Monitoring IVHM Systems using a Monitor-Oriented Programming Framework

Sudipto Ghoshal[1], Solaiappan Manimaran[1], Grigore Rosu[2], Traian Florin Serbanuta[2], and Gheorghe Stefanescu[2]

[1]Qualtech Systems Inc., 100 Great Meadow Rd Ste 603, Wethersfield, CT 06109-2355

Email: [1]{sudipto,mani}@teamqsi.com

[2]Department of Computer Science, University of Illinois at Urbana-Champaign
201 N. Goodwin, Urbana, IL 61801

Email: [2]{grosu,tserban2,stefanes}@cs.uiuc.edu

Abstract

We describe a runtime verification approach to increase the safety of IVHM systems by an integration of TEAMS models and Monitor-Oriented Programming (MOP). The TEAMS model is used to automatically extract relevant runtime information from the controlled system by means of events. This information is passed online to the MOP engine, allowing to verify complex temporal properties and to discover running patterns which are of interest in detecting and preventing faulty behaviors.

1 Monitor-Oriented Programming (MOP)

MOP [3, 2, 1] has its roots in a runtime verification system, PathExplorer (PAX) [7, 6], developed jointly with former NASA colleagues. PAX has found mission critical errors in NASA software. In a recent OOPSLA'07 paper [3], it was shown that the MOP framework can monitor large programs against complex parametric temporal specifications at a typically unnoticeable runtime overhead, rarely above 10%.

Many properties can be monitored in parallel in MOP. The execution trace against which the various properties are checked is extracted via automatic code instrumentation from the running program as a sequence of events – state snapshots. Events produce sufficient information about the concrete program state in order for the monitors to correctly check their properties. A monitor is typically interested in a subset of events.

In MOP, the runtime monitoring of each property consists of two orthogonal mechanisms: *observation* and *verification*. The observation mechanism extracts property-relevant and filtered system states at designated points, e.g., when property-specific events happen. The verification mechanism checks the obtained abstract trace against the (monitor corresponding to the) property and triggers desired actions in case of violations or validations. Observation and verification are therefore independent: the algorithm used within the monitor does not affect how the execution is observed, and vice versa. MOP is a highly configurable and extensible runtime verification framework. Depending upon configuration, the monitors can be separate programs reading events from a log file, from a socket or from a buffer, or can be in-lined within the program at the event observation points.

Properties can be specified in MOP by means of *logic plugins* which essentially encapsulate and standardize monitor synthesis algorithms for various formalisms of interest. Here are several logic plugins currently provided by MOP:

— Design by Contract: A JAVA logic plugin for JASS has been implemented in MOP. JASS supports the following types of assertions: method pre-conditions and post-conditions, loop variants and invariants, and class invariants. Pre-conditions must be satisfied by the caller, but the callee is in charge of checking them. Post-conditions need to be satisfied at the exit of a method.

— Temporal Logics: Temporal logics proved to be indispensable expressive formalisms in the field of formal specification and verification of systems. Many practical safety properties can be naturally expressed in temporal logics, making them desirable specification formalisms in the MOP framework. Login plugins for both future and past time temporal logics are available.

— Extended Regular Expressions: Software engineers and programmers understand easily regular patterns, as shown by the interest in and the success of scripting languages like PERL. We believe that regular expressions provide an elegant and powerful specification language also for monitoring requirements, because an execution trace of a program is a string of states. Extended regular expressions (ERE) add complementation to regular expressions, allowing one to specify patterns that must not occur. An ERE logic plugin is available in MOP.

2 TEAMS Models

TEAMS [5] is a model-based diagnosis system. The TEAMS model of a system is a dependency model capturing relationships between failure modes of the system and their observable effects. QSI's TEAMS application provides a user-friendly environment for developing dependency models of complicated aviation systems, while allowing the specification of several additional practical aspects about the system that are required by the run-time inference engines, namely TEAMS-RT and TEAMATE to provide efficient diagnosis.

QSI's TEAMS Tool Set [5] consists of four software applications: TEAMS Designer, TEAMS-RT, TEAMATE and TEAMS-RDS. Underlying these tools is the model and diagnostic data knowledge base called TEAMS-KB. As said, the TEAMS model of a system is a dependency model that captures relationships between failure modes of the system and their observable effects. The model is created in TEAMS Designer, or imported into TEAMS Designer from other data capture environments, and then analyzed and converted into run-time versions for export to the run-time reasoners TEAMATE and TEAMS-RT. The TEAMS Designer application provides a user-friendly graphical environment for developing dependency models of systems while allowing the specification of several additional practical aspects about the system that are required by the run-time inference engines to provide efficient diagnosis. It does so by allowing the modeler to specify cause-effect dependencies using a hierarchical, multi-layered (multi-signal), directed graph representation of the system. In this graphical representation, the system's physical elements (subsystems, components, etc.) are represented as module nodes; the physical locations, where the measurements of the system's performance or other attributes are made for the determination of a failure or anomalous condition, are represented as test-point nodes; and the dependency relationships are represented as directed links (edges).

Once a TEAMS model specification is complete, a reachability analysis can be performed in TEAMS to internally generate the dependency matrix model of the system subject to analysis constraints specified by the user. When the dependency-matrix model is available, diagnosis becomes the process of using the dependency relationships and the observed failures or anomalies to infer their possible causes. The functional requirements of the reasoning engine that performs the diagnostic inference depend on the manner in which the observations about the system's state become available. The TEAMS-RT inference engine processes failure events (exceedances, built-in

test failures, performance anomalies, etc.), as they become available. It uses the data to infer the status of the root causes (the identification of one or more component faults). Thus, TEAMS-RT is appropriate for processing onboard data that is either received in real time or downloaded post-mission/operation. The TEAMATE diagnosis reasoner not only performs inference of component health status, but also computes an optimal sequence of (active) tests that needs to be performed for fault isolation, given the current inferred health status, the allowable set of tests, and any precedence constraints on the tests. Thus, TEAMATE is appropriate for ground-based deployment where troubleshooting is performed interactively.

3 Monitoring TEAMS Specifications using MOP

We report partial work on developing a TEAMS logic plugin for MOP, which will automatically generate monitoring code from the TEAMS temporal specifications.

In a system where requirements are monitored and recovery code is executed when violations are detected, the correctness of the entire system relies only on the guarantees provided by the monitor and the recovery code. Verification of the entire system can be decomposed into checking the correctness of the monitor and of the recovery code, which are expected to be much simpler and cheaper than verifying the original program. Currently, we are working on developing an integrated framework for IVHM system monitoring, control and verification. In this framework, the TEAMS tools will be used to capture the requirements specifications of the flight system. The MOP framework, extended with a TEAMS logic plugin, will process the captured system specifications and generate monitoring code automatically. The generated monitors will check the flight system at runtime via the monitoring mechanism provided by TEAMS, steering the system if failures are detected. This way, system models and runtime verification together are expected to form a solid foundation for developing reliable aviation systems.

References

[1] F. Chen, M. D'Amorim, and G. Rosu. A formal monitoring-based framework for software development and analysis. In: *Proc. ICFEM 04*, volume 3308 of LNCS, 2004, pp. 357–373.

[2] F. Chen, M. D'Amorim, and G. Rosu. Checking and correcting behaviors of Java programs at runtime with Java-MOP. In: *Proc. Workshop on Runtime Verification (RV 05)*, ENTCS Vol. 144(4), 2005, pp. 3-20.

[3] F. Chen and G. Rosu. Mop: An efficient and generic runtime verification framework. In *Proc. OOPSLA'07*, ACM Press, 2007, pp. 569-588.

[4] F. Chen, T.F. Serbanuta and G. Rosu. jPredictor: A predictive runtime analysis tool for Java In: *Proc. ICSE'08*, to appear.

[5] S. Deb, S. Ghoshal, V. N. Malepati, and D. L. Kleinman. Tele-diagnosis: Remote monitoring of large-scale systems. In: *Proc. The IEEE Aerospace Conference*, 2000.

[6] K. Havelund and G. Rosu. Monitoring Java programs with Java PathExplorer. In: *Proceedings of the 1st Workshop on Runtime Verification (RV 01)*, ENTCS, Vol. 55, 2001.

[7] K. Havelund and G. Rosu. Monitoring programs using rewriting. In: *Proceedings, International Conference on Automated Software Engineering (ASE 01)*, IEEE, 2001, pp. 135-143.