

A Demonstration of Model Driven Development with Automatic Implementation using Gedae

William I. Lundgren
 Gedae, Inc.
 1247 N. Church Street, Suite 5
 Moorestown, NJ 08057
 (856) 231-4458 x104
willundgren@gedae.com

James W. Steed
 Gedae, Inc.
 1247 N. Church Street, Suite 5
 Moorestown, NJ 08057
 (856) 231-4458 x102
jsteed@gedae.com

Kerry B. Barnes
 Gedae, Inc.
 1247 N. Church Street, Suite 5
 Moorestown, NJ 08057
 (856) 231-4458 x103
kbarnes@gedae.com

Abstract

Gedae is a development environment designed to enable and automate a model driven development process. In this paper we illustrate the model driven development process using Gedae for an example real time target search and tracking algorithm. The paper goes through the full development process including development of a functional model and simulation of both the functionality and performance in the development environment. The emphasis for this tracking and pointing application is the functionality. We use a second application to illustrate the mode processing. In that application we have a RADAR environmental simulator that is simplified to one dimension but models a moving target with a varying signal return and a variable resolution RADAR system. We show the application searching for a target and switching to a tracking mode when it finds the target. It uses an MTI tracking algorithm. We also show examples of implementation specification, automatic implementation by Gedae and visualization of the implementation and execution. For those illustrations we use an FFT application (RADAR pulse compression) and target a processor with one general purpose processor and 8 DSP processors.

The process is broken into three distinct components as shown in Figure 1. The user develops a functional specification, and then separately defines an implementation of that functionality on an N-processor virtual machine. Gedae then makes dozens of compiler passes to create an efficient, functionally equivalent, and deadlock-free multi-processor application. The application we have chosen locates the source of a sound in 3 dimensions using an array of microphones. The generated application is a complete deployable system for multiple processors. This paper will provide a listing of the functional features that are implemented in the functional model of the application and the implementation features

that were used to make the application run in real time on the target system.

Functional Specification

We will now discuss the functional characteristics included in the application that were constructed during the model driven development process for this example. There are 3 phases of processing in the functionality. The first phase does spot formation on data from an array of microphones arranged in a circle track. For simplicity of the algorithms we assume that the surface is known and the sound source sits on the surface. We create a two dimensional image. The setup of the system is shown in figure 2. The second phase of processing does detection of sound sources captured in the audio map created by the spot formation algorithms. The third phase of processing is the conversion of the (x,y) location detected into pan and tilt angles to drive the gimballed camera. Also included, but not described in this paper, is an environmental simulator that simulates the train moving with the associated audio signal and camera image. See the top level block diagram of this system in figure 3.

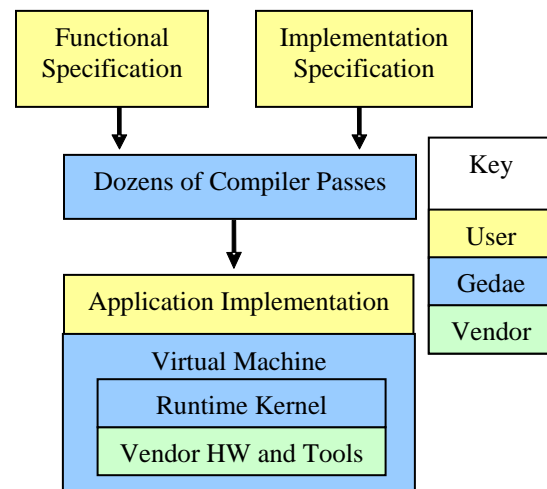


Figure 1 – Core Gedae implementation process

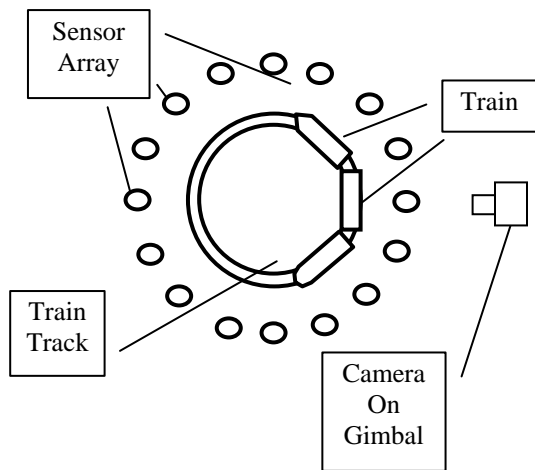


Figure 2 - The physical layout of the system

The SpotFormation box is shown in figure 4. The algorithm converts the stream of data into a series of sliding windows. It does this for each of the sensors. Once that is done the calculated delays from each spot location, a 64 by 64 grid, for each sensor is applied to “un-delay” the data stream. The resulting sensor signals are correlated. If the correlation is high then the signal originated from that spot. Study the block diagram in figure 4 to see the series of operations in this spot formation.

The block diagram in figure 5 is the detection algorithm. The algorithm works by first convolving a kernel with the image to smooth the image removing the higher frequency noise. We then find the region that exceeds some threshold and assume that represents the shape of the source of the audio – in this case the train.

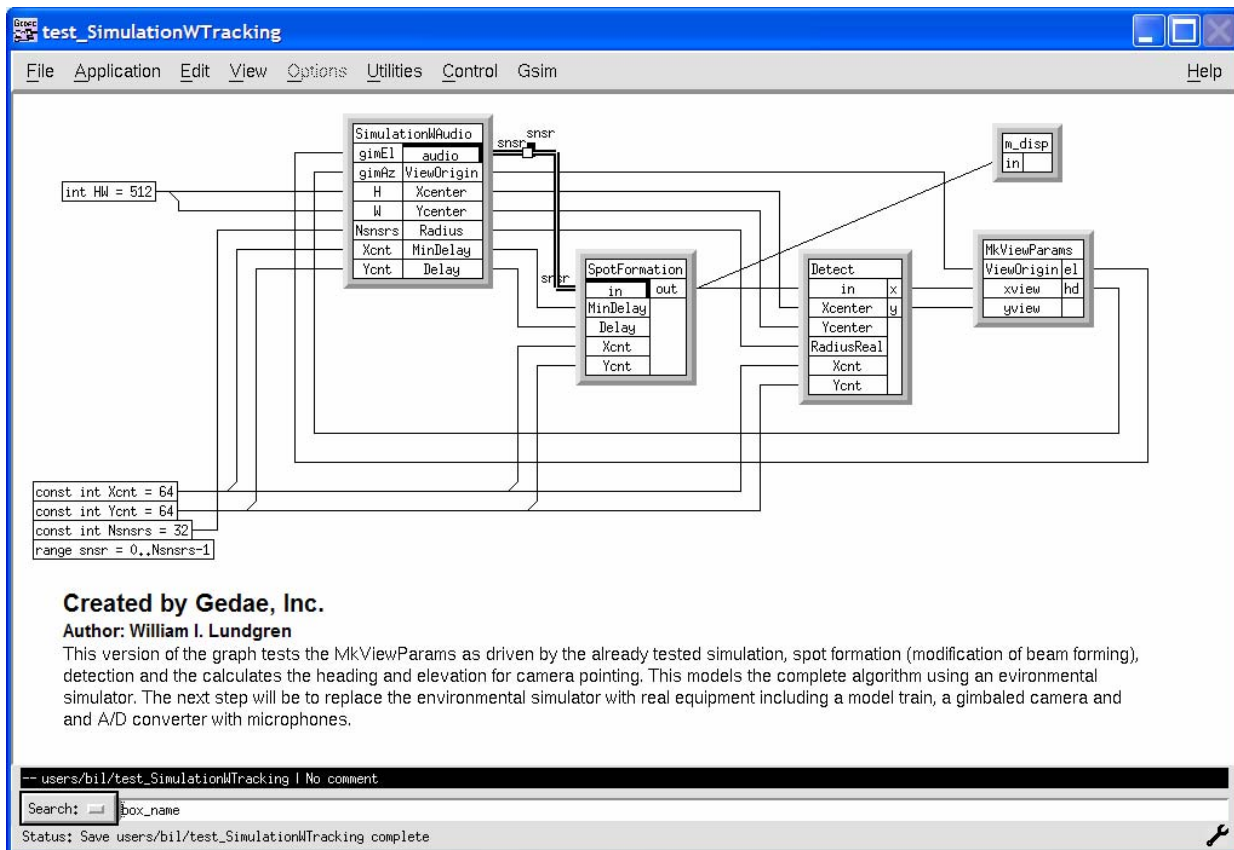


Figure 3 – The top level block diagram shows the major application components.

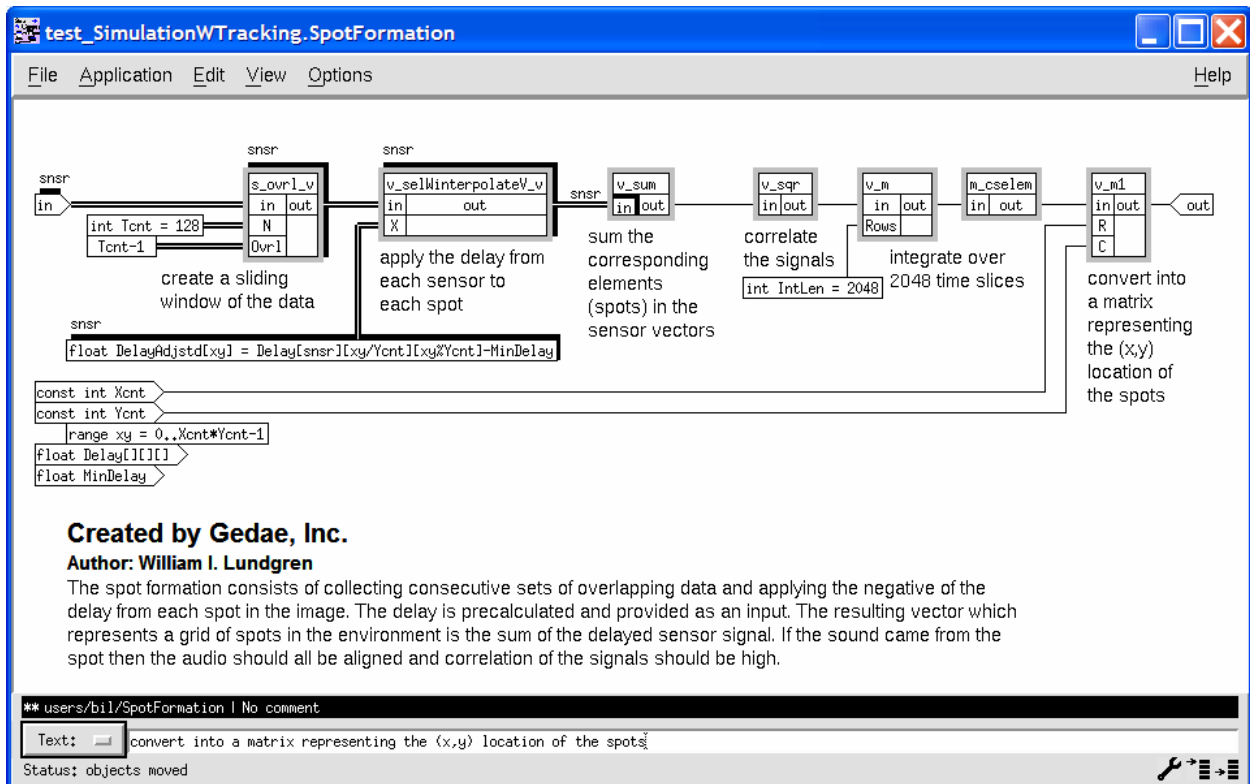


Figure 4 – This block diagram of the spot formation shows the chain of functions applied.

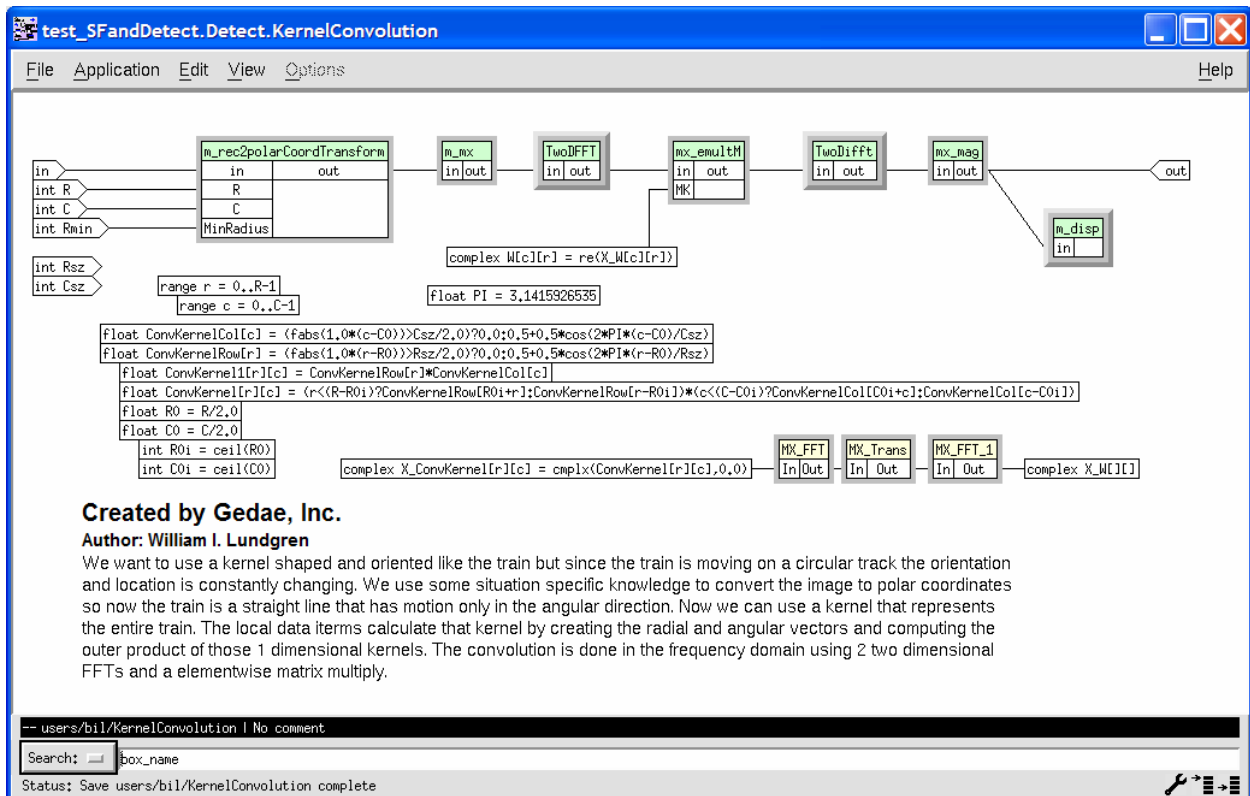


Figure 5 - This block diagram shows the structure of the detection algorithms.

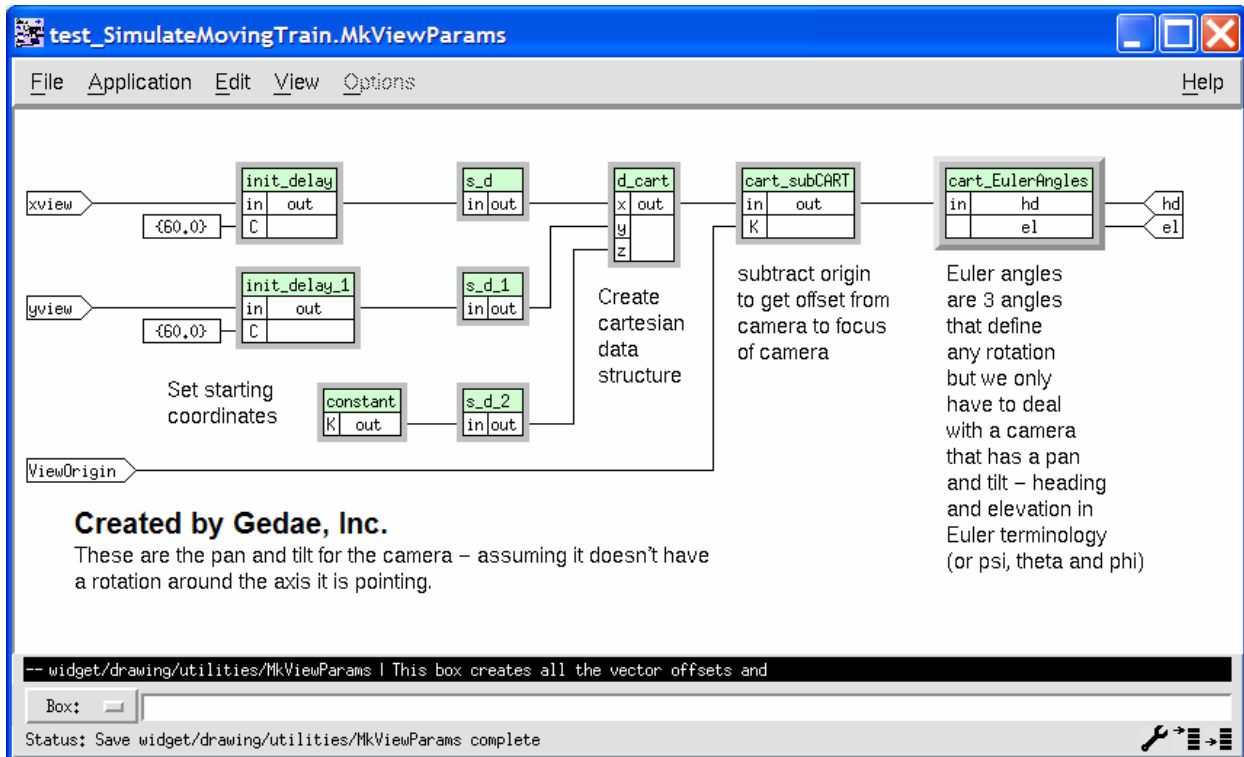


Figure 6 - This block diagram shows the processing that covers an (x,y) location into a heading and elevation angles used to drive the camera gimbal.

The last algorithm in the functionality is shown in figure 6. The view location is subtracted from the sound source location to get the vector from the camera to the sound source. The Euler angles of the vector are then calculated and the heading and elevation are the angles to set the gimbal to in order to point to the target.

Some results of the processing are shown in figure 7, 8 and 9. Figure 7 shows the sound map created by the spot formation. Figure 8 is the sound map filtered and transformed to polar coordinates. Figure 9 is the simulated image that results from the camera pointing at the target. The structure of the algorithm as tested by the simulated data is quite successful. Future work will apply these same algorithms to real sensor data and modify as necessary. The purpose of this paper is to show how functionality is structured in the model driven tool and the clarity of the functionality. These block diagrams successfully illustrate that point. We leave it to the developers using Gedae to create state of the art algorithms for signal, image and data processing.

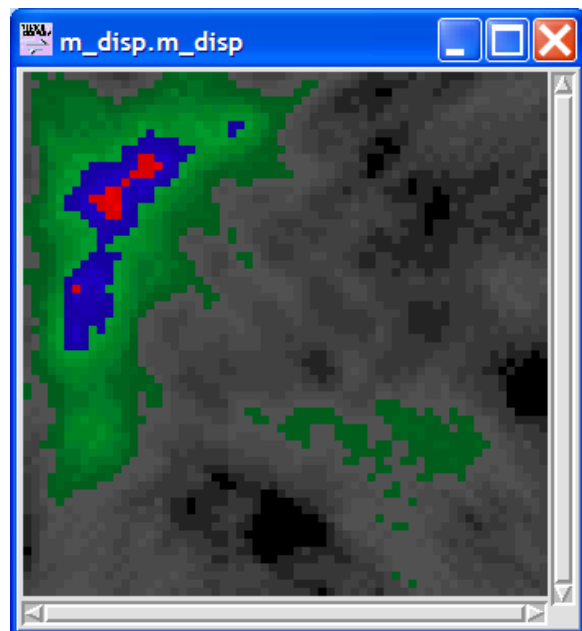


Figure 7 - This is the sound map created from the microphone inputs.



Figure 8 - This is the sound map after filtering and conversion to polar coordinates.

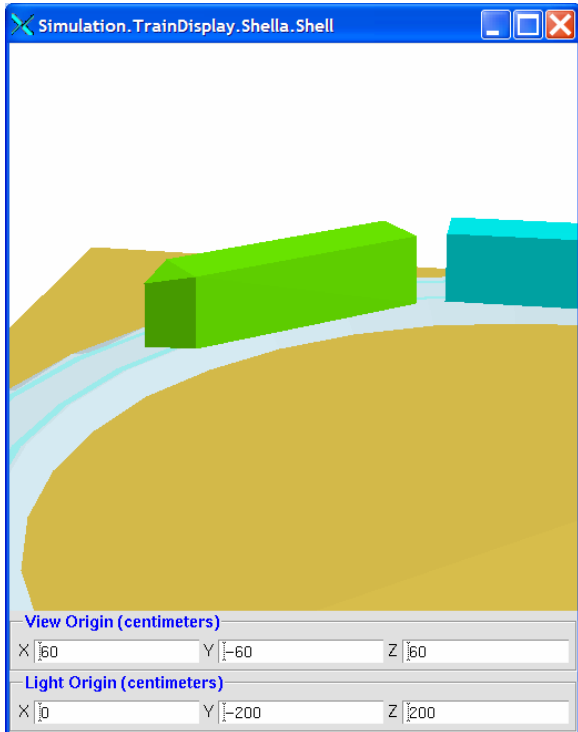


Figure 9 - This figure shows the results of the environmental simulator and the developed signal and image processing operations.

Implementation Specification

The characteristics of the implementation are specified through simple table-based GUIs. These tables provide many degrees of freedom for optimizing the implementation. Figure 10 shows a block diagram of a top level graph for testing the parallelization of the range processing algorithm. The algorithm itself is shown in figure 11. Figure 12 is the implementation table used to specify the partitioning of the functionality. It is a listing of all the functional components in the system in a hierarchical table. Note the family of range boxes is mapped to 8 partitions named SPE0 through SPE7. Figure 13 shows how the partitions are mapped to processors.

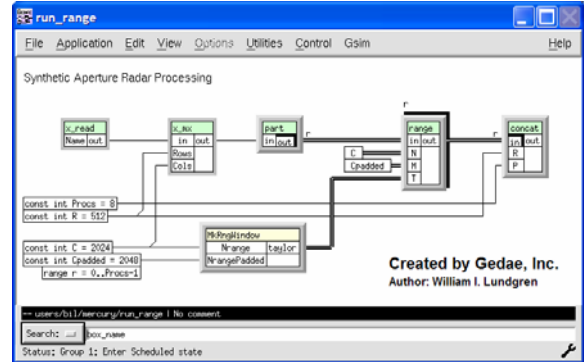


Figure 10 – This block diagram shows signal source, data partition box and family of range boxes that implement the range algorithm of a RADAR application.

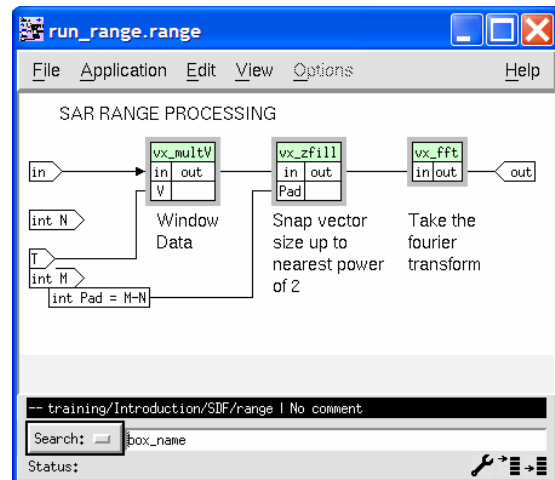


Figure 11 – This diagram shows the algorithm for RADAR range processing (pulse compression).

Name	Part	SubSched
x_read	src *	
x_mx	src *	
part	PPE *	
[0]range	SPE0="SPE"+\$1	
[1]range	SPE1="SPE"+\$1	
[2]range	SPE2="SPE"+\$1	
[3]range	SPE3="SPE"+\$1	
[4]range	SPE4="SPE"+\$1	
[5]range	SPE5="SPE"+\$1	
[6]range	SPE6="SPE"+\$1	
[7]range	SPE7="SPE"+\$1	
concat	PPE *	

Figure 12 - Partitioning table for defining how functionality will be partitioned for eventual mapping to an array of processors.

Name	CP ProcNum	System Name	Trace Size	Trace MemType	Params
PPE	99 *	gsim_host	10000	default	-d 0,001 *
SPE0	100=100+\$1	gsim_host	10000	default	
SPE1	101=100+\$1	gsim_host	10000	default	
SPE2	102=100+\$1	gsim_host	10000	default	
SPE3	103=100+\$1	gsim_host	10000	default	
SPE4	104=100+\$1	gsim_host	10000	default	
SPE5	105=100+\$1	gsim_host	10000	default	
SPE6	106=100+\$1	gsim_host	10000	default	
SPE7	107=100+\$1	gsim_host	10000	default	
src	98 *	gsim_host	10000	default	-d 0,001 *

Figure 13 - This table is used to map the partitions - PPE, SPE0-7 and src to logical processors in our multiprocessor target.

Name	Size	Length	Priority	Period	Retry	Memory Type	SH
Part SPE0							
Schedule 8	2084864	5	0		1 sc	default	
Schedule 12		1	0	0.5			
Part SPE1							
Schedule 7	2084864	5	0		1 sc	default	
Schedule 13		1	0	0.5			
Part SPE2							
Schedule 6	2084864	5	0		1 sc	default	
Schedule 14		1	0	0.5			
Part SPE3							
Schedule 5	2084864	5	0		1 sc	default	
Schedule 15		1	0	0.5			
Part SPE4							
Schedule 4	2084864	5	0		1 sc	default	
Schedule 16		1	0	0.5			
Part SPE5							
Schedule 3	2084864	5	0		1 sc	default	
Schedule 17		1	0	0.5			
Part SPE6							

Figure 14 - Table of threads created to implement the application. Note the large memory size.

Name	Size	Length	Priority	Period	Retry	Memory Type	SH
Part SPE0							
Schedule 8		8	0		1 sc		
Segment dma_2	16192	8				default	
Segment dma_3	16384	8				default	
Schedule 12		1	0	0,5			
Part SPE1							
Schedule 7		8	0		1 sc		
Segment dma_2	16192	8				default	
Segment dma_3	16384	8				default	
Schedule 13		1	0	0,5			
Part SPE2							
Schedule 6		8	0		1 sc		
Segment dma_2	16192	8				default	
Segment dma_3	16384	8				default	
Schedule 14		1	0	0,5			
Part SPE3							
Schedule 5		8	0		1 sc		
Segment dma_2	16192	8				default	
Segment dma_3	16384	8				default	
Schedule 15		1	0	0,5			

Figure 15 - Shows the thread table with the data size of each thread reduced to a manageable 13K bytes.

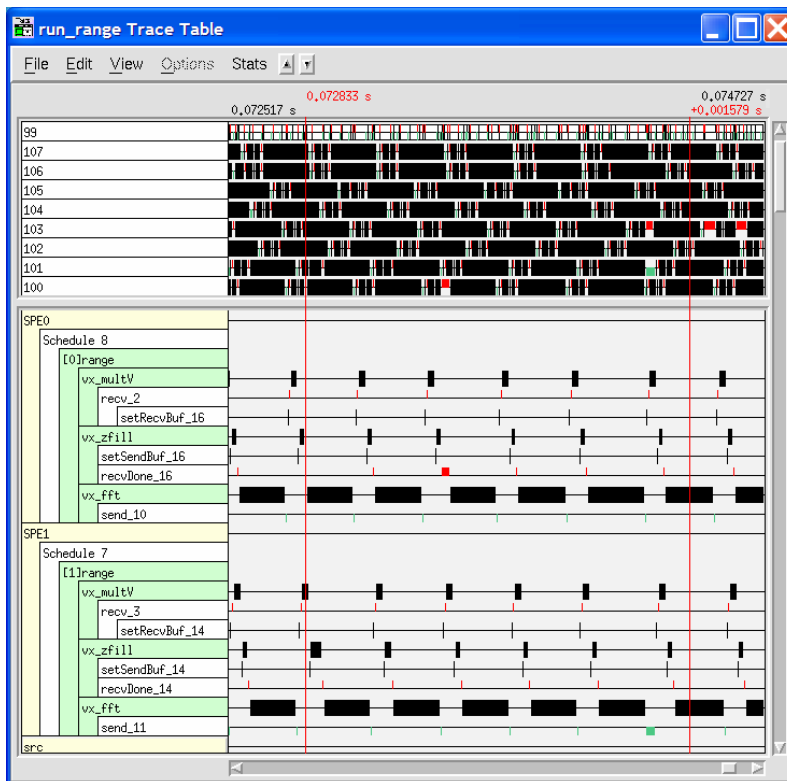


Figure 16 - Shows the trace table of the resulting application execution on 10 processors. This data was collected using Geda Simulation and is equivalent to what we will see on, for example, a 3 quad board system from Mercury Computers.

Implementation Generation and Visualization

From the implementation information specified by the developer, Gedae makes dozens of compiler passes to create a functionally equivalent, efficient multi-processor application. The nature of Gedae's flow graph-based functional specification provides for many methods of visualizing the implementation. The ability to visualize the implementation allows further optimizations to be done in an iterative development process.

- The Gedae thread scheduling algorithms are illustrated. Tools for visualizing the output of those algorithms are demonstrated
- The Gedae memory optimizations are illustrated. Dialogs displaying the entire memory map of the generated application are demonstrated.

As an example of these visualizations, Figure 14 shows the size of the generated threads, and Figure 15 shows the reduced size after automated strip-mining is applied. Figure 16 shows the Trace Table which provides detailed timing events for all execution on all processors in the system.

Conclusions and Future Work

Gedae is a model driven development environment that automates much of the implementation on a multiprocessor target. This paper shows the value added of a MDD development environment such as Gedae that provides separation of functionality and implementation and whose compiler algorithms recognize the structure of the target and optimize implementation of the functional model on that system. Future work will illustrate the tracking and pointing application as a real time system.