Formal Methods for Automated Diagnosis of Autosub 6000

Juhan Ernits
School of Computer Science
University of Birmingham
Birmingham, UK
ernitsj@cs.bham.ac.uk

Richard Dearden
School of Computer Science
University of Birmingham
Birmingham, UK
rwd@cs.bham.ac.uk

Miles Pebody
National Oceanography Centre
University of Southampton
Southampton, UK
M.Pebody@noc.soton.ac.uk

Abstract

This is a progress report on applying formal methods in the context of building an automated diagnosis and recovery system for Autosub 6000, an Autonomous Underwater Vehicle (AUV). The diagnosis task involves building abstract models of the control system of the AUV. The diagnosis engine is based on Livingstone 2, a model-based diagnoser originally built for aerospace applications. Large parts of the diagnosis model can be built without concrete knowledge about each mission, but actual mission scripts and configuration parameters that carry important information for diagnosis are changed for every mission. Thus we use formal methods for generating the mission control part of the diagnosis model automatically from the mission script and perform a number of invariant checks to validate the configuration. After the diagnosis model is augmented with the generated mission control component model, it needs to be validated using verification techniques.

1 Introduction

There are vast areas of the Earth's seabed that have yet to be explored or studied. Recent discoveries of hot hydrothermal vents deep on mid oceanic ridges have revealed whole new ecosystems, many existing independently of energy from the sun. However the global extent and variety of these features is not known. The extensive and efficient exploration of these areas by the oceanographic science community is one example of the use of Autonomous Underwater Vehicles (AUVs). These vehicles are able to operate independently for several days with future developments extending this time span to several months. The Autosub 6000 project of the National Oceanography Centre in Southampton is a continuation of a successful series of projects, that takes the Autosub AUV to depths of up to 6000 m.

During more than 400 previous scientific missions, the predecessors of Autosub 6000 have suffered both near losses and one actual loss. In two cases the AUV has been recovered with a remotely operated underwater vehicle at significant expense. In once case the Autosub2 AUV was permantly lost 17Km under the 200m thick Fimbul Ice Shelf in the Antarctic. There are numerous cases of missions that have had to be aborted but where recovery was possible by the operations team and the attending support ship. Based on the experience of operating the Autosub AUVs a project to apply automated diagnosis and recovery methods for Autosub 6000 was initiated with the primary focus being on the detection of faults that may result in collisions with the seabed. Collision with the seabed is undesirable because it is has been demonstrated to have been one of the primary causes of vehicle loss. The approach we are taking is to use the Livingstone 2 (L2) diagnosis engine on Autosub. L2 is a discrete, model-based diagnosis system that is compositional, allowing models of individual comonents to be plugged together relatively easy to build larger models. We describe L2 in more detail in the next section.

Autosub 6000 [5] provides a configurable payload space of 0.5 m^3 that enables scientists to explore deep water with a range of sensor equipment. Typical missions have included temperature and salinity profiling, water sampling, seafloor mapping, seafloor photography and chemical analyses in areas ranging from tropical waters around Bermuda to under ice in the Arctic and Antarctic. The AUV needs to be reconfigured for each mission, meaning that the sensory equipment may be replaced and depth, position, mission control and abort parameters are changed. Furthermore, for each mission a custom mission script that details the waypoints which the AUV must pass and the actions to be performed must be produced.

E. Denney, D. Giannakopoulou, C.S. Păsăreanu (eds.); The First NASA Formal Methods Symposium, pp. 181-185

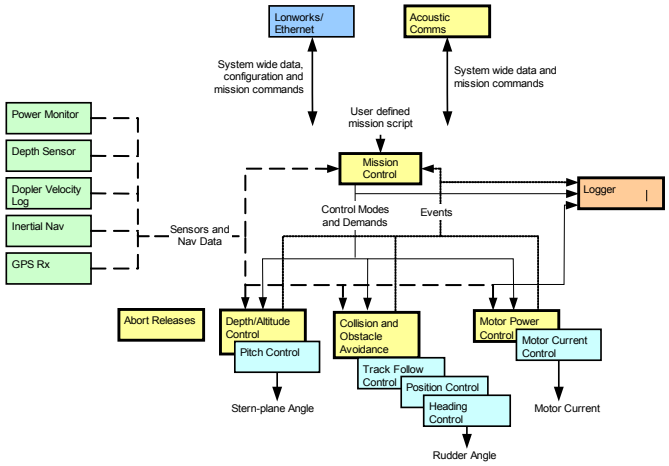

Figure 1: Architecture of the control system of Autosub 6000.

The fact that Autosub is frequently reconfigured between missions means that it is fundamentally different from the space missions that L2 has previously been used for. As well as changes to the payload of the vehicle, each mission has its own configuration script that sets a number of parameters such as maximum depth and limits of the dive plane angles, and the activities carried out will also vary from mission to mission. This leads to an important challenge: How can we update the diagnosis model quickly and correctly between missions? We believe that formal methods provide part of the solution to this challenge by allowing the mission script and configuration script to be sanity checked before use and allowing diagnosis models to be automatically generated from the scripts that can be integrated with the hand-built models of the components.

Generating parts of the diagnosis model automatically either requires proving the model generation correct or applying some verification techniques to the resultant model to raise the level of confidence that integration of the automatically created components with the rest of the model will be reliable.

It has been suggested by Kurien and R-Moreno [3] that the risks and costs of applying model-based diagnosis outweigh the expected value. We believe that while this may be true for space missions, applications such as Autosub are much more compelling domains for model-based diagnosis and that the application of formal methods for automatic generation and validation of the diagnosis model can significantly improve the diagnostic power of the system.

2 Automated diagnosis for Autosub 6000

Our diagnosis approach is based on Livingstone 2 [2, 9], a model-based diagnosis tool developed at NASA's Ames Research Center that has been deployed in several applications. Livingstone works using a discrete event model of the system, and a constraint solver to detect inconsistencies between the model and the system and finds explanations for them.

A Livingstone model is composed of components (which may map onto physical components), connections between components and constraints. A component is specified by variables, with a set of discrete, qualitative values for each variable in its local nominal and failure modes. For each mode, the model specifies the components' behavior and transitions.

Most of the subsystems and components of the Autosub AUV will be modelled ahead of time, i.e. before the AUV is sent to carry out concrete tasks. Fig. 1 illustrates the network variable based control system used in the AUV, based on the principles explained in [6]. The Mission Control component

communicates with a number of other components in the system and could thus be a valuable resource for improving the precision of diagnosis. The challenge is to automatically create models representing the information in the mission script and the configuration script and to integrate them with the hand-built component models. This has two advantages which can substantially improve the reliability of the system. First, the mission and configuration scripts constrain the behaviour of the vehicle so diagnosis is easier since the diagnosis engine knows more precisely what should be happening. This means that, for example, the diagnosis engine can detect when the vehicle differs from its allowed behaviour, perhaps by diving too deep. The second advantage is that the diagnosis model can be used as an abstract simulation of the system, so it can be used to perform a number of tests on the mission script, for example to check whether the vehicle can actually carry out the mission successfully if all systems remain nominal. This is similar to the approach taken by Livingstone PathFinder [4].

3 Mission Control

The mission control node [7] is the Autosub component that executes the mission script. The script is an event driven sequence of commands in a text file that is compiled and then downloaded to the mission control node as part of the mission preparation procedure. When a scripted mission event occurs the next commands are issued to the various motor, actuator and sensor control nodes. The mission control then iterates to the next element in the mission script and waits for the next specified event. In addition to the main mission script the operator may specify up to two alternative endings to the mission in the form of mission termination scripts. These can be triggered by a configurable set of events and provide an alternative ending to a mission.

A mission script contains a list of event triggered actuator mode and demand settings contained in a basic structure format called a mission element:

```
when( event_0, event_1, event_2)
    mode_0( demand_0),
    mode_1( demand_1),
    ...
    mode_n( demand_n);
)
```

With actual events modes and demands this might look like:

```
when( GotPosition, ElementTimeout),
    SetElementTimer( 0:0:30:0),             // Time argument 0 days, 0 hours, 30 minutes, 0 seconds.
    MotorPower( 320),                       // Set motor to run at 320W
    PositionP( N:52:30.0, W:15:0.0),        // Position control navigation
    Depth( 2000);                           // Depth control 2000m
)
```

Built into the mission controller are a number of *event processing tasks*. These range from simple input signal polling to the maintenance of internal timers and arbitration between inputs for starting and stopping a mission. Mission control events in general may be divided into events that are generated within the mission control node itself and those that are received from other nodes as network variable updates. A disjunction of up to three events may be specified to trigger a mission element.

The position, depth and motor controllers in the Autosub take commands in the form of an *operation mode setting* and a *demand*. For example the position controller has a mode called ”heading” that is given a demand in degrees of the compass. The mission control outputs a data structure that communicates the required mode and demand to the respective controller. In addition auxiliary modes and demands can be sent to payload sensors. The controllers in the Autosub act on the last received mode and demand. There are a small number of modes and demands that affect the behaviour of the mission control. These enable the setting of parameters that define the internal generation of mission events, in particular depth thresholds and timeout values for mission elements.

A number of features including obstacle avoidance, track following and beacon homing do not form part of the Mission Script. Their effect is at a level below the operator programmer interface and event states provide a means of programming mission script responses to them.

4 Construction and validation of the diagnosis model

The first step in the automatic construction and validation of the diagnosis model is to formalise the general requirements that apply to all mission scripts and configuration. For example, we have identified a number of requirements such as "the configuration variable `ncSafeMaxDepth` has to be less than abort weight release max depth" and "the variable `ncMinDepth` has to be less than the maximum depth" that are domain specific but should hold globally for all mission configurations. Previous experience shows that the configuration files that accompany mission scripts can have errors and we expect even such simple checks to improve the overall robustness of the system.

While constructing the diagnosis model from the mission script we apply analysis to the mission script to make sure that demands set in the mission script do not violate any of the general domain specific requirements or configuration parameters. The mission script is comprised of two different kinds of statements, those which block awaiting some particular event (a `when` statement), and those which set operation modes and demands after an event triggering the next step of the mission is observed. So, an example of a safety check performed during the model generation is to make sure a mission script does not have a demand exceeding the maximum allowed depth for the mission.

Each of the `when` statements is converted into two states, the state where the `when` statement is waiting for the triggering event and an intermediate state where the body of the `when` statement is executed. Such a setup allows the diagnosis model to monitor mission progress and detect if the mission control component has omitted broadcasting any demands listed in the mission script. After the demands have been seen by the diagnosis model, the model moves on to a state waiting for an event that triggers the next `when` statement. The mission control component of the diagnosis model is a linear string of such pairs of states. Those states where alternative termination scenarios are allowed can branch to similar pairs of states specified by appropriate termination scripts.

Automatic generation of parts of the diagnosis model will necessarily raise questions about the correctness of the resultant model. While proving the model generation correct would be one option, we will initially use techniques such as Livingstone PathFinder [4] and Livingstone model verification [8] to verify the properties of the resultant diangosis model.

One possible way to validate the resultant model after the mission control component of the diagnosis model has been generated is to create an example mission scenario from the excerpts of the data of previous missions matching the constraints and demands of the appropriate step of the mission script. This provides a way to have dry runs of successful scenarios and also various error scenarios and use an approach like Livingstone Pathfinder to check if the diagnoses of L2 subsume the faults that have been introduced into the scenario script. One research challenge is to see if we can build a useful abstract model of the behaviour of the AUV that encapsulates a large number of concrete scenarios that can be used to prove that no false positives can occur.

The diagnosis model is coupled with the actual variables of the system via monitors which convert continuous variables into discrete values. While the discussion of how exacly the continuous variables are tracked is out of scope of this paper, another of our research challenges is how to model such monitors mathematically for establishing the correctness of their implementations and to further establish the adequacy of the diagnosis model given the properties of the input signals.

Automation of verification and proof procedures in such applications is of vital importance as mission scripts are built and modified during research cruises and it is likely that there is neither time nor expertise

for manually validating or building the diagnosis models at sea between subsequent missions.

Another of the open challenges is the *verification of the diagnosis engine* itself. Ideally all code that runs on an autonomous platform should be at least proved correct for safety policies such as memory safety. There are technologies and tools that support such proofs based on Hoare style annotations of pre- and postconditions and loop invariants for establishing various safety policies [1]. In addition, with recent advances in verification tools, an engine written in C++, such as Livingstone 2, with additional support for continuous variables is a verification challenge. The L2 code has been empirically tested to be flight safe, but modifying it will quite likely introduce new issues which can be discovered with verification techniques. We invite interested parties to contact us if there is interest for pursuing this path.

5 Conclusions

We have presented a work in progress on applying formal methods in the context of automated diagnosis and recovery to Autosub 6000 AUV. The work includes several research challenges that are relevant to the formal methods community. One of the challenges is automatic generation of the mission control component of the diagnosis model from mission scripts and configuration. Success of such generation can only be guaranteed by proving the resultant model to satisfy formalised safety (and liveness) policies using verification techniques. Another open challenge is proving the Livingstone 2 engine and the hybrid monitors that we add to the engine to be correct regarding a number of safety and liveness policies.

Acknowledgements. This work was funded by the UK Natural Environment Research Council as part of grant NE/F01256X/1, Automated Diagnosis for Fault Detection, Identification and Recovery in Autosub 6000.

References

[1] E. Denney and B. Fischer. Correctness of source-level safety policies. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 894–913. Springer, 2003.

[2] S. C. Hayden, A. J. Sweet, and S. Shulman. Lessons learned in the Livingstone 2 on Earth Observing One flight experiment. In *Proc. AIAA 1st Intelligent Systems Tech. Conf., Am. Inst. Aeronautics and Astronautics*, 2004.

[3] J. Kurien and M. D. R-Moreno. Costs and benefits of model-based diagnosis. *Aerospace Conference, 2008 IEEE*, pages 1–14, March 2008.

[4] A. E. Lindsey and C. Pecheur. Simulation-based verification of autonomous controllers via Livingstone PathFinder. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2004.

[5] S. McPhail, M. Furlong, V. Huvenne, and P. Stevenson. Autosub6000: Results of its engineering trials and first science missions, 2008. presented at the 10th Unmanned Underwater Vehicle Showcase, The National Oceanography Centre, Southampton, UK.

[6] S. D. McPhail and M. Pebody. Navigation and control of an autonomous underwater vehicle using a distributed, networked control architecture. *The International Journal of the Society for Underwater Technology*, 23:19–30, 1998.

[7] M. Pebody. The contribution of scripted command sequences and low level control behaviours to autonomous underwater vehicle control systems and their impact on reliability and mission success. *OCEANS 2007 - Europe*, pages 1–5, June 2007.

[8] C. Pecheur and R. G. Simmons. From Livingstone to SMV. In *FAABS '00: Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers*, pages 103–113, London, UK, 2001. Springer-Verlag.

[9] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, pages 971–978, 1996.