integration, specialized software platforms are now employed in various unmanned vehicles, including unmanned air vehicles (UAVs), mobile ground robotics and underwater vehicles. The "software platform" provides a software framework and supporting tools in an integrated development environment.

This paper describes some examples of the use of software platforms in the development of unmanned vehicles and common characteristics of the software platforms.

INTRODUCTION
System software development and integration is a major challenge for unmanned systems. Different tools, modeling abstractions and engineering skills are used at various stages of a typical development project. These stages may range from writing low-level data input and output (IO) device drivers in a programming language such as C, to building physical simulation models and control algorithms with dynamic simulation tools such as Simulink, to developing human machine interfaces (HMI) in a high-level language such as Java. All of these must be finally integrated into a coherent, documented, and fully-tested system.

Unfortunately, this process often breaks down, especially since "ad hoc" software designs provide too little structure to help the process. Software platforms represent an effort to eliminate the developer’s risks of such break downs.

A software platform provides a framework and supporting tools. The framework is a domain-specific software architecture that provides the means to build components, assemble them into subsystems, and integrate those into applications. Frameworks also provide partially-implemented patterns of behavior that are finalized for a particular application.

An example of a popular software framework is the Microsoft Foundation Class (MFC) system for Windows GUI software development. The MFC framework provides partially implemented graphical widgets and a means to interconnect them in a predefined a user-driven, event-action processing queue.

The framework typically supports a hierarchical software component model with domain-specific behavioral patterns. In the MFC framework, a graphical interface can be composed hierarchically from the primitive graphical entities such as pull-down menus and selection boxes into complex windows. The framework provides necessary behavioral system that handles mouse and keyboard stimulus and routes the stimulus to the correct graphical element. The graphical elements then provide a well-defined interface to handle the events as they come from the MFC framework.

A framework for unmanned vehicles would support discrete time data-flow and event-based finite-state machine components. The framework would also provide an infrastructure to construct layered control software of increasing complexity from components. Object-oriented implementation mechanisms are also powerful implementation tools for a framework.

The scope of control in unmanned systems can vary from low-level servo loop control, to trajectory planning, to mission control and distributed interactions (squadron or "swarm" behavior). Commercial software development tools can aid at different points of control software development. For example, many developers are using off-the-shelf C compilers to develop sensor and actuator interfaces. For higher level modeling and simulation, developers are using tools such as Simulink to model complex control systems. At the same time, system designers are using Unified Modeling Language (UML) tools to define the objects and interfaces.

More and more, modeling tools generate production-quality code for integration into application. The frame-
work is the infrastructure that integrates software components from such disparate sources.

**CHARACTERISTICS OF UNMANNED SYSTEMS**

Unmanned vehicles for air, land, sea, or space are becoming increasingly sophisticated with multiple levels of control and automation (Figure 1). With increasing demands on maneuverability and operational performance, the electronics and the intelligence they contain become more complex and pose significant hardware and software integration challenges. The major elements of on-board vehicle electronics include sensor subsystems for positioning and reconnaissance, actuation subsystems for driving the vehicles, control subsystems for navigation, and guidance and mission-specific tasks.

Increasingly-sophisticated sensors with overlapping and redundant functionalities are becoming available. They, in turn, require highly sophisticated, multisensor fusion algorithms and support for multiple operating modes. The various sensing modalities work in conjunction with the control algorithms at multiple levels of task abstraction to accomplish a mission.

The control software is developed hierarchically with multiple levels of functionality. At the lowest level of software functionality are the device drivers that interface with the hardware. This layer abstracts the hardware interface in a meaningful way for use at higher levels of functionality. The device driver software layer must be very efficient and typically involves hand-written, custom code in C or C++. Sometimes the drivers are provided by the hardware vendor and need to be wrapped in custom code for integration with the rest of the system.

The next level of functionality is the low-level servo controls, necessary to achieve primitive control behaviors. The software at this level is characterized by periodic execution of data processing algorithms running at fixed rates of execution. This software is often hand-written custom code in C or C++.

The next level of functionality builds on top of the primitive servo behaviors and adds higher levels of intelligence and controls. The software at this level is typically characterized by event-driven behavior, conditional logic and sensor-based decision making. The software at this level is developed using a variety of techniques including hand-written C or C++ modules for conditional logic and behavior, graphical finite state machines, and Simulink models.

More and more, unmanned systems are inherently distributed, involving multiple processor nodes. For example, an unmanned system communicates with manned command and control stations for monitoring and mission control. In the future, squadrons of vehicles may be deployed as a group, where the vehicles must intercommunicate and coordinate activities to accomplish a mission.

The communication software linking the disparate systems is developed using a variety of techniques and interconnection middleware. These range from ad-hoc techniques, to custom, in-house solutions, to off-the-shelf middleware such as CORBA® for communications among distributed object and NDDS® for data-centric publish-subscribe communications.

Thus, the most appropriate framework for unmanned systems provides an infrastructure upon which the developer can assemble discrete components, built with a variety of tools, into increasingly complex subsystems that can be integrated locally with other subsystems, but at the same time allow interaction with remote subsystems and applications.

**SOFTWARE FRAMEWORKS FOR UNMANNED SYSTEMS**

Software frameworks are already in use by companies developing unmanned vehicles and robotic systems. Examples include the DARPA-funded Open Control Platform (OCP), developed by The Boeing Company to support Software Enabled Control (SEC) research projects, and Constellation® (nee ControlShell®), a commercial framework developed originally by researchers at the Aerospace Robotics Lab at Stanford University and now developed and commercialized by Real-Time Innovations (RTI).
Activities in OCP involve many of the SEC project participants, including research platforms at CalTech, Georgia Tech,12 Oregon Graduate Institute, University of Minnesota, Stanford University, University of California, Berkeley and the University of Washington. As a commercial development tool, the software framework in Constellation has seen a variety of applications in the unmanned vehicle domain, including:

- The OTTER autonomous underwater vehicle developed by the Aerospace Robotics Lab at Stanford University and Monterey Bay Aquarium Research Institute.14
- The QUEST teleoperated underwater vehicle by Alstom Schilling Robotics.
- The Robonaut anthropomorphic robotic system under development by NASA at Johnson Space Center.15

The use of a software framework can drastically reduce development time while creating robust and flexible application software. One significant benefit of software frameworks is that they provide a well-defined integration mechanism. Such mechanisms integrate low-level device drivers, hand-written or custom code, telemetry and communication software, and model-based intelligence with no to minimal rework. With the progress of model-based development and automated generation of production-quality code, the software framework can reduce development time substantially.

The following subsections describe the key characteristics of software frameworks and their benefits in the development of unmanned vehicle systems.

For the purposes of illustration, we shall consider the development of a vehicle cruise control system. In this simple illustration (Figure 2), a human driver issues commands to the cruise controller to turn the automatic cruise on or off, resume the cruise speed, accelerate, or coast. The driver can also send input to the vehicle directly, for example, by pressing the gas pedal, braking or shifting gears. The cruise controller uses feedback from the operator and drive sensors (e.g., speedometer) to maintain the cruising speed.

**COMPONENT-BASED**

Software frameworks for unmanned systems should support component-based development.10-12 A software component is an encapsulated software building block that accomplishes some well-defined functionality.

For example, Figure 3 shows the graphical representation of the Cruise Controller feedback control component. It has an interface composed of input (velocity), output (cruiseThrottle) and reference data (VelGain, AccGain). It also has a port (CruiseCommands) for accepting commands from the driver.

**Figure 3. Cruise controller "software component."**

Besides data ports, components may also provide services and functionality-based intra-component communications based on method invocations and messages. The software framework handles the interconnections and distribution of the communication mechanism between the components. The architecture and organization of the end-user application is readily mapped into the component system.

The benefits of a component-based approach include:

- **Reusability:** Components can be built, tested, and verified once, then used later in various places within the same and different applications.
- **Clarity:** Components highlight the structural elements of the software implementation and often correspond to physical components.
- **Reconfigurability:** Components can easily be "rewired" in a different interconnection topology.
- **Interoperability:** Components with compatible interfaces can communicate via those interfaces.

**INTERFACES ARE SEPARATED FROM COMPONENT IMPLEMENTATIONS**

Software frameworks clearly separate interfaces from implementation.2,3 Interfaces define data and methods that must be supported by a component that implements
Software Platforms for Autonomous Systems

The interface (Figure 4). Interfaces can be hierarchically composed. Implementations may be provided by a variety of components (Figure 5).

The key benefits of separating component interfaces from implementations include:

- Replaceability: A component’s implementation can easily be replaced with a different one (provided it adheres to the same interface).
- Extensibility: A preexisting component can be extended to provide additional functionality by implementing new component interfaces.
- Team development: Interfaces can be shared among team members who provide different component implementations that must interoperate.

Leveraging domain expertise: Each level of system functionality can call for a different skill set and expertise. With a common software framework, well-defined interfaces can be created and implemented by experts in that domain.

HIERARCHICAL

The software architecture for unmanned systems are hierarchical. Sensor and actuator data flows and state machines are assembled into subsystems which then interact with other subsystems through more data flows and state machines to build the systems. Some frameworks provide graphical editors for assembling components into more complex components and drilling down from one layer to the next (see Figure 5).

The more the developer can hide complex assemblies inside high-level components, the easier it is build the application. For example, team members working on different subsystems just need to use the high-level interfaces without getting bogged down in the low level details. Other key benefits of hierarchical components include:

- Re-use working code: Subsystems can be developed and independently tested. Other groups can use the code confident that it is in working order.

Figure 4. Cruise Controller "software interface." Compact view can be expanded to reveal details.

Figure 5. Cruise Controller component implementing the Cruise Controller software interface: (a) compact external view of the component, (b) expanded external view of the component, (c) hierarchically expanded internal view of the component revealing implementation details.
- Rapid prototyping: Custom components can be created and combined with prebuilt components to quickly prototype working applications.
- Scalability and layering: Higher levels of abstraction can be realized by combining low-level components.

**APPLICATION-SPECIFIC BEHAVIORAL DOMAINS**

The structure of a software framework for unmanned systems supports domains of execution that are particularly applicable to multi-layered, control-system software architectures.9,10 The two dominant execution domains for unmanned systems are **discrete time synchronous data flow** and **event-based finite state machine (FSM)** behavior.

Synchronous data flow behavior (Figure 6) captures the periodic execution of actions at a certain rate and in a certain order. Finite state machine behavior (Figure 7) captures the high-level decision-making logic that is triggered by various events inside and outside the system.

Frameworks targeted for unmanned systems provide built-in behaviors to address their specific control requirements. For example, a software framework may provide built-in facilities to create modal systems that switch on/off subsets of components depending on the mode of operation (Figure 8). Moreover, software frameworks have a threading model that supports multiple periodic rates of sampled data-flow execution and other execution models.

The benefits of domain-specific behaviors provided as part of the framework itself include:

- **Re-use**: Commonly recurring behavior patterns do not have to be reimplemented.
- **Consistent behavior**: The framework provides the behavior pattern, so the behavior does not change from one developer to the next.
- **Better performance**: The framework can provide optimized implementations of recurring behavior patterns.
SOFTWARE PLATFORMS FOR AUTONOMOUS SYSTEMS

SOFTWARE FRAMEWORKS VS. PLATFORMS

A **software framework** is an implementation system that predefines the common behaviors of a set of software applications. It simplifies the development of applications that use those behaviors.

A **software platform** is a framework plus an integrated suite of development and management tools. The tools help developers to reduce time-to-market at different stages of the application-development process. Figure 9 illustrates some of the classes of tools.

![Figure 9](image)

Figure 9. A Software platform includes a software framework and integrated tools for development, deployment, debugging, monitoring, and control.

The following subsections describe types of tools that can help unmanned system developers simplify common chores.

INTEGRATION CAPABILITIES

Software frameworks cut across all functionality levels in unmanned systems software and provide an architecture to integrate components at different service levels. Many unmanned systems, for example, are composed of low-level components written in C and C++, legacy code, high-level components developed with model-based programming tools such as MATLAB and Simulink, and communications interfaces implemented via middleware such as CORBA\(^{16}\) or NDDS\(^{9}\).

The cruise control system introduced in Figure 2 is modelled in Figure 10 and demonstrates why the software platform needs to have integration capabilities. The boxes indicating the driver (Driver), vehicle (Automobile) and cruise controller (CruiseController) are now represented by iconic components with specific inputs, outputs and interfaces for the signals and functions.

![Figure 10](image)

Figure 10. A Cruise Control System: the Cruise Controller is integrated with software components for the Vehicle and Driver.

In this figure, the Automobile component can be a software simulation or the actual vehicle hardware. The vehicle dynamics simulation model may have been developed using MATLAB/Simulink, a popular tool for controls engineers. A real hardware component may include device drivers written in C and C++.

The data connections between the Cruise Controller component and the Automobile component may be over local shared memory or over a network. The Driver component may be an HMI or user interface that simulates a human driver, perhaps written in Java. The connection between the driver and the cruise controller may also be over a network or through the field bus, for example using CORBA in simulation and then the CAN field bus in deployment.

Even in this simple model, there is a comprehensive interplay of components built with different tools (C, Simulink, Java), in different modes (simulation versus deployment), and with different communications (CORBA versus CAN). We can see that the ability to integrate components is important. The benefits of using a software platform that provides the ability to integrate such disparate components include:

- **Skill integration**: The different skills required to build a device (device I/O, communications, Simulink modeling) can be unified around a single framework.
- **Infrastructure programming automation**: Process automation increases overall autonomous system performance and enhances the ability to meet mission goals.
• Higher quality: A platform (especially a commercially available one) is likely to get exercised in many different projects and is likely to be better tested with fewer bugs.

PROGRAMMING AND PRODUCTIVITY TOOLS
Modern unmanned systems are developed with a fast-track (spiral) development process and are generally very interdisciplinary projects. A software platform can support the use of programming aids from multiple vendors to enhance productivity and accommodate various development skills and paradigms. These aids include:

• A graphical editor to define and assemble components at each level of the application hierarchy.
• Systems analysis and design aids, including simulation and algorithm development tools such as MATLAB and Simulink.
• Software design, analysis, and documentation aids based on the Unified Modeling Language (UML).
• Source code application development environments, such as Tornado® and Visual Studio®, and debuggers such as StethoScope®.

COMPONENT REPOSITORY MANAGEMENT
As the components developed with a software framework proliferate, they must be managed. A software platform can provide the infrastructure to manage the software components. This infrastructure can include:

• Repositories and "explorer" for organizing and cataloging components and their documentation (Figure 11).
• Facilities to search for components based on keywords or other criteria.
• A method for sharing components among teams of developers.
• Customized color palettes to highlight frequently used components.
• Facilities to support component evolution, such as the versioning and deprecating of components.

MONITORING AND CONTROL TOOLS
Unmanned systems operate in real-time. Beyond the user interfaces and HMIs that are part of an autonomous system, a software platform can provide tools to monitor and control a live running application.

• Monitoring and displaying of live and captured data signals (Figure 12). The tool would let the operator select signals and specify any triggering events to start and end the capture. The data should be made available for display in real-time or stored for later analysis.
• Monitoring and displaying of live states. For example, the tool could animate the current status of a finite state machine (see Figure 7).
• Navigation and control of runtime components. For example, the operator could send commands to individual components, change parameters without modifying run-time code, and test "what if" scenarios. Such capabilities can be immensely helpful during field diagnosis and repair.

Figure 11. Repository for software components promotes sharing and reuse.

These tools can include:
To be used effectively, a software framework must be complemented by tools that promote effective sharing and reuse among users, development teams and organizations. Integrated development environments (IDEs) can integrate such tools into the software development process, including:

- Configuration management tools to manage project artifacts, including code and documentation.
- Requirements management tools (such as DOORS) to provide traceability and validation against high-level product requirements.
- Unit testing tools to verify and validate a component.
- Flexible deployment tools to build and deploy on multiple hardware platforms and targets.

**SOFTWARE PLATFORMS IN UNMANNED SYSTEMS EXAMPLES**

Software platforms have been applied to a variety of commercial and research applications, including UAVs, space robotics, mobile ground robotics, and underwater vehicles. In this section we describe a few commercial and research applications and the benefits realized from the use of software frameworks.

**ALSTOM SCHILLING’S QUEST ROV**

Alstom Schilling specializes in teleoperated manipulator systems and Remotely Operated Vehicles (ROVs) for hazardous and underwater environments. Schilling has used RTI’s software framework across multiple product lines. Their most recent commercial product is the Quest ROV for underwater servicing, seen in Figure 13(a).

Schilling used the software platform to control the Quest’s dual robotic arm system (not visible in picture). They also used the platform to control the dual robotic arm system used in a nuclear waste cleanup project. The software framework provided a reliable platform for experimentation and rapid prototyping and allowed them to re-use over half of the software components from a sub-sea operations project for the nuclear waste cleanup project. This reduced the development and integration time by an order of magnitude.

This application makes extensive use of software framework features, such as multiple modes, data-flow programming, finite state machine programming, and low-level device drivers. For example, the ROV control software has multiple modes of operation, including tele-operated open loop, tele-operated open loop with pitch and roll compensation, cruise open loop, cruise with roll and pitch compensation, and individual thruster control, among others. In each mode, a different set of controllers, sensors, and actuators are active and operate in a periodic sampled data-flow manner. A finite state machine encodes the decision making logic, switching the mode based on inputs such as the operator, health monitoring sensors, and operational conditions.

**STANFORD’S OTTER UNDERWATER VEHICLE AND MESICOPTER UAV**

The OTTER is a hover-capable, underwater vehicle developed by Stanford University in conjunction with the Monterey Bay Aquarium Research Institute (MBARI). Seen in Figure 13(b), the OTTER used a software framework for the development and integration of software at different levels of functionalities. For example, the low-level device drivers for the underwater sensors and motors are integrated with high-level control algorithms and task-level decision making logic using a software framework.

Another unmanned application at Stanford is a “mesicopter,” a small 4” helicopter, seen in Figure 13(c). It has 6 degrees of freedom. The control algorithm is based on a vision sensor feedback.

A software framework was used to wrap the vendor-supplied device driver for the vision camera in custom code and to integrate it with custom code for the control algorithm. This application demonstrates the integration capabilities of a software framework.

The vision processing was done on a dedicated computer hooked up to the cameras. The processed vision data was fed via a TCP/IP network to the controls algo-
Software Platforms for Autonomous Systems

rithm on another computer. The output of the control algorithm was fed to dedicated hardware with motor controllers.

GEORGIA TECH’S UAV HELICOPTER

Georgia Tech is using a software framework to build a platform for UAVs. This project is part of the DARPA-sponsored SEC effort. The UAV platform is a helicopter, seen in Figure 13(d). The application uses a software framework and demonstrates, first through simulations and then via flight trials, fault detection and controller switching. In the application, the helicopter’s altitude actuator becomes non-operational during flight. This is detected by a preprogrammed fault detection and controller reconfiguration module that sends reconfiguration commands to switch controllers.

This application is a good example of the integration and communication capabilities of a software framework between modules written in different programming languages and running on different processors. The framework also allows reuse of legacy components that can be replaced with new components as the system develops.

CONCLUSION

Unmanned systems developers have come to rely on a combination of software technologies. Low level device drivers are written in C; high-level designs are defined in UML; MATLAB is used to build Simulink models; Java is used for the HMI; in-between, developers build their data flow processors and state machines. Then the struggle begins to integrate them all into a coherent system.

A software platform can simplify the developers efforts through all phases of unmanned system development. A software platform provides a framework upon which components - representing the Simulink models, device drivers, HMI interfaces, state machines, etc. - can be assembled incrementally into the working system. The software platform lets the developer use best-of-class tools to help build components, analyze performance, debug, and optimize.

The software platform can also provide practical, day-to-day benefits for code development:

• Reusable components: A component-based architecture allows components to be built and
Software Platforms for Autonomous Systems

- Tested once by experts in that domain, and reused in multiple applications.
- Improved productivity: Component repositories can be organized so teams can share and reuse components.
- Rapid prototyping: Domain-specific components, such as device drivers and complex components, can be assembled quickly for proofs of concept and demonstrations.

Software platforms are changing the way unmanned systems are developed. They provide a framework that relieves the developer from a large part of the design effort and promotes reusable components that take months off prototyping and production.

REFERENCES


