AFRL-IF-RS-TR-2004-267
Final Technical Report
October 2004

A MULTI-TIME SCALE MORPHABLE SOFTWARE MILIEU FOR POLYMORPHOUS COMPUTING ARCHITECTURES (PCA) - COMPOSABLE, SCALABLE SYSTEMS

MPI Software Technology, Incorporated

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. L168

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-267 has been reviewed and is approved for publication.

APPROVED: /s/

CHRISTOPHER J. FLYNN
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCTOBER 2004	Final Jan 02 – Nov 03

4. TITLE AND SUBTITLE
A MULTI-TIME SCALE MORPHABLE SOFTWARE MILIEU FOR POLYMORPHOUS COMPUTING ARCHITECTURES (PCA) - COMPOSABLE, SCALABLE SYSTEMS

5. FUNDING NUMBERS
C - F30602-01-C-0079
PE - 62712E
PR - L168
TA - MS
WU - MS

6. AUTHOR(S)
Anthony Skjellum,
Hong Yuan, and
Yoginder S. Dandass

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MPI Software Technology, Incorporated
110 12th Street North Suite D103
Birmingham Alabama 35203-1537

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTC
26 Electronic Parkway
Rome New York 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-267

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Christopher J. Flynn/IFTC/(315) 330-3249/ Christopher.Flynn@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
Polymorphous Computing Architectures (PCA) rapidly "morph" (reorganize) software and hardware configurations in order to achieve high performance on computation styles ranging from specialized streaming to general threaded applications; the former is most like a pipeline processor, while the latter is most like a collection of common desktop processors. This project explored a multi-time scale morphable software environment (denoted 'milieu') whose purpose was to suggest ways to build applications from components of varying scales; these components were to include small and large amounts of code, different algorithms, and different parallelism. The purpose of PCA is to reduce the cost, size, power, and other "footprint" aspects of computing that can be most efficiently done with radically different kinds of computing elements; pre-PCA systems connect disparate parts with low utilization outcomes. This project sought to create a model-based software framework for reactive monitoring, optimization, modeling, resource negotiation and allocation, regeneration, and verification. Model-based software uses principles like "legos" or "tinker toys" to model interactions and roles, and allow a framework in which software is created. This project report addresses the research objectives and key accomplishments, while introducing the technical concepts and approaches developed in this project for the PCA software development. Future work is also recommended in this report.

14. SUBJECT TERMS
Polymorphous Computing, Polymorphous Software

15. NUMBER OF PAGES
31

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Acknowledgement

The authors would like to thank Mr. Robert Graybill of DARPA/IPTO, the PCA program manager, and Mr. Christopher Flynn of AFRL/IFTC, the government's contracting agent, for their support of this effort. The authors also thank Dr. Ted Bapty and Mr. Brandon Eames at Vanderbilt University for their valuable suggestions and help. The authors also acknowledge all the help and support of Mr. Jon Hiller of STA, who helped support program logistics, and create opportunities for enhanced collaborations throughout the effort. We also appreciate the excellent technical interchanges during the Morphware Forum meetings.

The authors of this report would also like to thank the following colleagues/project participants:

Charles Summey
David Dampier
Dave Leimbach
Diane Wooley
Murali Beddhu
Torey Alford

Summary

As a part of the Polymorphous Computing Architectures (PCA) program sponsored by the Defense Advanced Research Projects Agency (DARPA) [24], the project "A Multi-Time Scale Morphable Software Milieu for PCA-Composable, Scalable Systems" was funded to address the research of PCA software system design and development. To support the use of these polymorphous computing systems, this program created a model-based software framework for reactive monitoring, optimization, modeling, resource negotiation and allocation, regeneration, and verification[1].

In order to meet the requirement that PCA application programs be created for the changing mission scenarios at runtime, this research proposed the idea that combinatorial optimization drives choice of components for missions. In this architecture, a set of alternatives for "how" to do a certain operation are defined by component builders (programmers) and placed in component libraries (warehouses or repositories). These components publish their capabilities and costs, and mathematical optimization is used to choose the best choices, based on the mission planner's set of objectives (high speed, low power, low jitter, etc).

An early prototype of multi-level polymorphous environment was developed. The prototype integrated Model-Driven Architecture (MDA), an advanced software engineering approach, into the design and development of the PCA application system. The research work and contributions included definition of componentization, morph types specification, and threaded VM decisions. The vision of this project is that polymorphous software environment is independent of PCA hardware. A fundamental view held through this research is that compilers do not perform global resource management. Most of the next phase of the PCA work, which follows the current phase for which we were funded, emphasizes using compilers to handle certain optimization tasks, and the reader may wish to consult future results from these programs to determine exactly which kinds of optimizations are handled by compilers vs. component libraries in emerging PCA solutions of the coming years.

Additionally, as a part of the team effort of the PCA research community, this project research contributed to standardization efforts of Morphware Forum to ensure software component approach across PCA morphable hardware. The research developed models and taxonomies to describe PCA hardware and software to reveal opportunities for performance, portability, evolvability, and optionality, elucidated requirements and capabilities of a full programming system for PCA, and developed component metadata definition and concepts. The concepts (performance, portability, evolvability, and optionality) address specific qualities of software that can be appreciated by different stakeholders in a mission or system program. These features cross-cut between runtime behaviors for a specific mission (with hardware changing availability over a runtime), and the long-term "productivity" of software housed in component libraries,

[1] Many of the terms described here and in the rest of the report are described in the Glossary (see pages 2-3) to make this presentation more accessible to people generally knowledgeable about computer science or computer hardware, but not necessarily about advanced architectures, high performance computing, or software engineering concepts.

targeted for PCA hardware (that may change over time). The research and experiments conducted under this project identified the requirements for ways that high-level and low-level compilers help to provide the SAAL and SAPI abstractions (see the Glossary). In other words, a model for how software and hardware would be assembled was provided from the outset, including the SAAL and SAPI abstractions, and this project provided an elaboration of how to build and use software on PCA hardware, consistent with the initial vision given by DARPA and subsequently ratified by the Morphware Forum.

Highlights of this project report include the implementation of a model-driven PCA mission build chain, method for organizing complex systems as components, mapping metadata with components and system build tools, and automatic and user-driven selection of the "best" design alternatives for components given mission goals. Major concepts introduced in this report include interaction of the compilers and compile-time resource management tools at build time, interaction of the runtime system, dynamic resource manger, and PCA software at runtime, optimization driving choice of components for mission, and the use of metadata to control component optionality. This model has not been fully adopted by the Morphware Forum, but represents a revolutionary approach to build applications with complex, heterogeneous (CPU and FPGA) and/or PCA hardware. It is evident that the framework will be of interest in future versions of complex software systems with evolving mission requirements over different time scales.

The key achievements summarized in this report include morphing diagram, integrated build architecture, componentization concepts, and metadata system design. The research achievements have had some significant impacts to the PCA research community. As reflected in the research materials published on Morphware Forum [21], these impacts include inputs to high-performance component design and specification, metadata system, component model unification, **c**omponentization, morph types specification, and threaded VM decisions.

Introduction

The Polymorphous Computing Architectures (PCA) program [24] is a Defense Advanced Research Projects Agency (DARPA) effort "to develop a revolutionary approach to implementing embedded computing systems that support reactive, multi-mission, multi-sensor, and in-flight retargetable missions, and that reduce the time needed for payload adaptation, optimization, and validation from years to days to minutes" [23]. Unlike the traditional software development approach of "hardware first and software last," the PCA program moves beyond conventional computer hardware and software to flexible, "polymorphous" computing systems. "A polymorphous computing system (chips, processing architecture, memory, networks, and software) will 'morph' (take on or pass through varying forms or implementations) to best fit changing mission requirements, sensor configurations, and operational constraints during a mission, for changing operational scenarios, or over the lifetime of a deployed platform" [23] .

In many high performance applications, processing data from a sensor or a network of sensors network is required; radar, sonar, and multimedia applications all follow this paradigm. Fast data traversal into and optionally out of the computer system is often required. The need for high performance, flexible implementations of this type of applications is one of the major reasons that motivated DARPA to develop PCA technology [23].

An example to show the type and scale of a particular sample streaming sensor application is the Integrated Radar-Tracker (IRT) benchmark application created by MIT Lincoln Laboratory [14]. The IRT is a MATLAB-based "end-to-end specification of a modern intelligence, surveillance, and reconnaissance (ISR) radar system" [23]. Motivated by a space-based radar application, the IRT system "embodies all of the major attributes required in a defense-oriented PCA application test: both streaming and data-dependent threaded computation with multiple sub-types of each (*e.g.*, fast transforms *vs.* vector-matrix arithmetic in the streaming elements); heavy computational loads; and multiple application-level parallelization and morphing opportunities" [23]. As a representative PCA application, the IRT benchmark is used in this project research for implementing the proposed model-driven PCA mission build chain, which is introduced in detail in a later section. The final product of this research (called the story board), actually componentized the IRT benchmark, provided both a high-level and a low-level compilation, and demonstrated it running on an x86 laptop. This working story board illustrated the viability of the build chain on actual working, but pre-production components based literally on the IRT benchmark, and including many of the advanced concepts proposed by this project.

This project, "A Multi-Time Scale Morphable Software Milieu for PCA-Composable, Scalable Systems" described in this report concentrated on the research of PCA software design and development. To support the use of these polymorphous computing systems, this program created a model-based software framework for reactive monitoring, optimization, modeling, resource negotiation and allocation, regeneration, and verification. An experiment to implement a model-driven PCA mission build chain for the IRT benchmark application was conducted in this project in order to demonstrate the concepts and approaches proposed for the PCA software

development and engineering. The key concepts developed and the experiments conducted for this research project are introduced in the following sections.

The rest of this report is organized as follows. The section “Methods, Assumptions and Procedures,” describes the methodology and experiments that were developed for a number of key aspects of the research; the section “Results and Discussion” addresses major technical achievements and most recent research results; The section “Conclusions and Recommendations” summarizes the project accomplishments and provides some recommendations for future research direction.

Methods, Assumption, and Procedures

This section introduces the methodology and experiments, developed and conducted within the research of this project. With a focus on the key technical contributions, this section examines a set of critical concepts and approaches explored in the research.

Metadata Concept and Integrated PCA Architecture

Figure 1 illustrates the systematic strategy designed and developed in this project for building a morphable software program for a target PCA mission application. Metadata drives software morphing on different levels of the software and hardware architecture. Metadata is closely associated with source code of application components, and these two aspects together form a "metacode" structure. Metadata can be divided into off-line metadata and online metadata. Off-line metadata is gained through long-term performance evaluation in experiments, while online metadata is obtained from runtime resource management tool. Based on the change of online runtime metadata, the system intelligently searches the design space of components and design alternatives. By checking the off-line metadata for each candidate, the system finally selects the most suitable candidate for composing optimal or near-optimal solutions.

Predictive Model and Optimization

One of the major efforts of the research was the investigation of a predictive model that can be integrated into the PCA build chain for future parallel application (extensions to what was achieved in this project). A predictive software model was designed to dynamically select parametric design alternatives for morphing application programs. This software model is supposed to input mission-specific metadata, meet resource constraints, and generate optimal composition of the application program. By appropriately handling data distribution, the program generated through this system executes on parallel resources and expects to achieve QoS in terms of memory and time in the presence of resource constraints. In predictive models, the size, cost and space of a component's runtime features can be known from a function that is stored along with the program component, and the evaluation of this function allows for decision making by other software to enhance the user program's final outcome.

The benchmark application for this experiment is a poly-algorithm application for parallel dense matrix multiplication on two-dimensional process grid topologies [18]. The basic assumption is that no single algorithm always achieves the best performance on different matrix and grid shapes. Base on this assumption, the objective of the experiment was to minimize time to solution while avoiding extensive data remapping. The key issue to be resolved is to determine under what situations an algorithm achieves the best performance. Various data distribution techniques including linear data distribution and block scattered data distributions were investigated. A prototype optimizer was designed using a dynamic programming approach for searching optimal program composition.

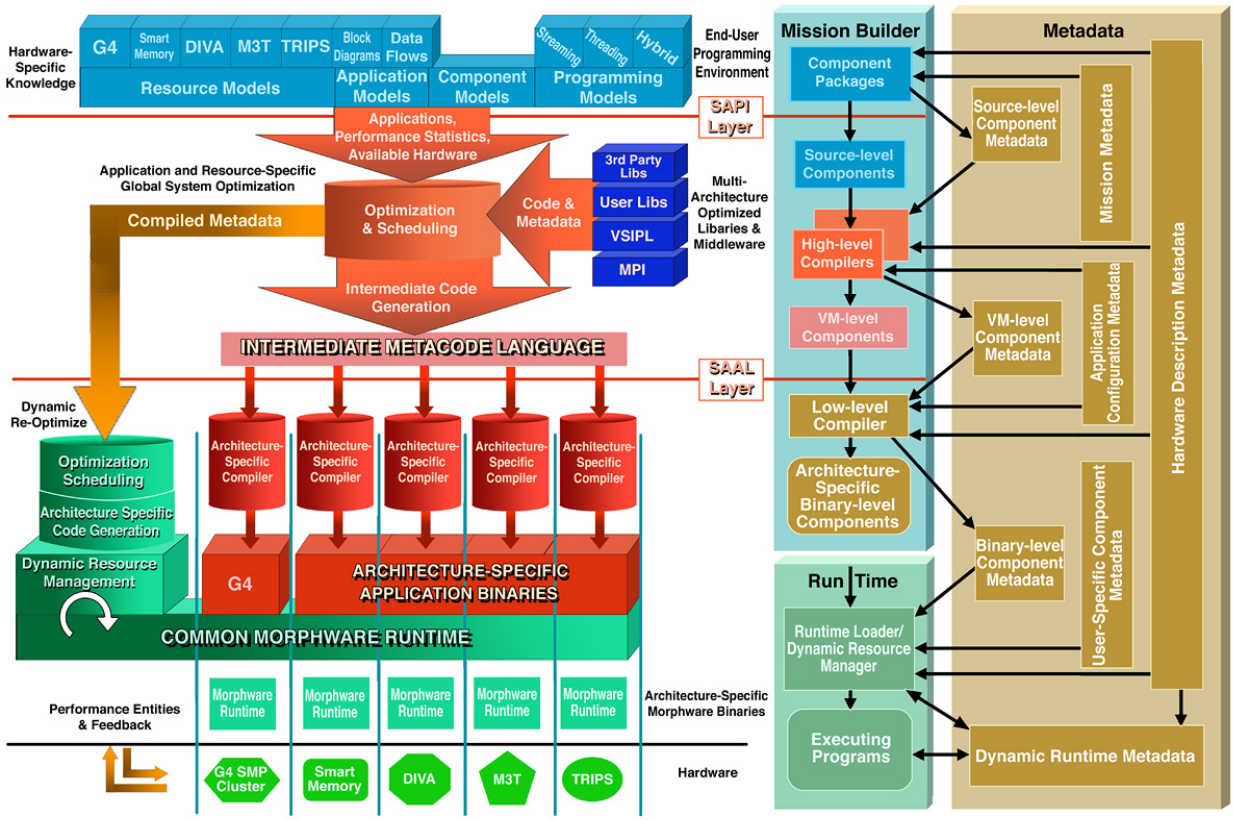

Figure 1. Integrated PCA Architecture

"Final Morphware Diagram"

There are many possible scenarios and situations in which a change in the configuration of a PCA system may be desired. Therefore, identifying and categorizing the types of these situations to aid in identifying the hardware and software services required by applications, operating systems, and run-time resource managers have become a key concern of the researchers in the PCA research community. The research of this project provides a major contribution to this collaborative effort.

As eventually published online by the Morphware Forum [23], this categorization encodes three orthogonal aspects of the attributes of a morph. These are as follows:

- whether the morph is initiated directly by an API[2] call within the application code, or is it initiated by the run-time system or compiler invisibly to the application programmer;
- whether the physical resources allocated to the application must change or stay the same; and
- whether the components of the application (or the entire application) continue to execute or are reloaded or replaced.

Figure 2, as published by Morphware Forum [23], summarizes the set of morph types resulting from these attributes.

	Run-time System		Application Programmer		Compiling System	
	Components continue	Components change	Components continue	Components change	Components continue	Components change
Resource allocation doesn't change	*Type 0a*	*Type 1a*	*Type 2a*	*Type 3a*	*Type 4a*	*Type 5a*
	Run-time environment changes transparently to the running application.	Run-time system changes components to reconfigured but equivalent set of resources.	Application makes API call to make suggestions.	Application makes API call to change processing mode but does so within existing resource set.	Compiler instructions reconfigure allocated resources.	Compiler switches to a different library able to use the same resources.
Resource allocation changes	*Type 0b*	*Type 1b*	*Type 2b*	*Type 3b*	*Type 4b*	*Type 5b*
	Run-time system changes resource allocation of a running application transparently to the application.	Run-time system configures resources and loads components at application startup.	Application makes API call to give up or gain some resources.	Application makes API call to add or replace one or more components using different resources.	Compiler requests different resources to meet change in performance specified by metadata.	Compiler switches to a different library that uses different resources.

Figure 2. Final Morphware Diagram

[2] API stands for "Application Programmer Interface" and is a means by which the SAAL and SAPI layers may provide access to certain functionality to a component (and therefore indirectly to a user). As opposed to a component model that specifies capabilities and not literal code, an API model is needed for certain fixed capabilities, like morphing, in the PCA framework.

Morphware Compilation Approach

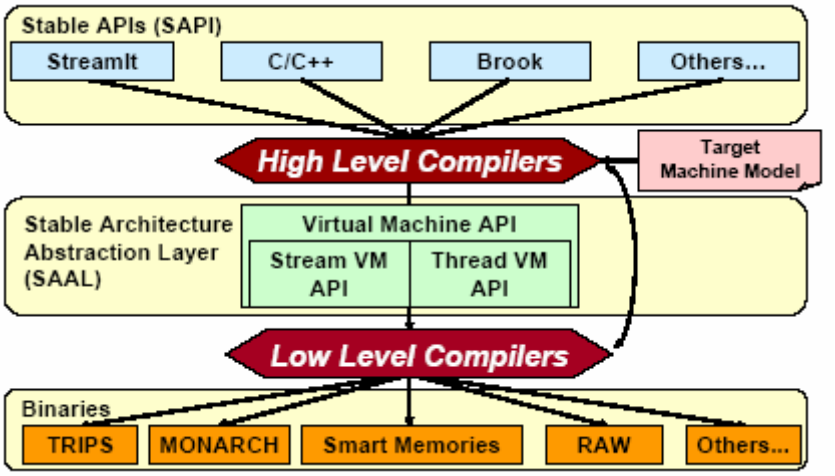

Figure 3. Morphware Compilation

In September 2003, the Morphware Forum [21] released the morphware compilation diagram (Figure 3) as a fundamental PCA system architecture. In this architecture, applications are written in one or more of the source languages. The target platform and the set of possible configurations of the target platform are described using the PCA machine model metadata context. The researchers of the Morphware Forum (including the workers on this project) have also worked to define a metadata context for application information including a number of elements such as performance requirements (*e.g.*, computational throughput, latency and power), input rates, and module interconnections.

According to this architecture, "the high-level compiler is responsible for selecting the desired platform configuration, and for converting the SAPI source code to SAAL code. The high-level compiler performs coarse-grained parallelization of the input application, based on the granularity appropriate to the target platform. In order to accomplish this effectively, the high-level compiler must have a description of the target platform. This description is provided in the PCA machine model metadata context, where the capabilities and resources of PCA platforms are described in terms of a common, high-level model that allows the high-level compiler to perform accurate performance estimates. The output from a high-level compiler is SAAL code and a desired initial platform configuration. Depending on the application source language, the SAAL intermediate code may be Streaming Virtual Machine (SVM) code, User Virtual Machine (UVM) code, or Threaded Virtual Machine-Hardware Abstraction Layer (TVM-HAL) code" [23].

The low-level compiler is used to build an architecture-specific executable. “The low-level compiler accepts the SAAL code and desired platform configuration metadata output by the high-level compiler and produces native machine code suitable for execution on the target PCA platform. If a performance constraint, resource requirement, or desired configuration is unobtainable by a low-level compiler, that low-level compiler fails and reports the error” [23].

Conforming to the architecture diagram released by the Morphware Forum, the research of this project contributed in many aspects for detailed implementation of the mission building process that specifically demonstrated the feasibility of the morphware compilation architecture. A model-driven build chain was implemented to prove the concepts described above in detail. In the model-driven mission build chain, a method for organizing complex software systems as components has been examined. The issues investigated through the experiment include metadata specification, mapping metadata with components, user-driven selection of “best” alternatives for components given mission goals, and the automatic application program generation through high-level and low-level compilations. Additionally, a set of systematic ideas was proposed to show the interaction between the compilers and compile-time resource management tools at build time, as well as the interaction among the runtime system, dynamic resource manager, and PCA software at runtime. A compile-time resource management tool is proposed in order to select appropriate component alternatives. Meanwhile, the experiment of the project examined parametric component with metadata characteristics such as runtime performance and resource requirements.

Componentization Concepts

PCA application software is viewed as a component-based system. A major concern of the research of this project is identification of the PCA software component and componentization of PCA application software. As defined in the glossary, a component holds software that defines certain functionality, and publishes requirements while offering features (*e.g.*, “10 watts on architecture X for a 1,024 point single precision FFT.”) Component-based software can be viewed as a composition of a set of the components. Specifically in the PCA mission build chain introduced in this report, for demonstration purposes, a component is narrowly defined as a reusable software functional unit, particularly a function block in a sequence of program operations, which is usually associated with a source code file that fully qualifies the capabilities.

The research and experiments of this project shows that the software components are hierarchical. Given the metadata context information, certain components of an application may be dynamically replaced at runtime in order to meet the changing mission requirement and resources constraints. Moreover, design and implementation alternatives are explicitly represented in the hierarchy at any level. PCA Application is a component composed from a collection of components, component grouping specifications, abstract Virtual Machine specifications, and component-Virtual Machine element mappings. PCA mission is composed of PCA applications, Virtual Machine instantiation, component to physical Virtual Machine element mappings, mode-change rules, and mission constraints that govern application behaviors.

PCA Build Chain: A Model-Driven Problem Solving Environment

In order to demonstrate the concepts for the PCA system architecture, metadata, and componentization, a PCA mission build chain demonstration was designed and implemented on MIT Integrated Radar Tracker (IRT) benchmark application [14]. As a model-driven architecture, this mission build chain provides a design approach using modeling technology, then derive application programs from the graphical models. Specifically, the mission build chain includes the following steps:

- Application system and metadata modeling.
- XML model parsing.
- Multi-level compilation for intermediate code generation and architecture-specific binary generation.
- Runtime execution.
- Optimization with design space searching.

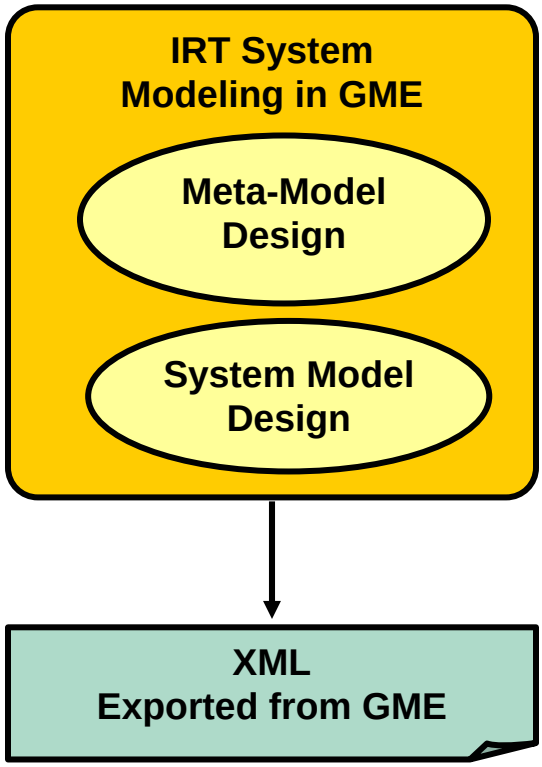

Figure 4. Modeling in PCA Build Chain

PCA Build Chain: System Modeling

The PCA modeling uses GME graphical modeling interface. Generic Modeling Environment (GME) is a modeling toolset developed at Vanderbilt University [15]. GME is a configurable toolset that supports easy creation of domain-specific modeling. The primarily graphical, domain-specific models can represent an application and its environment, including software modules, hardware resources, and their relationship. GME provides a graphical user interface for visually designing PCA applications through a two-step process including metamodeling and domain-specific application modeling. GME can also assist software developers to synthesize application programs from the graphical models.

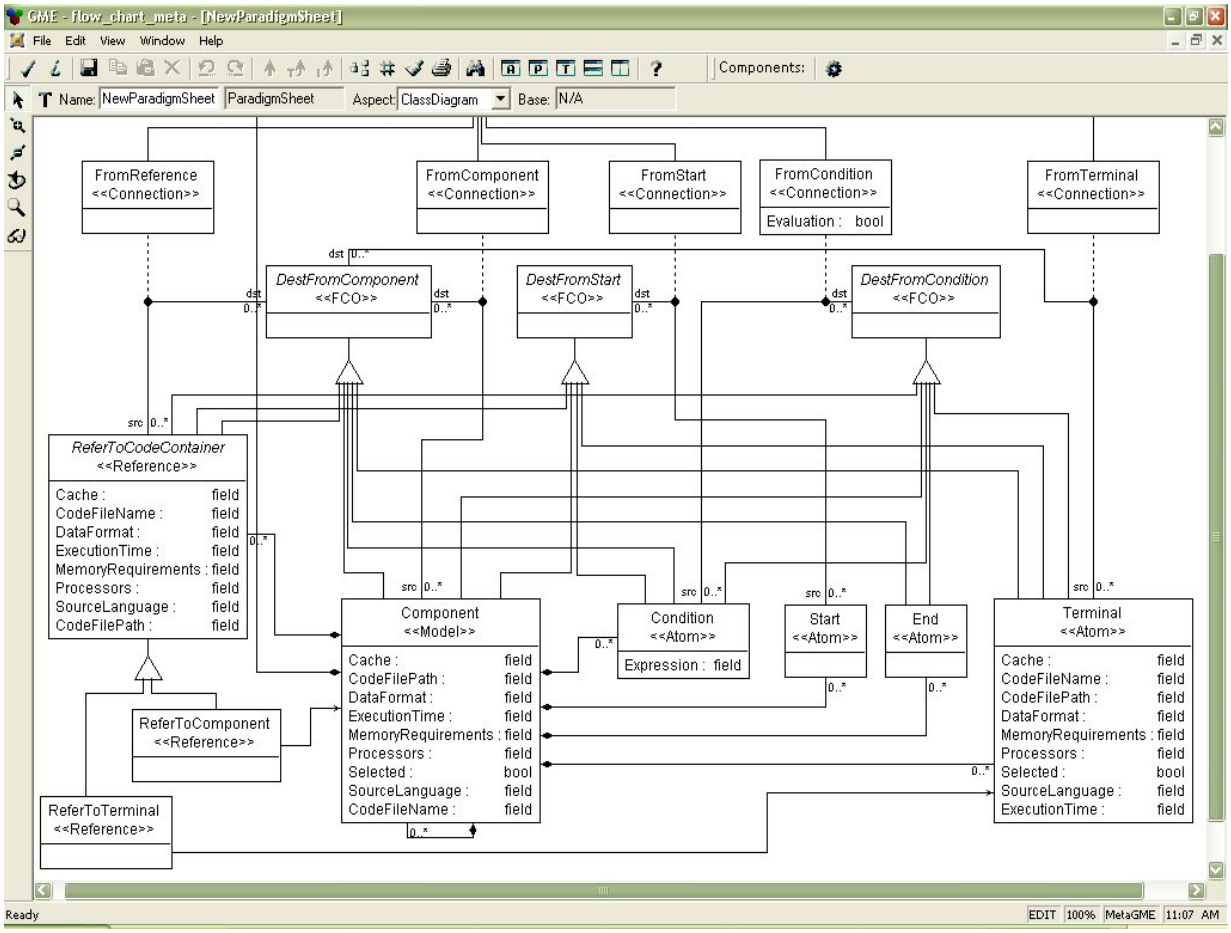

Figure 5. Metamodel in PCA Build Chain

The modeling process requires two steps, metamodeling and domain-specific application system modeling. Metamodel, as shown in Figure 5, is an UML-based model that defines objects that can be instantiated and used in the application model [28]. Based on the definitions of

metamodel, developers can model a specific application in the application model shown in Figure 6, which is domain-specific for the IRT benchmark application. In the application model, the system component hierarchy and design alternatives are identified. Model component interaction and dependency are defined. Metadata values are assigned and associated with components and alternatives. Finally a XML representation of the model is exported [12].

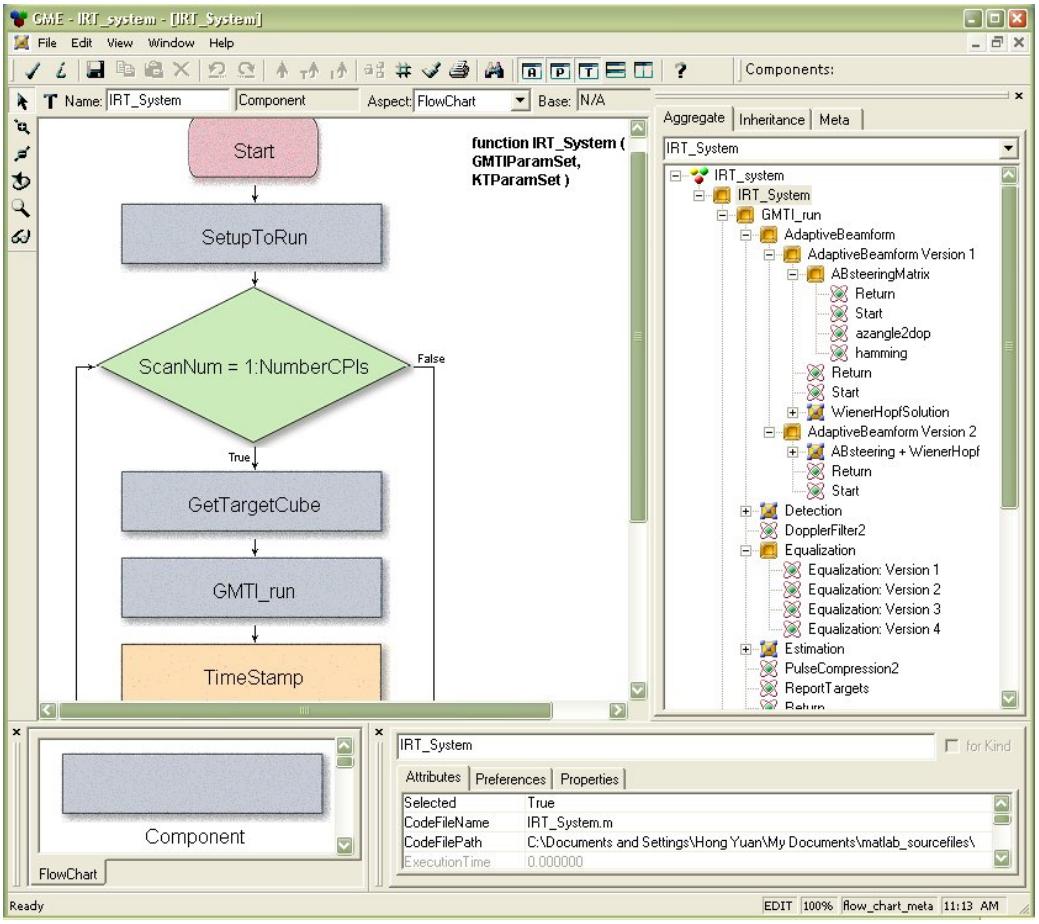

Figure 6. Application Model in PCA Build Chain

PCA Build Chain: Program Generation

Once a system model is created, a model processor parses the XML formatted model, and traverses through the model component hierarchy and retrieve metadata specification of each component while selecting optimal design alternatives. Finally, the model processor dynamically generates makefiles for intermediate code generation and compilation.

The model processor parses XML model representation by utilizing the Universal Data Model (UDM) framework [32], developed at Vanderbilt University. UDM provides a set of supporting tools that are used to generate C++ programmatic interfaces from UML class diagrams of data structures. UDM command-line utility processes the XML representation of the metamodel and automatically generates a set of C++ interface files, including a C++ header file, a C++ implementation file, and a XML document type definition file. These files allow a C++ model processor program to easily access all components design in the graphical model.

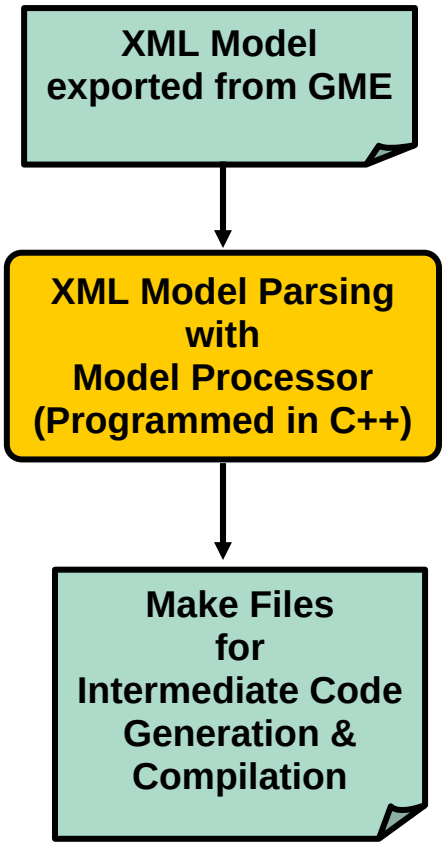

Figure 7. Model Processing in PCA Build Chain

PCA Build Chain: Compilation

As introduced in previous sections, the PCA application compilation process is a multilevel compilation that uses “High-level Compiler” and “Low-level Compiler” (see Figure 3). The “High-level Compiler” first collects the source code implementations of all the selected components designed in the model from a common source code repository. Then the compiler compiles all the selected pieces of the source code and generates the intermediate code, which is the Virtual Machines. This compilation step translates the application program from application source language to the intermediate language. The “Low-level Compiler” compiles the intermediate code and generates architecture-specific executables. Metadata is expected to be interpreted and generated for the resource manager at this level.

PCA Build Chain: Runtime Execution

Eventually, the binary executable program generated from the above steps is executed on the specified architecture. The metadata generated in the previous step is used by the resource manager and the application.

“Metacode” Evolution

The PCA mission build chain is a demonstration of the “metacode” concepts developed in the research. “Metacode” concept proposed in this research tightly associates the metadata information with PCA software components, so that these two key elements are seamlessly integrated with each other as a single PCA “metacode”. The combination of the metadata and components facilitates the nature of PCA system that the reconfiguration (“morphing”) is dependent upon the changing metadata information.

According to the long-term vision, the metacode system will evolve following the steps illustrated in Figure 8. The metacode system development will gradually evolve from a set of dispersed modules into a seamlessly integrated system.

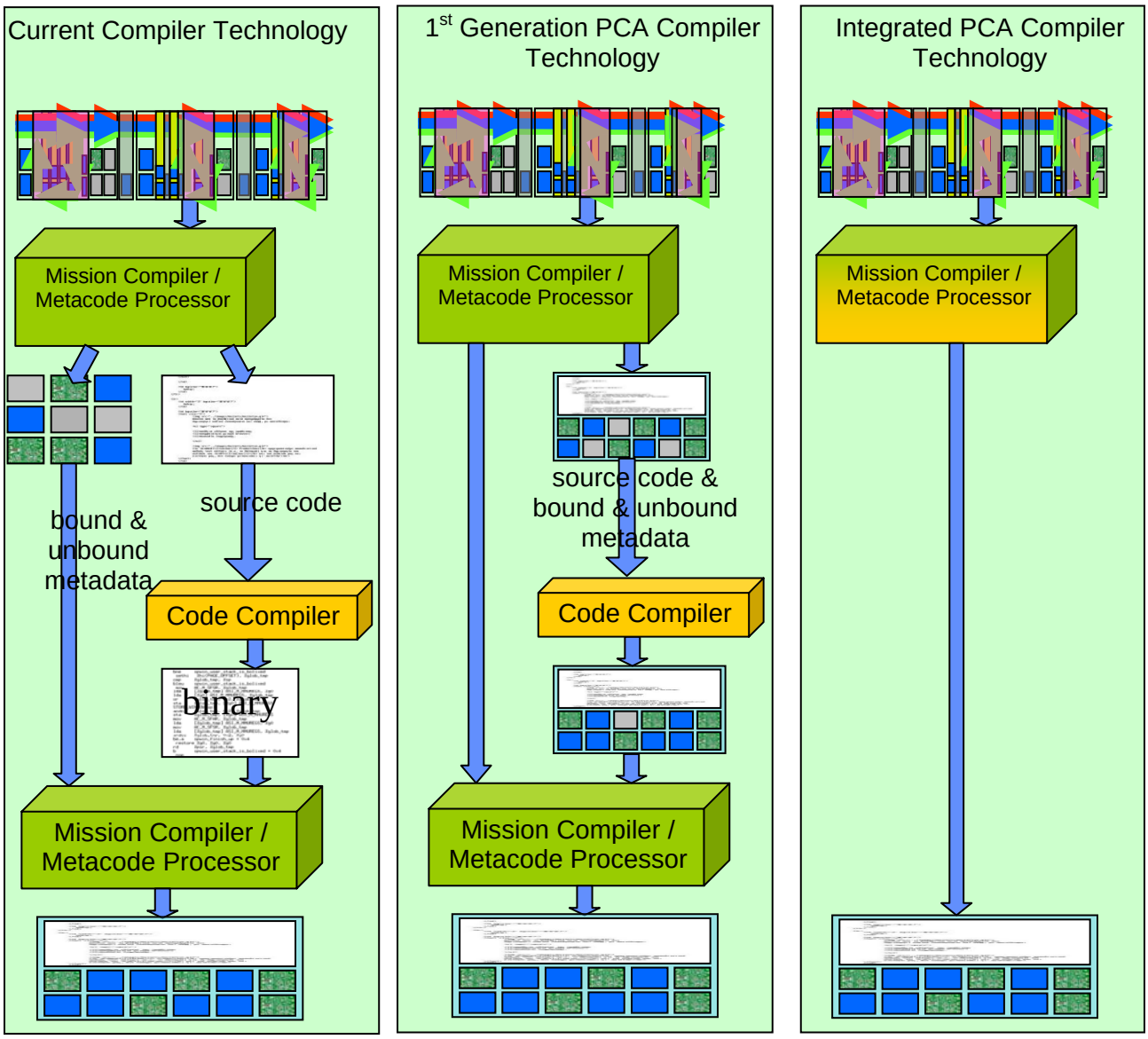

Figure 8. Evolution of the "Metacode" Concepts

Other Academic Activities

During the period of the project, a number of formal and informal academic activities were conducted. These activities are not only for the research progress, but also for supporting other research teams in the PCA community. These academic activities include close collaboration with Georgia Institute of Technology and the Space and Naval Warfare Systems Center San Diego through meetings, weekly conference calls, and discussions. The activities also include the efforts to get the entire forum to consider morphs as a major issue, showing that missions are a key level of concern and a single monolithic compilation is not the entire story of PCA software, as well as close collaboration with Vanderbilt University on the concept of morphable software milieu. Through these academic activities, many initial ideas were gradually developed and improved. These activities directly resulted in several significant contributions to the Morphware Forum in clarifying componentization concepts, developing metadata specification, and improving low-level threaded VM specification.

Results and Discussion

With an emphasis on the PCA software development, the research and experiments of this project produced a number of key accomplishments. A storyboard of PCA mission building process was developed. A predictive software model with choices of parametric algorithms was investigated. Integration of metadata concepts with components and PCA tool chain was examined. Conforming to the PCA concepts defined by Morphware Forum [21], the milieu architecture and unifying system view were explored for integrating and elaborating SAAL, SAPI, compilers, runtimes, and the views developed during the research of this project.

The innovative ideas, concepts and approaches gained through this research project have had certain impacts to the research in the field of High Productivity Computing System (HPCS). For instance, the model-driven architecture approach for developing the morphable software can be applied to clusters and HPCS type systems. A paper has been prepared for submission to journal to systematically discuss the concepts and approaches developed in this program.

Notably, the research of the model-driven architecture in this project suggests a comparatively innovative perspective for software engineers to understand software development process. In convention, software design and decision-making are isolated from software coding and implementation. Time-consuming and painful work to redesign and re-implement solutions is inevitable when mission and runtime environment change. PCA application demands integration of design and implementation in order to rapidly make decision and automatically construct application.

Graphical modeling language can be conceived as a high level programming language for decision-making, which employs visual form instead of textual syntax to express the same semantics. Associated with different model compilers, the design models can be compiled to lower level implementations in textual languages, such as Matlab or C. As high-level abstract design, the graphical models are portable across multiple textual source languages. The portability is enhanced by the standard XML model representation.

Unlike a conventional programming language whose syntax and semantics are static, the graphical modeling language is configurable through metamodeling. Such modeling language can be configured to cover a superset of multiple conventional programming languages in terms of syntax and semantics. Even though initially configured to support a single source language, domain-specific model can be easily and rapidly reconfigured to cover broader range of source languages.

Meanwhile, the modeling tool allows metadata gained through performance monitoring and experiments to be mapped on components and their alternatives. This facilitates design space exploration based on input runtime metadata information.

The PCA mission build chain shows how source code implementation for components can be automatically extracted and composed into an application program. This capability of system

synthesis can be enhanced in the future through developing code generators to support automatic code generations for multiple languages.

For optimization, the PCA mission build chain conceptually shows how different design alternatives can be dynamically selected based on different metadata to produce changing application composition. The design space exploration applying heuristic searching techniques will be a topic for separate study in the future [11, 13]. PCA application is expected to accept user program code in multiple source languages, and be able to execute application program on various hardware environments. In the multilevel compilation, the intermediate code plays a role of virtual machine to support portability and interoperability. Multilevel compilation strategy highly reduces the complexity of compiler development effort.

Conclusion and Recommendations

To conclude, during the research of this project, a number of valuable and innovative concepts and approaches were explored. Several experiments were conducted and analyzed. These experiments demonstrated that the concepts and approaches studied during the research could be applied into the real world problem for PCA applications. These concepts include componentization of software, non-functional parametric information specifications, application system modeling, and dynamic morphing of PCA application program.

The research accomplishments gained from the research are suitable for future publications. Besides the academic accomplishment of the research, the toolkit prototyped and implemented under this project leaves wide space for future commercialization.

In addition, the research would benefit other research areas in the field of compute science. As a software engineering approach, the model-driven software development and engineering methodology employed and investigated in this research project suggested some innovative concepts for software engineering research. The experiments of this research reveal that the model-driven architecture can be an effective and promising approach to solve software engineering problems and increase productivity of software development. This innovative research perspective may lead to some reconsideration of conventional software engineering approaches. These topics are worthy of further research.

For the future work, the vision of polymorphous software environment independent of PCA hardware will be further developed. A continuing effort to develop the “metacode” concept would help produce a real product for PCA software development. Further research on metadata and component interactions will be needed.

The prototype of the PCA mission build chain developed in the project can be improved towards an Integrated Design and Development Environment (IDDE). Such an IDDE has identifiable benefits for program definition, optimization, and evolution, by addressing multiple concerns simultaneously. These concerns include optionality, morphability, and evolvability as primary software requirements.

Based on the current achievement, the future effort to improve the PCA mission building process may include the following specific tasks:

- Develop resource manager for handling runtime metadata;
- Develop performance monitoring tool that collects QoS for specifying off-line metadata in design time;
- Develop runtime scheduler for prioritizing runtime tasks; and,
- Develop optimizer by employing design space searching techniques for runtime morphing.

The Integrated Software Design and Development Environment (IDDE) will be suitable for modeling component-based software and hardware system, specifying and mapping mission parameters and resource constraints, automatically generating sequential and/or parallel programs, and compiling application program on targeted hardware for runtime execution.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principle, Techniques, and Tools*, Addison-Wesley, Reading, Massachusetts, 1986.

[2] "Autocoding Toolset: Software Tools for Automatic Generation of Parallel Application Software," http://www.mcci-arl-va.aa.psiweb.com/techpape.htm Date of access: October 8, 2003.

[3] J. Axelsson, "Analysis and Synthesis of Heterogeneous Real-Time Systems," Dissertation, Linkoping Studies in Science and Technology, 1997.

[4] J. Axelsson, "Architecture Synthesis and Partitioning of Real-Time Systems: A Comparison of Three Heuristic Search Strategies," *Proceedings of the 5th International Workshop on Hardware/Software Co-Design*, Braunschweig, March 24-26, 1997.

[5] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, and S. Asaad, "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," VLSI Design, Vol. 10, No. 3, pp. 281-306, 2000.

[6] V. Berzins, and D. Dampier, "Software Merge: Combining Changes to Decompositions", *Journal of Systems Integration, Special Issue on CAPS*, Vol. 6, No. 2, 1996.

[7] D. Dampier, Luqi, and V. Berzins, "Automated Merging of Software Prototypes", *Journal of Systems Integration*, Vol. 4, No. 1, 1994.

[8] B. P. Dave, G. Laksminarayana, and N. K. Jha, "COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems," *IEEE Transactions on Very Large Scale Integration*, Vol. 7, No 1, 1999.

[9] R. P. Dick, and N. K. Jha, "MOGAC: A Multi-objective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 10, 1998.

[10] B. Eames, T. Bapty, B. Abbott, S. Neema, and C. Kumar, "Model-Integrated Design Toolset for Polymorphous Computer-Based Systems," *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03)*, Huntsville, Alabama, April 7-10, 2003.

[11] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search," *Design Automation for Embedded Systems*, Kluwer Academic Publisher, Boston, Vol. 2, No. 1, 1997.

[12] "Extensible Markup Language (XML)," http://www.w3.org/XML Date of access: October 8, 2003.

[13] J. H. Holland, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[14] J. M. Lebak, “Preliminary Design Review: PCA Integrated Radar-Tracker Application,” External Report PCA-IRT-1, MIT Lincoln Laboratory, Lexinton, Massachusetts, April 2002.

[15] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason IV, G. Nordstrom, J. Sprinkle, and P. Volgyesi, “The Generic Modeling Environment,” *Proceedings of Workshop on Intelligent Signal Processing*, Budapest, Hungary, May 17, 2001.

[16] A. Ledeczi, J. Davis, S. Neema, and A. Agrawal, “Modeling Methodology for Integrated Simulation of Embedded Systems,” *ACM Transaction on Modeling and Computer Simulation*, Vol. 13, No. 1, pp. 1-22, 2003.

[17] E. A. Lee, "Overview of the Ptolemy Project," *Technical Memorandum UCB/ERL M01/11*, University of California, Berkeley, March 6, 2001.

[18] J. Li, A. Skjellum, and R. Falgout, "A Poly-Algorithm for Parallel Dense Matrix Multiplication on Two-Dimensional Process Grid Topologies," *Concurrency: Practice and Experience*, Vol. 9, No. 5, pp. 345-389, 1997.

[19] “MILAN: Model-Based Integrated Simulation Framework,” http://www.isis.vanderbilt.edu/projects/milan/index.htm Date of access: October 8, 2003.

[20] “Model-Based Integration of Embedded Software (MoBIES),” http://www.rl.af.mil/tech/programs/MoBIES/ Date of access: October 8, 2003.

[21] “Morphware Forum,” http://www.morphware.org/ Date of access: October 8, 2003.

[22] M. Palesi, and T. Givargis, “Multi-Objective Design Space Exploration Using Genetic Algorithms,” *Proceedings of the 10th International Workshop on Hardware/Software Codesign*, Estes Park, Colorado, May 2002.

[23] “PCA 101, Introduction to Morphware Software Architecture for Polymorphous Computing Architectures,” http://www.morphware.org/PCA101 Date of access: March 8, 2004.

[24] “Polymorphous Computing Architectures (PCA),” http://www.darpa.mil/ipto/programs/pca/index.htm Date of access: October 8, 2003.

[25] “Power Aware Computing/Communication (PAC/C),” http://www.darpa.mil/ipto/Programs/pacc/index.htm Date of access: October 8, 2003.

[26] M. Richards, D. P. Campbell, and K. M. Mackenzie, “The Morphware Stable Interface: A Software Frameworks for Polymorphous Computing Architectures,” *Proceedings* of the 28th Annual GOMACTech Conference, Tampa, Florida, March 31 - April 3, 2003.

[27] M. Richards, “Overview Briefing for the Morphware Stable Interface Facilitation Project,” Presentation to DARPA PCA PI Meeting, June 27, 2001, http://www.morphware.org/overview.pdf Date of access: October 8, 2003.

[28] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1988.

[29] A. Skjellum, and C. Baldwin, "The Multicomputer Toolbox: Scalable Parallel Libraries for Large-Scale Concurrent Applications," Technical Report UCRL-JC-109251, Lawrence Livermore National Laboratory, Livermore, California, 1991.

[30] A. Skjellum, "Driving Issues in Scalable Libraries," *Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, California, February 15-17, 1995.

[31] "Universal Data Model (UDM) Framework," http://www.isis.vanderbilt.edu/Projects/mobies/filedownloads.asp Date of access: October 8, 2003.

Glossary

Component
A component is a software construct or entity that is specified or designed to encapsulate functionality into a single unit, normally with specific interfaces (input-outputs). According to [26] and other general understanding of components in Computer Science, a component can be viewed as a load-balancing unit, an entity with a specific/well-defined computational functionality (such as a basic algorithm or data motion operation plus computation), or a compiler optimization unit (such as a cache reorganization step or copy step). Components may be code level component that encapsulate functions (such as a FFT) or virtual machine level component to encapsulate machine code and state needed to be viable in an execution environment. Component-based software can be viewed as a composition of a set of the components, where different compositions are chosen within and between programs in order to create hierarchies, sequences and dataflows, plus feedbacks. In the PCA mission build chain introduced in this report, a component is narrowly defined as a reusable software functional unit, particularly a function block in a sequence of program operations that is usually associated with a source code implementation file. Components often publish their requirements and their capabilities as part of "meta data."

Design Alternative
For a component there may be one or more algorithmic implementations that achieve the same functionality, while one of the options is chosen for application construction. These options are defined as design alternatives, and the capability to allow user select options is defined as user optionality. Sometimes, these are also called "poly-algorithms.[3]" Functionally equivalent design alternatives are statically modeled in design time and dynamically selected by scheduler at runtime to meet changing runtime environment and mission parameters. Design alternatives are resolved through design space exploration to produce optimal composition of application program.

Domain-specific Model
In the GME modeling environment (mentioned next), a domain-specific model is the design model of the target application system. A Domain-specific model visually represents the structure of the system, such as data flow, control flow, and hardware architecture. It is composed of the instances of objects defined in the metamodel and conforms to all the rules and constraints defined by the metamodel.

Generic Modeling Environment (GME)
GME is a modeling toolset developed at Vanderbilt University. GME is a configurable toolset that supports easy creation of domain-specific modeling. "The primarily graphical, domain-specific models can represent an application and its environment, including software modules, hardware resources, and their relationship" [15]. GME provides a graphical user interface for visually designing PCA applications through a two-step process including metamodeling and domain-specific application modeling. GME can also assist software developers to synthesize application programs from the graphical models.

[3] A term classically coined by Dr. John Rice of Purdue University in relationship to problem-solving environments.

Integrated Radar-Tracker (IRT) System

As a PCA benchmark application, the Integrated Radar-Tracker (IRT) system is "an end-to-end specification of a modern intelligence, surveillance, and reconnaissance (ISR) radar system. Motivated by a space-based radar application, it embodies all of the major attributes required in a defense-oriented PCA application test: both streaming and data-dependent threaded computation with multiple sub-types of each (*e.g.,* fast transforms *vs.* vector-matrix arithmetic in the streaming elements); heavy computational loads; and multiple application-level parallelization and morphing opportunities. Developed by MIT Lincoln Laboratory (MIT/LL), the benchmark consists of a MATLAB simulation that serves as an executable specification, sample data sets, spreadsheets for estimating the computational loading of the application, and instructions for installation and operation" [23].

Metadata

As described in [23], "PCA systems are designed to meet a variety of goals and constraints, and to function on a wide variety of platform configurations. PCA applications require a large amount of information in addition to the procedural definitions of programs in source languages. Examples of this information are the computing resources available on a particular host platform, the set of possible configurations of these resources, desired optimization goals for a particular piece of software, and computing resources required by a particular piece of compiled software. This set of descriptive and extra-functional information is known collectively as metadata."

Metamodel

As a generic configurable modeling environment, GME requires a configuration step that must be taken before anything meaningful can be done. The configuration process itself is also a form of modeling, the modeling of a modeling process. This is called metamodeling. The output of the metamodeling process is a compiled set of rules represented in the metamodel paradigm that configures GME for a specific application domain. The metamodel paradigm is based on the Unified Modeling Language (UML) [31].

Morphing

"The PCA program is building advanced computer architectures that can re-organize their computation and communication structure to achieve better overall application performance" [14]. The reorganization is referred to as morphing. A morphing process can be considered an optimization process that dynamically selects optimal alternative implementation for the changing execution environment determined by mission requirement and hardware resource availability. Morphing capability plays a critical role in reactive software system. The application software program produced through morphing is supposed to be adaptive to the architecture-specific execution environment.

Stable Architecture Abstraction Layer (SAAL)
The SAAL, a concept developed by the Morphware Forum, is “a set of portable APIs that encapsulate abstractions of the computing resources present in PCA devices, as well as the operations on those resources used by the PCA source languages and APIs. This portability layer abstracts and simplifies PCA hardware for the source languages, and provides a consistent abstract set of resource types and functional support requirements for PCA hardware developers. The SAAL portability layer also simplifies the deployment of new PCA platforms and new source languages and APIs by providing a single common target for each. New languages and APIs must only provide a mapping to the SAAL portability layer, instead of build tools targeting every possible target platform. New PCA platforms need only supply a compiler for the SAAL to work with all of the existing source languages and APIs” [23].

Stable Application Programming Interface (SAPI)
The SAPI is a concept developed by the Morphware Forum. In a PCA application, “the source languages and APIs collectively are referred to as the Stable Application Programming Interface (SAPI) layer. The top-level input at the SAPI level is processed by the high-level compiler (HLC) appropriate to the source language(s) used. The high-level compiler outputs SAPI code, which is the input to the low-level compiler (LLC) for the target PCA platform of choice**”** [23].

Virtual Machine (VM)
The VM is an abstract machine for which an interpreter exists (such as another program, or a computer processing unit). Virtual machines are often used in the implementation of portable executors for high-level languages. The high-level language is compiled into code for the virtual machine (an intermediate language), which is then executed by an interpreter written in assembly language or some other portable language like C or Java.