Automatic Derivation, Integration and Verification of Synchronization Aspects in Object-Oriented Design Methods

Quarterly Report
DARPA Order K203/AFRL Contract F33615-00-C-3044 [*]

Matthew Dwyer John Hatcliff Masaaki Mizuno Mitch Neilsen
Gurdip Singh

January, 2001

In this project, which we call S^2aVES (Specification, Synthesis and Verification of Embedded Systems), we investigate solutions to the problem of implementing robust, efficient global synchronization policies in multi-threaded, distributed and object-oriented embedded software systems.

Synchronization is an important aspect in the design of complex embedded systems. Object-oriented analysis and design techniques have been used successfully to design and maintain large software systems, but multiple shortcomings in their treatment of synchronization aspects limit their effectiveness in many applications. We propose to advance the use of object-oriented methodologies to design high-assurance embedded systems by addressing the following challenges:

- high-level, modular specification of global, cross-cutting synchronization aspects,
- automatic derivation and weaving (i.e., integration) of verifiable synchronization code into core functional code,
- automated verification of critical safety and liveness properties of woven embedded code.

To meet these challenges, we are developing techniques, tools and methodologies that support high-level specifications of synchronization aspects, derivation and weaving of aspect code into core functional code written in languages commonly used for embedded applications.

Foundations

We have developed a methodology to develop synchronization code based on the global invariant (GI) approach in the context of the Unified Process in UML [3]. In our methodology, the component code is developed by the Unified Process and synchronization code is developed as aspect code which is later woven into the component code to produce complete object-oriented programs in languages and/or platforms based on semaphores, general monitors (such as C++ in POSIX pthreads), Java synchronized blocks, and active monitors.

Progress previously reported

First report

This quarter

We have made progress in four distinct areas that move the project toward the realization of its goals: specializing it to a common embedded system platform, generalizing it to treat a broad set of embedded system's programming idioms, developing the automated reasoning necessary to support the synthesis of efficient implementations from the high-level specification, and extending the verification techniques available to reason about woven code.

[*]Kansas State University. ({dwyer, hatcliff, masaaki, neilsen, singh}@cis.ksu.edu)

Platform Specialization To apply our methodology to development of CAN-based real time control systems, we are developing translations of high-level invariants for CAN-based environments. We are studying the following types of solutions:

Distributed shared memory-based solution CAN has a property that all messages are broadcast and received by all the processors in the same order. Therefore, it is feasible to implement a middleware to realize linearizability based distributed shared memory. That is, every processor holds a copy of all shared memory variables. Using this system, existing translations to semaphore or monitor based programs can be used directly. Ye Su, a graduate student, is currently implementing this solution.

Point-to-point message passing solution In many cases, synchronization involves sending messages (signals or notifications) to specific processes rather than to all processes, for which point-to-point message passing may prove more efficient. We are currently exploring modifications of distributed event-service to implement the constraints.

Generalization The global invariant approach seeks to lift the specification of synchronization and scheduling aspects of program activities to a very high level. To make such specifications usable to practitioners we have identified a set of global invariant patterns.

We have been expanding the set of global invariant patterns to cover more application problems. A group of application problems have been collected to serve as a testbed for our approach. We have been able to specify all of the synchronization required of those problems as the composition of global invariant patterns. We are preparing a paper to report on the breadth of synchronization problems that can be handled by our approach.

We have been generalizing our methodology in to work for sequential event-driven systems. The basic approach to adapting the methodology has been outlined and work is proceeding on detailing the mappings to several common event-driven frameworks.

We are working to incorporate functionality that is found in real-time, priority scheduling systems in our global invariant methodology. This work is in the initial study phase.

Automated Reasoning Our synchronization code synthesis system requires us to generate guards based on weakest preconditions at entrance and exit of critical regions. For efficiency, these guards should be simplified if feasible. We have developed techniques for guard simplification using a package of decision procedures for first-order arithmetic. These techniques have been implemented in a prototype tool described in the following section.

Verification Despite the fact that in our approach synchronization code is derived systematically, it is usually the case that additional system requirements remain to be demonstrated in the final system. One approach to achieving this is to apply model checking to reason about the correctness of safety and liveness requirements of a model of the system that is derived from its source code. Model checking is notoriously expensive and it will be necessary to aggressively abstract the source code to enable practical checking of desired requirements.

We have developed an approach, and associated tool-support, for systematically abstracting Java programs in such a way that verification of requirements is preserved [2]. One difficulty with such property-preserving abstraction approaches is that they may produce false negative results. We have developed an approach, and associated tool-support, for analyzing such false negative results to determine if they indicate actual program defects or are simply an artifact of an overly approximate abstraction [4]. These methods have been shown to be effective on several multi-threaded Java applications, including a real-time scheduler for embedded avionics applications.

Applications

Progress previously reported

First report

This quarter

One application for this methodology is in developing correct software for Seaborne and Airborne Target Systems. David Purdy, as director of the Surface Targets Team of the Naval Air Warfare Center Weapons Division, at Point Mugu, California, has developed a Common Digital Architecture (CDA 101) for internetworking target system controllers. The architecture provides abstractions that can be used to simplify the task of programming target control systems on a real-time Controller Area Network (CAN). They have completed the code for the Seaborne Target 2000 System (ST2000). We have studied the code to identify synchronization patterns in the code so that we can re-engineer the application using our methodology and compare the difference in performance. Although we expect little difference in processor utilization, etc., our methodology generates correct code by construction. This application will provide us with a proof of concept, and allow us to assess the quality benefits and performance tradeoffs.

Since most complex systems are comprised of many smaller subsystems, it will also provide us with a small subsystem within a larger Naval Training System. We plan to continue collaboration with the Surface Targets Team as they move into the next phase and focus on Airborne Targets. They have graciously provided us with both software and hardware to help us test our proof of concept.

Tools

Tool-support for the methodology will focus first on the code synthesis and weaving problems and later on the integration with UML notations. Code synthesis and weaving for Java is being implemented in the context of the Bandera [1] system. Code synthesis and weaving for C-based CAN systems is being implemented with a specially built back-end.

Progress previously reported

First report

This quarter

Bandera uses a Java byte-code-based intermediate form. To generate Java source code one must *decompile* the byte-codes. This is a difficult problem due to the fact that the process of compiling Java to byte-codes typically throws away information that is irrelevant for the purpose of generating executable code (e.g., source type information is lost). We have developed a Java decompiler that is able to overcome those obstacles in all but a few cases (e.g., complex boolean expressions). Work is underway, by graduate student Robby Joehanes, to enrich the information stored in the intermediate form by the compiler to lift those limitations.

To be effective, the development of the coarse-grained (i.e., await statement based) solution for expressing synchronization patterns must be automatic. To achieve this we have been experimenting with decision procedures in the SVC and PVS systems that are capable of reasoning about logical formula with counter variables. These experiments have shown that it is possible to handle the formula that arise from specifications in the current global invariant language.

These experiments have led to the development, by graduate student William Deng, of a prototype tool to demonstrate the central functionality of our proposed synchronization code synthesis tool. This prototype takes as input a Java program which implements core functionality of the desired system while omiting all synchronization code. An auxiliary input (a) specifies the designated critical regions of the supplied code, and (b) gives an global invariant that states occupancy constraints on the declared regions. The invariant constraints are expressed using two integer variables R_{in} and R_{out} for each region R; R_in and R_out count the threads entering and exiting R, respectively.

Using a package of decision procedures, the tool automatically synthesizes a high-level synchronization solution expressed in a language of guarded commands. This high-level solution is then translated automatically to a Java-based solution. The Java-based solution provides methods that are called at the beginning and ending of each critical region to ensure appropriate synchronization. The core functional code and the synthesized synchronized code are woven together by inserting these region delimiting methods into the core functional code.

Since we wish to target multiple execution platforms we have begun to design a common intermediate representation for the coarse-grained solution. This will allow both the Java and C-based synthesis and weaving systems to exploit the coarse-grained solution engine described above. Work will proceed on this during the next quarter.

Collaborative efforts

Previous progress:

First Report

This quarter:

Naval Air Warfare Center

Over the past several years, we have participated in working groups on the development of High-Level Protocol Standards for Controller Area Networks. An emerging standard for marine craft by the National Marine Electronics Association is NMEA 2000. It is intended to replace a long standing protocol for Global Positioning Systems (GPS), namely NMEA 0183 – virtually all GPS receivers currently support NMEA 0183, and many new models now support NMEA 2000. Beyond defining a GPS Protocol, the standard also defines message transmission between all types of standard sensors, actuators, and controllers on a Controller Area Network. Such networks are used for Seaborne and Airborne Target Systems. David Purdy, is also a member of the NMEA 2000 Working Group. Consequently, we have on-going discussions regarding the development of a Common Digital Architecture (CDA 101) for internetworking target system controllers. We are in the process of re-engineering their hand-coded applications using our methodology. We plan to continue collaboration with the Surface Targets Team as they move into the next phase and focus on Airborne Targets. They have graciously provided us with the support needed to test our methodology.

Rockwell Collins

We have initiated contact with a group at Rockwell Collins Advanced Technology Center that is focusing on building next generation embedded avionics software in Java. They are working on several systems that might serve as case studies for our approach (e.g., an adaptable software radio applications). More generally they are interested in working towards the deployment of middleware components in an avionics system. We believe that some of our specialization techniques might be applicable to improving the efficiency of such components. Furthermore, our synchronization specification and synthesis approach could target such components. In the long-term we imagine that our approach could support multiple targets simultaneously thereby enabling common specification of system synchronization for a heterogeneous distributed embedded system. Ongoing discussions are serving to identify specific technical aspects of their projects that might be suitable for our tools and methods as they mature.

References

[1] J. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Păsăreanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd International Conference on Software Engineering*, pages 439–448, Limerick, Ireland, June 2000. IEEE Computer Society.

[2] M.B. Dwyer, J. Hatcliff, R. Joehanes, S. Laubach, C.S. Păsăreanu, Robby, W. Visser, and H. Zheng. Tool-supported program abstraction for finite-state verification. In *23rd International Conference on Software Engineering*, Toronto, Canada, May 2001. IEEE Computer Society.

[3] Masaaki Mizuno, Gurdip Singh, and Mitchell Neilsen. A structured approach to develop concurrent programs in UML. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 451–465. Springer, 2000.

[4] Corina S. Păsăreanu, Matthew B. Dwyer, and Willem Visser. Finding feasible counter-examples when model checking abstracted Java programs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, April 2001.