Reconfigurable mission system: A proof-of-concept

Manuel Darveau
R&D Department
Miranda Technologies
3499, Douglas-B.-Floreani
Montreal, Quebec Canada
mdarveau@miranda.com
http://www.miranda.com
P : (514) 333-1772
F : (514) 333-9828

Martin Morissette
SONIA AUV project
École de technologie supérieure
1100 Notre-Dame West
Montreal, Quebec Canada
martin.morissette@gmail.com
http://sonia.etsmtl.ca
P : (514) 396-8800 #7622
F : (514) 396-8624

Félix Pageau
SONIA AUV project
École de technologie supérieure
1100 Notre-Dame West
Montreal, Quebec Canada
felix.pageau@gmail.com
http://sonia.etsmtl.ca
P : (514) 396-8800 #7622
F : (514) 396-8624

Abstract

An autonomous robot is useful and successful if it is able to accomplish the desired actions in the right sequence and properly react to environmental changes. The mission system (or Artificial Intelligent (AI) software module) of such vehicles is responsible for decision making. Modifications to this system are time-consuming and error prone since code modifications, recompilations and executable uploading are industry-used standards. The SONIA Autonomous Underwater Vehicle (AUV) team from École de technologie supérieure (ETS) in Montréal, Canada has developed a proof-of-concept demonstrating the advantages of on-the-fly reconfigurable mission system. The mission system is composed of an on-board mission framework built around the simplest AI system, a state-machine based system. The AI system is fully reconfigurable using a telemetry interface featuring mission saving, mission loading, state reconfiguration inside a mission, state transition management based on conditions, parameters assignation and model checking to ensure mission integrity.

Introduction

An autonomous vehicle is a robot system capable of accomplishing tasks without human supervision or intervention in order to complete a mission objective. The mission system is a software system which ensures that the autonomous vehicle fulfills its deployment objectives. This system, also commonly referred to as the Artificial Intelligence module of the robot, provides strategies, directives and decision-taking mechanisms to the control software modules.

The mission system has many responsibilities including preventing situations that could be harmful to its environment or itself, planning displacements and fulfilling mission-specific tasks. As such, it follows that it has a high overall complexity. This complexity is not only inherent to the code itself but also to the development and testing processes of the mission system. In order to minimize the complexity and to lower the defect rate in the mission system, the SONIA AUV team has developed a robust engine, a suite of reusable mission building blocks and a graphical tool for easy, on-the-fly, mission reconfiguration.

Background

Since 1999, the SONIA AUV team from École de technologie supérieure in Montreal, Canada is devoted to the development of Autonomous Underwater Vehicle (AUV) technologies. Over the years, six underwater vehicle prototypes have been developed in order to compete at the annual AUVSI and ONR's International Autonomous Underwater Vehicle Competition[1]. This academic competitive event is focused on the development of autonomous systems by engineering students so that they will be part of the industry in the future. For the past years, the AUVSI and ONR's AUV competition made use of a provision in the rules that allowed the organizers to entirely reconfigure the competition ground between qualifications and finals, leaving a mere hour to adjust the mission system of the autonomous underwater robot. As explained in the Defect Introduction Model[2], below a certain time threshold relative to the software's size, the quality is compromised by the introduction of defects. A timeframe of one hour is well below any reasonable threshold to prevent defect introduction when the team needs to analyze the changes needed, design a new solution and implement it on the robot. Since a single defect in a mission system could jeopardize the integrity of the AUV and its ability to accomplish the desired task, the development of an automated tool was required to reduce the

likelihood of defect introduction. With that goal in mind, the SONIA AUV team designed and implemented a mission system reconfiguration tool named the Mission Editor (ME).

State-machine-based mission system

Over the years, several mission system architectures and paradigms for autonomous underwater vehicles have been used by different research centers. Since the SONIA AUV team is entirely composed of undergraduate students including several freshmans, the team was looking for a simple architecture for the mission system. Our review of the literature determined that even though the fuzzy-logic approach developed by the Technical University of Crete[4] and the rule-based technology developed at the Naval Postgraduate School[5] yield good results, they are not compatible with our project. The main issue is that they require either having a precise kinematics and hydrodynamics model of the vehicle or using scripting languages, both of which are not possible to develop with our currently available team skill set. The finite state-machine approach[3] has been demonstrated as being the simplest fully functional architecture for an AUV mission system.

The finite state-machine is composed of three elements. First of all, a *state* possesses a set of specific parameters and behaviors. Each state has at least one *transition* to another state. The current state executes and provides action commands to the vehicle's

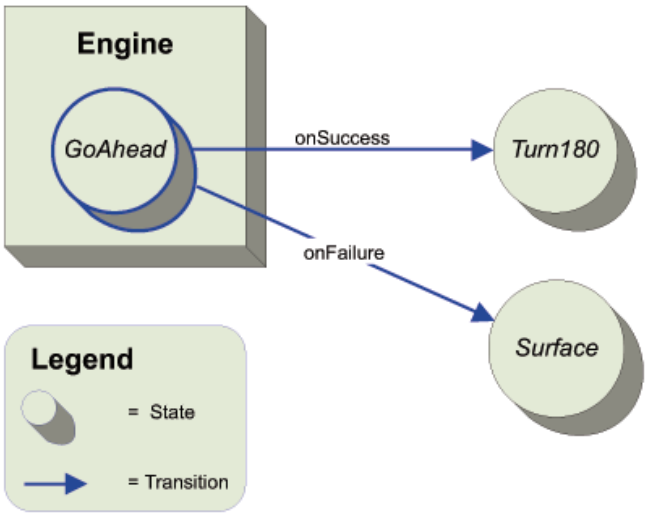

Figure 1: State Machine diagram

control engine. When it is detected that a state fulfills the requirements for one of its transitions, the active state is switched to the endpoint of the transition. The currently active state can also force a transition on high-priority external events, such as having run over the maximum allocated time slot or receiving a system event from an emergency situation. Figure 1 shows the main model elements of a finite state machine mission system. Using this model, the SONIA AUV team implemented for the first time in 2003 a mission system using generic finite state-machines.

Autonomous Underwater Vehicle software version 4 (AUV4)

The SONIA AUV team has developed a modular layered software architecture named Autonomous Underwater Vehicle software version 4 (AUV4). This software is entirely written in Java and runs under Linux on the embedded computer. AUV4 is responsible for data acquisition over the main Controller Area Network bus (CANBus) for all the sensors, data processing and fusion to determine positioning of objects and obstacles, running the state-machine-based mission system, controlling the thrusters using PID controllers[6] and sending the speed target to the thrusters over the CANBus. **Figure 2** illustrates the layered architecture of AUV4.

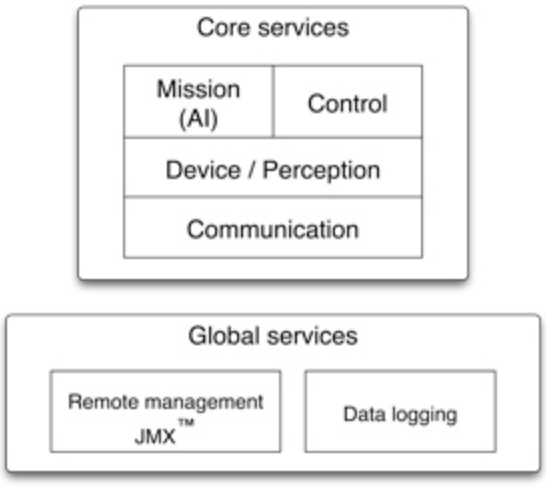

Figure 2: AUV4 Layered Architecture

AUV4 accomplishes these tasks (except communication) by using a single threaded control loop running between 10 Hz and 20 Hz. When the control loop is ready to execute the mission system, the state machine executes a single iteration of the current state. The code flow of this

state can set new target values to controllers and actuators. The control loop then executes one time step of the PID controllers to calculate the speed command for each of the thrusters while insuring that the superposition of commands does not prevent the higher priority control axis from remaining stable.

The states used by AUV4 can contain very complex logic. Missions are built by linking together these complex states with simple transition conditions so that a desired execution sequence is realized. The complexity of states that have successfully been integrated in AUV4 varies from a simple timer-based single-axis motion task to weighed-input logic systems and vision-driven object trackers employing sophisticated control algorithms. When a state has successfully executed beyond the fulfillment of one of its transition conditions, the next step in the mission order, along with its own parameters, can begin.

Problematic situation and outlooks

Before the development of the mission editor, we created each mission's state machine manually. This was done by first writing a set of simple but flexible state classes. A mission class was then created by instantiating states and transitions using static object references in the code. This was a tedious and error-prone process since the state transitions and parameters where hard-coded. Figure 3 is an example of a hard-coded mission

```
protected State initStateMachine() {
   WaitForMissionSwitch waitForMissionSwitch = new WaitForMissionSwitch(this);
   PassGate straightAhead = new PassGate(this, config);
   FindBin find = new FindBin(this, config);
   DropWithSonar drop = new DropWithSonar(this, config);
   ChangeHeading changeHeading = new ChangeHeading(this, config);
   GoToRecovery goToRescue = new GoToRecovery(this, config);
   ChangeDepth changeDepth = new ChangeDepth(this, config);
   EndMission end = new EndMission(this);

   waitForMissionSwitch.setNextState(straightAhead);
   straightAhead.setNextStateOnMissionSwitchOff(end);
   straightAhead.setNextState(find);
   find.setNextStateOnMissionSwitchOff(end);
   find.setNextState(drop);
   drop.setNextStateOnMissionSwitchOff(end);
   drop.setNextState(changeHeading);
   changeHeading.setNextStateOnMissionSwitchOff(end);
   changeHeading.setNextState(goToRescue);
   goToRescue.setNextStateOnMissionSwitchOff(end);
   goToRescue.setNextState(changeDepth);
   changeDepth.setNextStateOnMissionSwitchOff(end);
   changeDepth.setNextState(end);
   end.setNextState(end);
   end.setNextStateOnMissionSwitchOff(waitForMissionSwitch);

   return waitForMissionSwitch;
}
```

Figure 3: Code snippet from a hard-coded mission file.

using only 8 states. A mission using only a small number of states was already complex enough to require careful code inspection to detect if transitions were left unassigned. Given that the code was usually modified on a desktop development environment and then uploaded and run on the remote target machine, the code/test/modify cycle was long.

Moreover, when the mission system was manually produced, it was difficult to modify it to reflect the frequent changes in the set of installed sensors since several files needed to be modified and recompiled. Changes in installed sensors were typically caused by late integration of components, degradation of the system capabilities caused by collisions or transportation damages or other causes out of the team's control. To reduce the amount of files to modify when a sensor was added or put offline, different mission scenarios were prepared by the team to take in account the most likely combinations of hardware and mission tasks. Furthermore, while stationed at the deployment site, different approaches were tested and the best strategies were refined to create the final mission. This process was usually done in a hurry and every stakeholder had a different idea of the best way to accomplish mission tasks. There was clearly a need for a tool that could help us to rapidly modify missions dynamically while reducing the chances of introducing defects.

Requirements and specifications

The development process selected by the SONIA AUV team for the creation of the Mission Editor is the Unified Process[7]. After careful analysis of the problematic situations, requirements for the Mission Editor were identified[8] during the inception phase of the project and are summarized in **Table 1: Requirement specification**.

Table 1: Requirement specification

ID	Requirement definition
RS1	Create, delete and modify a mission in offline mode (not connected to AUV4).
RS2	Modify, import and export a mission in online mode (connected to AUV4).
RS3	Add and remove a state to a mission.
RS4	Add and remove a transition to a state.
RS5	Assign a value to a state parameter.
RS6	Assign a value to a state parameter while the mission is being executed in online mode.
RS7	Validate the integrity of a mission and report errors and warning.
RS8	Identify the current state while a mission is being executed in online mode.
RS9	Display a visual representation for each state and transition.
RS10	Display the mission log information while a mission is being executed in online mode.

Design and Implementation

The main *raison-d'être* for the Mission Editor is to avoid defect introduction during the development of missions. Therefore, the Mission Editor is designed to take advantage of the Model Driven Development[9] approach. This approach systematically focuses on the development of models instead of code, scripts or paper documents. The model can then be validated using model-checking techniques[10], interpreted by analysis tools and serialized as a mission script in XML format. Since the mission is generated as a script from a validated model, the mission can be assumed defect-free as long as the integrity of the script file is preserved.

The Mission Editor architecture is designed to use the existing software elements such as communication interfaces and the finite state-machine mission engine. The architecture of the Mission Editor is defined in 3 segments as illustrated in Figure 4.

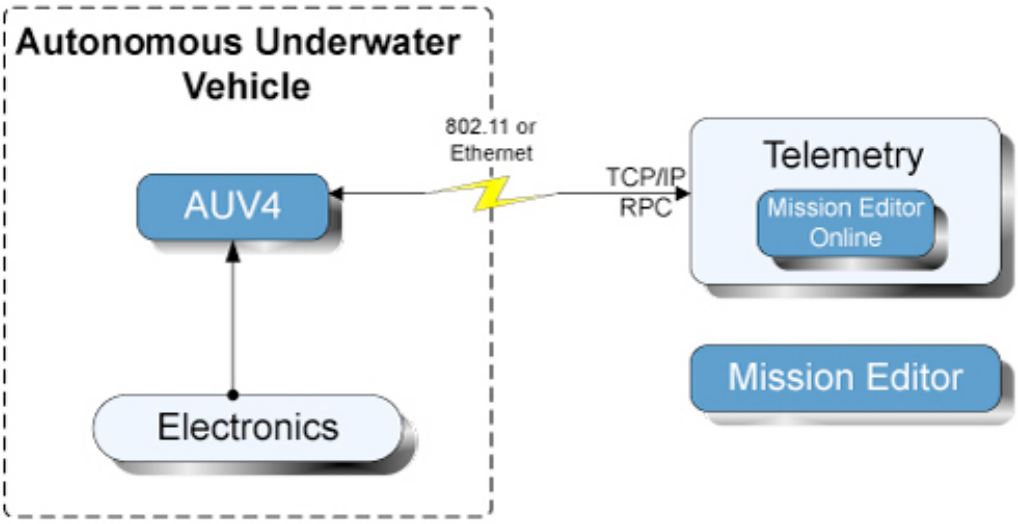

Figure 4: Architecture of the Mission Editor.

First of all, the state-machine and the reusable state building blocks have been reused from existing components of AUV4. The reuse of AUV4's mission system minimized the impact of the transition from hard-coded to dynamically-generated missions. The only change required was the creation of a serialization/deserialization module that loads the states and their parameters using reflection and indirect dynamic object instantiation from a definition stored in an XML mission file.

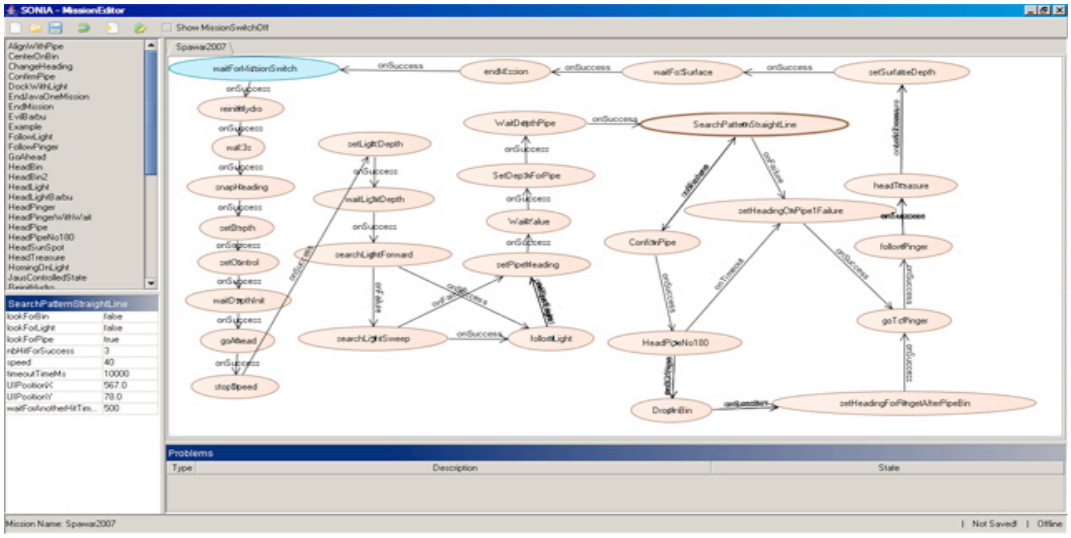

Figure 5: Mission Editor screen shot with the 2007 AUVSI mission.

The second element is the standalone Mission Editor, the main software which allows for the creation and modification of missions. The ME provides a graphical model-drawing interface based on the Open Source JGraph[11] library. Figure 5 shows a typical use-case of the ME interface at the 2007 AUVSI and ONR's AUV competition. States are represented as bubbles and the user can move, select or delete them. When a state is selected, the parameters for that state are displayed in the bottom left section for editing. To add a state, the user simply drags it from a list of every available state at the in the top left pane. To add a transition, the user clicks and drags the mouse from the origin state to the desired destination state. A popup window will then appear to allow the selection of the desired transition type. As defined in jABC[12], an MDD application, user roles can be divided in two groups: application experts, unmanned vehicle domain experts, and programming experts specialized in software programming. The simple interface of the Mission Editor enables domain experts to design or modify missions without prior knowledge of software programming. This greatly reduces the training required for autonomous vehicle support and deployment staff. Moreover, it allows technicians unskilled in programming to deploy the vehicle and create missions in the field using a PDA or rugged laptop and retrieve the vehicle at the end of its mission. All throughout the editing process, a real-time model checker displays a list of warnings and errors about the syntax and semantics of the state machine. Examples of detected problems include missing transitions, duplicate root state and missing mandatory parameters.

Thirdly, for all online tasks (actions requiring to be connected to AUV4), the Mission Editor Online (MEO) is invoked as a tool embedded inside the Telemetry interface of the SONIA AUV project. The integration inside the Telemetry interface allows the MEO to use the Telemetric communication interface interact with the AUV4 mission system. This integration permits the

MEO to identify the state currently being executed, display log messages from the mission system and to modify state parameters, on-the-fly, while the mission is being executed. To minimize code duplication, the same core code base is executed in both the standalone and the online versions of the ME.

Case study

To evaluate the benefits of using the Mission Editor versus hard-coding a mission, a simple back-and-forth mission has been implemented using both approaches. The mission is composed of seven states, ten state parameters and fourteen transitions. This is a very small mission in comparison to the 30 states required for the mission of the 2007 AUVSI and ONR's competition. The test subject is an experienced software developer that was involved in mission development for the past 4 years. He can be classified as both a domain and software-programming expert. The test results of the case study are included in **Table 2: Test result of the case study** .

Table 2: Test result of the case study comparison

	Time	**Fully functional**	**Number of defects**
With the ME	02m43s	Yes	0
Hard-coded	20m11s	Not the first time	1

This case study yields that using a MDD application such as the Mission Editor is significantly more time-effective than hard coding a mission offline in the system. Moreover, the mission created with the mission editor was defect-free from the start while the hard-coded version suffered from a typographical error that caused a defect and a failure in the mission. By using a model validation process, the Mission Editor can prevent defect from semantic, typographical and even some logical errors.

Future work and lessons learned

We are planning on adding several new functionalities to the Mission Editor in the upcoming months to further improve its capabilities. The addition of functionalities to modify transitions in the MEO for on-the-fly reconfiguration of state sequences will enable better flexibility in the online mode. Additionally the handling of XML mission file upload to the AUV from the MEO will accelerate the testing turnaround time.

Usage of finite state machine architecture allows the use of complex AI mechanisms such as expert systems or fuzzy logic controllers inside a so-called "macro-state". Therefore, the finite state machine might a desirable architectural choice for autonomous vehicle since its use facilitates the integration of Model-Driven-Development applications while allowing the use of other AI mechanisms for complex tasks. However, there are still technological limitations with the Mission Editor. For example, it is unable to detect design logic defects in missions. Only the validity of adherence to a model is checked and thus logic errors not taken in account by the model will remain undetected. A well-formed valid mission might still be using flawed logic from the designer to accomplish the task. It must be noted however that this shortcoming is reasonable since the level of validation performed by the ME tool is sufficient as to reduce the occurrence of the most common mission design issues.

Conclusion

Since 2006, the Mission Editor is a tool that enabled the SONIA AUV team to generate syntactically defect-free missions for AUVSI and ONR's International Autonomous Underwater Vehicle competition. This tool contributed to the success of the team, which ranked amongst the three-best in 2006 and 2007, and provided the platform needed for easy, on-the-fly, mission

reconfiguration. With the Mission Editor, even freshmen and engineering students in non-software fields have been able to create fully functional missions.

This demonstrates that end-users and field technicians with very little computer knowledge would be able to create missions for autonomous vehicles using Model-Driven-Development applications such as the Mission Editor. The use of applications based on that approach could greatly reduce the costs associated with the deployment of autonomous systems and the associated training time.

Acknowledgements

The SONIA AUV team would like to thank all of our sponsors for providing us with the tools and funding needed to build the AUV and its software. We acknowledge the amazing support we had from a large number of staff employees and students at École de technologie supérieure and from every team member of SONIA AUV. Thanks to Pierre Bourque and David Mercier for their support with requirements specification, Luc Trudeau for the early work on implementation and François St-Amant for the Mission Editor Online implementation. Last, but definitely not least, we would like to thank Tennessee Carmel-Veilleux for reviewing and editing this paper.

Contact information

SONIA AUV project
1100, Notre-Dame West, Office A-1320
Montréal, Quebec, CANADA
H3C 1K3,
http://sonia.etsmtl.ca

References

[1] Association for Unmanned Vehicle Systems International (AUVSI), & Office of Naval Research (ONR). (2007). *Final Mission Statement and Rules*. Retrieved from http://www.auvsi.org/competitions/water.cfm

[2] Devnani-Chulani, Sunita. (1998). *Modeling Software Defect Introduction*. Retrieved from http://sunset.usc.edu/publications/TECHRPTS/1998/usccse98-503/SoftModelIntro.pdf

[3] Girard, Anouck Renee. (1998). *A convenient state machine formalism for high level control of autonomous underwater vehicles*. M.S. dissertation, Florida Atlantic University, United States -- Florida. Retrieved July 29, 2007, from ProQuest Digital Dissertations database. (Publication No. AAT 1389012).

[4] Kanakakis, V., Valavanis, K.P., & Tsourveloudis, N. C. (2004). *Fuzzy-Logic Based Navigation of Underwater Vehicles*. Journal of Intelligent and Robotic Systems, 40(1), 45–88.

[5] Healey A. J., Marco D. B., & McGhee R. B., (1996) *Autonomous Underwater Vehicle Control Coordination Using A Tri-Level Hybrid Software Architecture*. Proceedings of the 1996 IEEE International Conference on Robotics and Automation

[6] Auslander, D. M., Ridgely, J. R., & Ringgenberg, J. D. (2002). *Control Software for Mechanical Systems: Object-Oriented Design in a Real-Time World*. Upper Saddle River: Prentice Hall PTR. 339 p.

[7] Jacobson, I, Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Boston: Addison-Wesley Longman Publishing Co. 463 p.

[8] Mercier, D., & Pageau, F., (2005). *Mission Editor Software Requirements Specification*. LOG410 Semester Project Report. Montreal, Canada. École de technologie supérieure. 33 p.

[9] Poole, John D. Hyperion Solutions Corporation. (2001). *Model-Driven Architecture: Vision, Standards And Emerging Technologies*. Proceedings of the ECOOP 2001 Workshop on Metamodeling and Adaptive Object Models.

[10] Mellon Clarke, E., Grumberg, Doron, & O., Peled, A. (1999). *Model Checking*. Cambridge: MIT Press. 314 p.

[11] JGraph Ltd. http://www.jgraph.com/

[12] Stefen, B., Margaria, T., Nagel, R., Jörges, R., & Kubczak, C., (2006). *Model-Driven Development with the jABC*. Proceeding of 2006 IBM Haifa Verification Conference.