# A 'substrate' for dynamically defined behavior in high assurance, embedded, real-time systems

Gary W. Daugherty, Rockwell Collins, gwdaughe@rockwellcollins.com

**Abstract-** This paper proposes development of a standard 'substrate' for dynamically defined behavior that supports the run-time definition of completely new and previously unanticipated behavior. This stands in contrast to most embedded, hard real-time systems that only support switching between previously defined states and alternatives. The proposal supports the ability to define new behaviors both as end user defined scenarios (which have a run-time representation), and as cross cutting strategies and policies (in an aspect-oriented sense). In the context of systems that learn, it can be used to exchange experience (learned strategies and optimized scenarios). In the context of command and control systems it can be used to issue 'orders' corresponding to radical, and unexpected changes in mission. The approach will initially be targeted to a networked Java platform for high assurance, embedded, hard real-time systems. Support for an explicit state based run-time representation allows us to easily analyze system behavior using light weight formal methods such as model checkers. The incremental specification of new behavior allows us to explore similar incremental approaches to analysis. Potential DARPA applications include the use of the platform as a mission computer on an unmanned combat vehicle or, more generally, as a node in a highly adaptable next generation tactical Internet.

## 1 Introduction

This paper proposes development of a standard 'substrate' for dynamically defined behavior that supports the run-time definition of completely new and previously unanticipated behavior. This stands in contrast to most embedded, hard real-time systems that only support switching between previously defined states and alternatives. The proposal supports the ability to define new behaviors both as end user defined scenarios (which have a run-time representation), and as cross cutting strategies and policies (in an aspect-oriented sense). In the context of systems that learn, it can be used to exchange experience (learned strategies and optimized scenarios). In the context of command and control systems it can be used to issue 'orders' corresponding to radical, and unexpected changes in mission. The approach will initially be targeted to a networked Java platform for high assurance, embedded, hard real-time systems. Due to the compact representation of new behavior, it should only require communication over low speed data links. Support for an explicit state based run-time representation allows us to easily analyze system behavior using light weight formal methods such as model checkers. The incremental specification of new behavior allows us to explore similar incremental approaches to analysis. Potential DARPA applications include the use of the platform as a mission computer on an unmanned combat vehicle or, more generally, as a node in a highly adaptable next generation tactical Internet[1].

## 2 Motivation

In a military context, the ability to dynamically adapt and optimize the behavior of groups of real-time, embedded systems is a fundamental part of battlefield wide attempts to radically improve overall efficiency and effectiveness. In a commercial context, it offers a similar ability to radically alter and optimize the behavior of large systems of networked, embedded, real-time systems (such as power grids, avionics and air traffic control systems) while they remain operational. In either context, we require the ability to introduce completely new and previously unanticipated behaviors, strategies, and policies rather than only switch, at run time, between previously defined alternatives. We also require the intelligent processing of large amounts of data, e.g., to support advanced Command and Control ($C^2$) applications via a tactical version of the W3C *semantic* Web. These capabilities must be provided in a hard real-time networked environment in which bandwidth and other resources are at a premium and behavior is often safety critical.

The short term result of this effort is expected to be a CORBA capable real-time Java platform with hardware and software to support the exchange and run-time composition of "pieces" of mission specific behavior (in the form of new mission specific scenarios, new cross cutting policies and strategies, and new supporting functionality). Making this a part of JTRS and the mission computers of unmanned vehicles should provide this capability to all the DoD services on a time line consistent with these programs.

[1] Such as that proposed by the Multifunctional On-the-Move Secure Adaptive Integrated Communications (MOSAIC) program or made possible using the Joint Tactical Radio System (JTRS)

The long term result of this effort is expected to be a standard for the run-time adaptation of embedded, hard real-time, high assurance systems, first as part of the Java Community Process, then later at a language independent level in terms of the Object Management Group's Action Semantics.

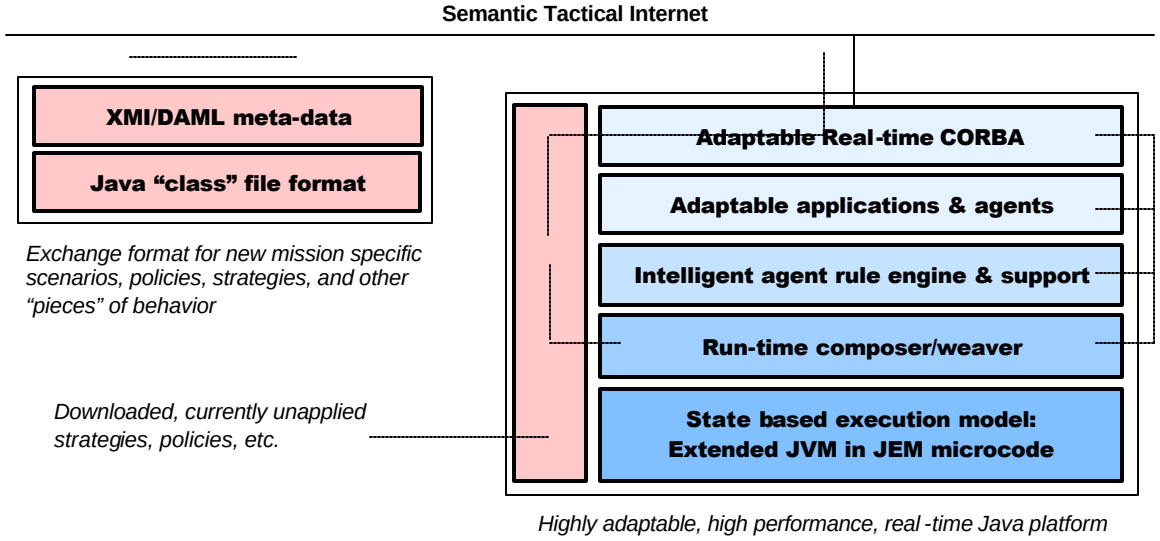


*Figure 1. Demonstration platform*

## 3 Proposal

**Java for hard real-time, high assurance applications.** Java provides a popular purely object-oriented language, a portable execution environment, an extensive collection of libraries and COTS software, support for run-time reflection, and the ability to dynamically download classes via the Internet. These are characteristics that have helped make it a popular language for mobile intelligent agents, aspect oriented programming and other research. Current Java systems, however, are often characterized by poor performance and are not considered suitable for hard real-time, high assurance applications. Native compilation can be used to overcome performance problems, but only by sacrificing the characteristics that make Java attractive, such as the binary portability required to support dynamic adaptation of mission specific behavior.

To resolve these issues, we propose to initially build upon work by Rockwell Collins and aJile Systems in the development of Java processor technology, then extend the approach to include other Java processors and accelerators, and to partitioned and secure processing systems[2]. The aJile JEM processor provides a microcoded implementation of a real-time version of the Java Virtual Machine (JVM) with extended byte code support for Java threads and I/O (Figure 1 Extended JVM in JEM microcode)[3]. A writeable control store supports the microcode definition of additional custom byte code instructions. The JEM is based on an earlier stack based processor developed by Rockwell Collins, the AAMP, that has been widely used in Boeing commercial aircraft. As a result, it has a pedigree in high assurance, hard real-time, and low power embedded applications. Support is provided for the current version of the Real-time Specification for Java (JSR-001), which allows applications to avoid non-deterministic garbage collection, and for brick wall time and space partitioning for high assurance applications.

**Ability to introduce new mission specific strategies and policies at run-time.** Typical of object-oriented languages, Java offers only limited support for the composition of behavior, and supports composition only at compile time. There is no support for composition involving *cross cutting* policies and strategies, or for the introduction of new policies and strategies at run time. In contrast, research in Aspect-Oriented Programming using

[2] Such as H4CR, a related virtual machine based proposal from Rockwell Collins that supports brick wall time and space partitioning for proveably secure systems.

[3] To ensure compatibility with the JVM standard, extended byte codes are introduced as post compilation optimizations of calls to conventional Java methods.

Java ([7] and http://www.aspectj.org) provides support for cross cutting strategies and the ability to extend existing behavior in ways not anticipated by its developers, e.g., to introduce probes to monitor remote system behavior, or assume control of remote systems, with proper authorization[4]. Current implementations of aspects, however, support composition only at compile time and make no attempt to address the needs of embedded, real-time applications or high assurance systems.

To remedy this, we propose to introduce a dynamic state based model of object-oriented execution targeted to real-time reactive behavior that extends the current class based static model. This state based execution model will provide an adaptable substrate, the "clay" that the other layers shape to achieve a high degree of run time adaptability. It will be supported by byte code extensions to the JVM, implemented in the JEM in microcode (Figure 1, Extended JVM in JEM microcode). Support will be provided for multiple dispatch as a means of representing orthogonal states. Dispatch tables will be per state, allowing run-time composition in terms of dispatch table transformations rather than directly modifications to the code. Aspects will be defined in terms of orthogonal states, and composed at run-time using state dispatch table transformations (Figure 1 Run time composer/weaver).

**Ability to introduce new mission specific scenarios.** Collections of execution scenarios (*use cases*) are commonly used to specify the behavioral requirements of systems. Scenarios are attractive because they present a view of the system's behavior from the user's perspective. While it is natural to also use scenarios to describe mission specific changes in the system's behavior, it is difficult to dynamically introduce new scenarios because they have no direct run-time representation.

The introduction of an explicit state based execution model, however, permits scenarios to be defined at run-time as a sequence of transitions. This permits us to add or remove mission specific scenarios at run-time by adding or removing associated states and transitions (Figure 1, Run time composer/weaver).

**Mission specific optimization of frequently executed scenarios.** Emperical evidence suggests that some 20% of all system scenarios are executed frequently, i.e., some 80% of the time. This appears to be true not only at a low level (system or subsystem level), but also at a higher level (system of systems or battlefield level). Emperical evidence also suggests that it is possible to achieve performance optimizations of 3X, 4X, or as much as 30X by taking advantage of detailed knowledge and special cases associated with individual scenarios[5]. The set of frequently executed scenarios (the frequently executed 20%), however, may vary from one type of mission to another, or from one stage of a mission to another, making scenario optimization a prime target for mission specific and stage specific adaptation. We expect to be able to introduce optimized versions of frequently executed scenarios using the same techniques used to introduce new mission scenarios (above).

**Support for Intelligent Agents.** Proposals for the semantic Web, for intelligent sensors, and for more capable autonomous vehicles rely heavily on intelligent mobile agent technology. Intelligent agents, however, are not typically real-time or hard real-time. Agent technology can also place a heavy burden upon scarce resources, especially in tightly constrained embedded systems.

This leads us to propose a solution involving extended byte code support for the rule selection and dispatching primitives needed by a rules engine (Figure 1 Intelligent agent rule engine). Such an approach is particularly elegant because the kind of deductive rules introduced by intelligent agent technology naturally complement the kind of reactive rules represented by transitions in our state based execution model. These two categories of rules are also recognized by the W3C RuleML standards initiative for the semantic World Wide Web (http://www.dfki.uni-kl.de/ruleml/).

**Application to systems with limited resources.** In order to rapidly introduce mission specific run-time changes over low bandwidth communication lines, we propose to transfer only unassembled "pieces" of behavior (e.g., just the new strategy or mission specific scenario) rather than completely assembled software classes. We also plan to conserve space in the target system by storing strategies, policies, use cases, etc. in an unassembled form until they are applied, avoiding the redundancy associated with storing multiple pre-assembled versions. Run-time composition is made fast by focusing only on dispatch table transformations, and by providing extended byte code support for the transformations themselves.

**The apparent conflict between high assurance and run-time adaptation.** Run-time adaptability offers new possibilities for addressing fault tolerance and other issues affecting safety. It, however, also raises its own issues. Using the proposed approach, run time composition should involve only transformations on state dispatch tables,

[4] The security policies of systems can themselves be represented as aspects, and changed dynamically.
[5] Based on work with system level "fast path optimizations" by Rockwell Collins for networked avionics applications communicating via CORBA.

rather than general modification of the code. This simplifies analysis and certification dramatically, and makes it much easier to verify the correctness of run-time composition. Extending prior research by Rockwell Collins on the use of Object-Oriented software in certified systems [6], it should be possible to provide a certification story acceptable to either the FAA or the military. The use of an explicit state based model for execution also invites the use of model checking and related techniques to analyze the system, e.g., at any time prior to broadcasting proposed changes.

## 4 Technical approach

Technically the approach involves the integration of a number of closely related concepts: formal state based type systems [5], multiple dispatch [2][4], formally specified aspects, behavioral subtyping [8][9], program slicing and partial evaluation [6], model checking, and mechanisms for the run time composition of behavior.

**Formal state based execution model.** The class and state machine models (static and dynamic models) of UML come from different backgrounds and are not well integrated at a fundamental level. For example, there is no widely accepted model of subtyping for state machines, as there is for types and classes [9]. Conversely traditional subtyping models suffer from problems related to co/contravariance as manifest in the "binary methods problem" [1]. These problems fundamentally limit flexibility and represent a barrier to formally correct composition. To resolve them, we propose to define a formal type model based on states and transitions rather than classes and operations, with a predicate based interpretation of orthogonal states [5] and a state based model of dispatch [3][5]. By using the overriding and dispatch rules for state machines and multiple dispatch, we are able to both resolve the binary methods problem and provide support for dynamically defined collaborative behavior. Overloading and overriding are viewed as related implementations of a single concept.

**Dynamic reclassification.** Dynamic reclassification refers to the ability to change the class of an object after it has been created without loss of object identity. Most programming languages do not support dynamic reclassification, even though may occur in the real world systems they model, e.g. when a person is promoted to a new role, when a programmable device is reprogrammed to behave as a different device, etc. The ability to dynamically reclassify objects is particularly important in highly adaptable, dynamic systems. We propose an implementation of dynamic reclassification based on matching pairs of state transitions. In between the execution of the two halves of the operation, space for the object may be reallocated. Alternately sufficient space may be allocated in advance to allow for the conversion. The choice is based on practical considerations related to speed, space and certifiability.

**Reclassification and client 'contracts'.** When an object or group of server objects is reclassified, existing clients may be affected. This situation may be viewed in terms of a guarantee of service. If the 'contract' with clients [10] contains a service guarantee, conversion to a class that fails to implement the client interface may not be allowed without a re-negotiation of the contract. Otherwise, existing clients may be notified of a loss of service by raising an associated exception.

**State based multiple dispatch and the representation of dynamic collaborative behavior.** Frequently the choice of action depends upon the states of multiple collaborating objects, including the type or state of the client making the call. The ability to directly express such collaborative behavior provides a run-time representation for patterns and aspects, which often involve interacting objects. By extending the concept of multiple dispatch to include dynamically defined states, rather than only statically defined types, we apply it in a much more useful context than is typical (e.g. in [1] and [2]), and provide both an elegant and alterable run-time representation of collaborative behavior.

**Rules of composition for dynamic collaborative behavior.** Subtyping involves the refinement of specifications through extension and overriding. This is appropriate when composing related behaviors in a manner that ensures substitutability. It, however, is often inappropriate when attempting to compose behaviors which are only loosely related. For example, behaviors that respond to a common set of events or, in the case of aspect oriented programming, a common set of meta-level events or "join points". For the composition of aspects, a state based execution model in which we can introduce orthogonal states, and in which we *maximize the execution* of compatible transitions, is preferable. In such a model, we consider both the precondition and the postcondition when we determine which transitions will fire. Barring an incompatibility with respect to the result (postcondition), all transitions eligible to execute [7] will do so.

---

[6] As part of an Aerospace Vehicle Systems Institute (AVSI) effort lead by Rockwell Collins, involving Boeing, Honeywell and Goodrich. These guidelines are expected to lead to FAA guidance of the use of Object-Oriented software for future FAA certified systems.

[7] All transitions whose precondition is satisfied.

**Formal aspects and safety critical properties.** Aspect-oriented programming currently lacks an underlying formal semantics, defines sets of join points (referred to as *point cuts*) only in terms of syntactic pattern matching, and fails to address problems related to the logical *interference* of aspects with one another. A more formal, logic based approach is necessary if we are to use aspects to enforce key properties of high assurance, safety critical systems. To infer that a policy establishes the desired system property requires that 1. the policy be shown to establish the property if logically and universally applied, 2. the policy be shown to have been applied at all join points to which it logically applies, 3. the weaving of the code be shown to be correct, and 4. no other aspect woven with the same code conflicts with this one. The state based execution model provides the basis to define a formal foundation for aspect oriented programming in which orthogonal states describe aspect specific behavior, and passage through a logically defined join point can generate a triggering event. Model checking and other formal methods will be used to address points 1 and 4. Because composition is based only on transformations of dispatch tables, we believe it can be both fast and certifiable in a manner that direct code modification could not.

**Multiple dispatch and partial evaluation.** Multiple dispatch and partial evaluation provide complementary mechanisms that support making an appropriate tradeoff between speed and space in embedded, hard real-time systems. Effectively multiple state based dispatch provides a means of 'slicing away' the 20% of the use cases executed most frequently, while partial evaluation provides a means of optimizing them in terms of the additional information (stronger preconditions) associated with the knowledge of specific scenarios. Partial evaluation can also be used to automate the conversion of existing class-method based code to state-transition based code [6].

## 5 References


[1] Kim Bruce, Luca Cardelli, Giuseppe Castagna, The Hopkins Objects Group, Gary T. Leavens, and Benjamin Pierce. "On Binary Methods", in *Theory and Practice of Object Systems*, John Wiley and Sons, Inc, 1995.

[2] G. Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science Series. Birkhauser, Boston. 1997.

[3] Craig Chambers. "Predicate Classes", ECOOP'93 Conference Proceedings, Kaiserslautern, Germany, July, 1993.

[4] Curtis Clifton, Gary T. Leavens, Craig Chambers, and Todd Millstein. "MultiJava: Modular Open Classes and Symmetric Multiple Dispatch for Java", *OOPSLA 2000 Conference Proceedings*, Minneapolis, Minn., volume 35, number 10 of *ACM SIGPLAN Notices*, pp. 130-145, ACM, New York, October, 2000.

[5] Gary Daugherty. "Unification of the Models for Types, Classes and State Machines", ECOOP'97 Workshop on Precise Semantics for Object-Modeling Techniques, TUM-INFO-5-1997-I9725-350/1.-FI, June 1997.

[6] John Hatcliff, Matthew B. Dwyer, Hongjun Zheng. "Slicing Software for Model Construction", *Journal of Higher-order and Symbolic Computation*. 13(4), A special issue containing selected papers from the1999 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation.

[7] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. "Aspect-Oriented Programming", in proceedings of the *Eleventh European Conference on Object-Oriented Programming*, Finland. Springer-Verlag LNCS 1241, June 1997.

[8] Gary T. Leavens and Krishna Kishore Dhara. Concepts of Behavioral Subtyping and a Sketch of their Extension to Component-Based Systems. In Gary T. Leavens and Murali Sitaraman (eds.), Foundations of Component-Based Systems, chapter 6, pages 113-135. Cambridge University Press, 2000.

[9] Barbara Liskov and Jeanette Wing. "A Behavioral Notion of Subtyping", *ACM Transactions on Programming Languages and Systems*, 16(6): 1811-1841, November 1994.

[10] Bertrand Meyer. "Applying design by contract." IEEE Computer 25(10):40-51, October 1992.

[11] K. L. McMillan, Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academic, 1993.


## 6 About the author

Gary Daugherty is a senior member of the technical staff of the Rockwell Collins Advanced Technology Center in Cedar Rapids, Iowa. His background is in object-oriented modeling and languages, formal specification languages, real-time reactive state based communication systems, operating systems, aspect oriented programming, safety critical FAA certified software systems, and artificial intelligence/expert systems. He has worked with Drs. Gary Leavens and Suraj Kothari of Iowa State University, Dr. Dave Hardin of aJile Systems, and Drs. John Hatcliff of Kansas State University and Matthew Dwyer of Kansas State University on related proposals and projects.