Evolution of the Milieu Approach for Software Development for the Polymorphous Computing Architecture Program

Yoginder S. Dandass
Mississippi State University

Anthony Skjellum
MPI Software Technology, Inc.

Theodore Bapty
Vanderbilt University

Charles Summey
MPI Software Technology, Inc.

Ben Abbott
Southwest Research Institute

Hong Yuan
MPI Software Technology, Inc.

A key goal of the DARPA Polymorphous Computing Architectures (PCA) program is to develop reactive closed-loop systems that are capable of being dynamically reconfigured in order to respond to changing mission scenarios. This is in contrast to current systems that are fixed in nature and rely on architecture and software optimizations targeted for specific missions. In order to accomplish this objective, a number of "malleable" processing elements, runtime support software, compilers, and application development tools are being developed by a variety of research teams. Application software development for PCA systems is expected to be particularly challenging because of the need to respond to rapid changes in the mission requirements and environment. The authors of this talk have formed a team that is focusing its effort on developing application modeling tools and middleware libraries that assist in managing the complexity of developing, deploying, and maintaining PCA applications.

The four high-level research objectives of this effort are as follows:

- Study, prototype and develop a modeling language for streaming and threaded resources and components,
- Perform Design Space exploration to enable PCA scheduling (including use of AI techniques),
- Achieve System Synthesis and Generation (including performance monitoring and feedback) to support, and
- Accomplish Dynamic Reconfiguration study and support for PCA.

Design space exploration/navigation and optimization techniques are used to map the application and specifications to component and resource implementations. Generation includes configuration of runtime dynamic resource managers and local system optimization to ensure correct and efficient morphs in dynamic, non-pre-verified configurations. Online composition of these verified/optimized designs implements the new target morph configurations.

Because it is impossible to pre-generate all potential configurations of a PCA system, online configuration is necessary. Since the typical design space for an application is large, online navigation of the entire design space proves infeasible. Instead, the design space is pre-digest down to restricted subspaces. These subspaces offer the flexibility of optimization among architecture, resource, and application variations with the ability to find solutions rapidly, in a guaranteed time period.

During system generation, allocated and unused resources along with mission and application requirement constraints are collected in the form of *metadata* that are compiled into the optimized system schedule thereby providing an annotated interface for various (perhaps competing) applications requesting resource configuration changes during runtime.

Metadata compiled by the optimization and scheduling tools (as depicted in Figure 1) drives the morphing performed by a PCA system. This compiled metadata is assembled from metadata inputs from the resource modeling, application and component modeling, and programming-style modeling tools. The compiled metadata also partitions the optimization search space, thereby reducing the time required by the runtime manager to dynamically optimize resource allocations.

In many cases, the compiled metadata will explicitly enumerate a number of optimal resource allocation schedules generated offline in order to expedite mode changes. The software architecture under investigation also provides for run-time system performance monitoring by the resource manager and allows the runtime system to perform limited online optimization in response to dynamic application requests and feedback from the hardware-based performance monitoring services.

Modeling allows explicit representation of alternative components, allowing functionally equivalent components to be included in the design when models are constructed. Alternatives are resolved through design-space exploration.

Component models contain information about how a component may morph, including the types of morph requests a component will issue, bounds on those requests (min/max number of processors a component will request), types of morphs that can be requested of the component (morph constraints), and so on.

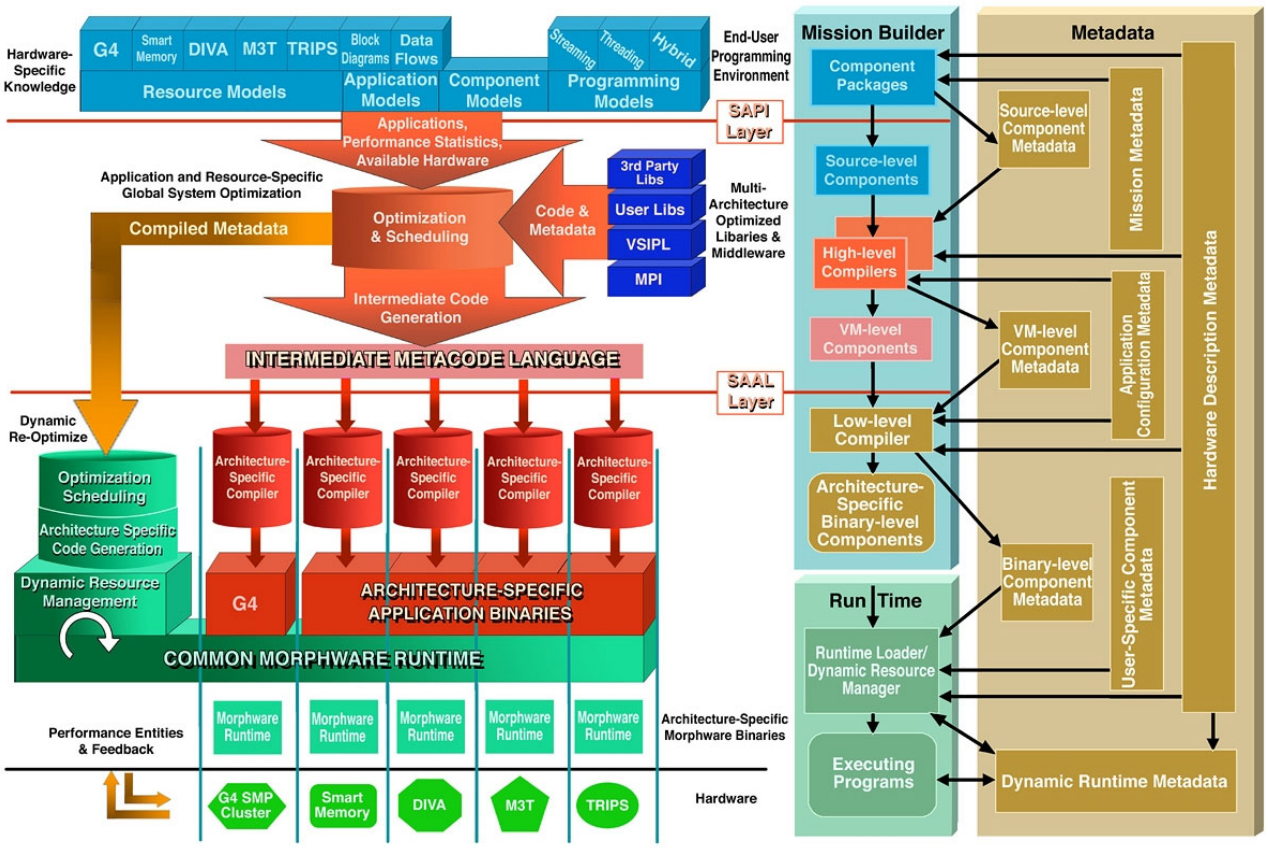

Figure 1: Overview of the Milieu Application Construction Environment

Once design space navigation has been performed, a scheduling optimization pass performs a fine-grained analysis on the components mapped to each node to determine the proper schedule for each node. Combinatorial optimization techniques developed in the areas of operations research and artificial intelligence are under investigation in order to systematically and efficiently search the solution space for optimal schedules. Scheduling algorithms based on variants of branch-and-bound and other AI-based search techniques to construct schedules using heuristics are also being investigated. Simulated annealing and genetic/evolutionary programming techniques are utilized for evolving near-optimal schedules for those problems that cannot be solved using heuristics alone. In many dynamic real-time systems it is impossible to anticipate and plan for all possible combinations of workloads a-priori. In such systems, neural network and reinforcement learning technologies can be applied to construct scheduling algorithms that learn effective scheduling strategies during system runtime.

The compiled application constraints support and coordinate individual application performance perturbation in a centralized, priority-based, fashion. As such, violation of required guaranteed real-time deadlines are avoided through the constraint verification process. For example, if a non-real-time application requests sharing of resources that would cause a currently active real-time application on the same PCA hardware to compromise its timing constraints, the non-real-time application may be denied its resource request. This implies that the non-real-time application was given an importance lower than that of the real-time application. The actual importance of the various applications will be based on an application suggested and common morphware runtime constrained priority number. Thus, even if a task is non-real-time, but is very important, a system operator may guide it to become "most important" (even to the cost of missing deadlines in other applications).

In addition to maintaining hard resource constraints, the dynamic resource manager also maintains and adjusts soft constraints by collating performance metrics feed back from currently running applications. These current and

historic performance metrics along with the compiled constraints supplied directly through the model guide online system optimization during runtime. Thus, an application that did not directly request an exact architecture could be bounced through a variety of shapes as the overall PCA system attempts to "tune" itself.

This talk will provide an overview of the approach use by the authors of the talk to manage the problem of PCA application development through the use of application modeling tools and middleware libraries. The talk also introduces the critical elements of the required tool chain that will lead the successful deployment of reconfigurable software components.