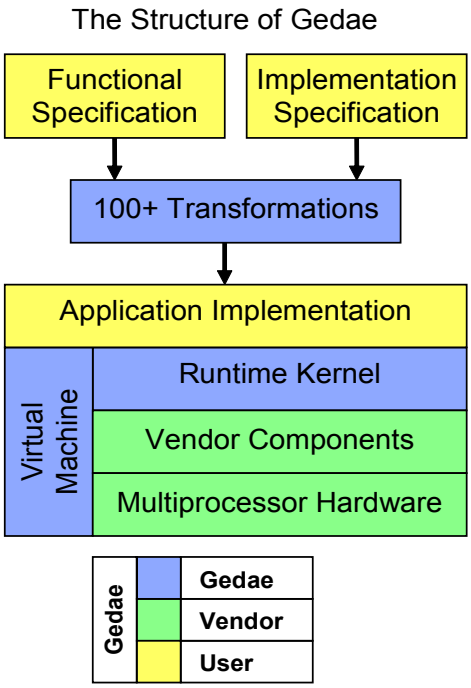
Gedae: Auto Coding to a Virtual Machine

William I. Lundgren, Kerry B. Barnes, and ***James W. Steed***
Gedae, Inc.
Phone: 856-231-4458
Email Addresses: {bill.lundgren, kerry.barnes, jim}@gedae.com

Gedae is an integrated application development environment. It has been under development since 1987 – though the concepts involved are rooted in much earlier work done in the areas of data flow and hardware simulation. In Gedae we have developed a language for describing an architecture-independent functional specification, a virtual machine on which the application runs, and transformations that create an efficient implementation of the application that runs on the virtual machine. In this paper we discuss three topics – the language, the virtual machine and the transformations.

The language was developed with two requirements – any functionality must be easily expressible, and the language must be transformable into an efficient implementation on the virtual machine. The Gedae Language consists of both the Gedae Primitive Language and the Gedae Graph Language. Much of the expressiveness is in the primitive description language. The language has over 50 expression features to define the behavior of functional ports. Port data flow requirements can be specified either prior to runtime (static) or at runtime (dynamic). Ports can add segment boundary markers on the data flow streams, thereby breaking the stream into independent data sets. Exclusive families of ports can send data down one branch or another to implement mode changes while maintaining coherent state vectors used by all the modes. Primitives can maintain their own local state variables and provide methods for execution, startup, termination and handling the beginning and ending of segment processing. The Gedae Graph Language allows the hierarchical development of graphs consisting of primitives, parameters and other Gedae graphs. The graph language can describe families of these entities to allow parameterized expression of parallelism. The resulting language permits



direct expression of signal and data processing algorithms, distribution for providing load balancing and fault tolerance, and application (or software, or mode) control.

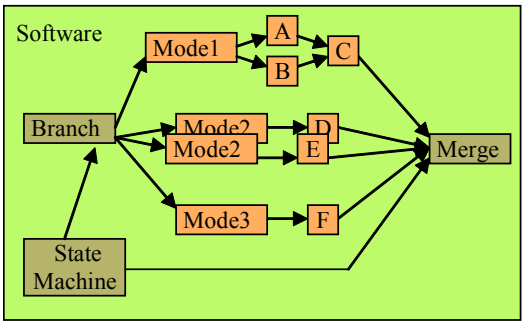
To achieve efficiency, the language and virtual machine were codesigned. The virtual machine contains a runtime kernel that executes components generated by the transformations. For example, the static scheduler executes predetermined execution sequences based on static data flow ports, and the dynamic scheduler executes groups of static schedules that interface through dynamic data flow ports. The virtual machine manages the segment processing and controls the efficient and timely transfer of distributed state vectors between processors. The virtual machine also allows for vendor specific optimizations of processing, such as, setting data transfer parameters. A thin layer

over the vendor-provide vector processing libraries allows primitives to execute efficiently.

One of the unique features of Gedae is the visibility of the implementation and the execution it provides. This visibility is possible because the language, the transformations and the virtual machine are all part of Gedae. The visibility allows the generation of detailed execution timelines and the symbolic viewing of any memory in the system. Primitive execution, queue state and data transfers between processors can be dynamically viewed when the application is running.

The transformations are the central part of Gedae and make possible the efficient execution of the application expressed in the Gedae Language on the Gedae Virtual Machine. The transformations are fully automated but can be guided by user supplied implementation parameters to control distribution, strip mining, data transfers, scheduling priorities (both static and dynamic), queue policies and memory management. Some of the transformations directly modify the graph into an equivalent graph to implement a user entered implementation decision. For the user to distribute a graph, the user specifies a partitioning of the graph and a mapping of the graph to individual processors. Gedae modifies the graph by inserting send and receive primitives that run on the separate processors and maintain the data flow and connectivity of the graph. The user does not have to modify the graph to achieve these results.

For example, the following graph has dynamic queues and is distributed to four processors by the user:



It is transformed into a new graph, as seen below, with send and receive boxes inserted to manage communications and dynamic queues also inserted to handle dynamic data flow boundaries. Other transformations include modifying the graph to implement strip mining of vectors, adding primitives to implement delay, and adding primitives to allow communication of the graph to the host program or to other Gedae applications. Data structures are also created to implement segmentation, mode control and distributed state coherency.

A sonar signal processing graph will be used to demonstrate how a graph is transformed into an implementation. It will be shown how the transformations can be used to modify the graph execution without changing the Gedae Language expression of the graph. The resulting implementations will be contrasted with how the same implementations would be achieved using traditional programming techniques.