Visual Threads: The Benefits of Multithreading in Visual Programming Languages

Christian Mueller-Planitz and Robert R. Kessler

UUCS-97-012

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 13, 1997

Abstract

After working with the CWave visual programming language, we discovered that many of our target domains required the ability to define parallel computations within a program. CWave has a strongly hierarchical model of computation, so it seemed like adding the ability to take a part of the hierarchy and execute it in parallel would provide a good way of solving the problem. This led us to the concept of the Visual Thread and its associated components. Effectively, the Visual Thread allows the programmer to specify a part of the hierarchy and execute that part in parallel with the rest of program. We have used this implementation in several domains and demonstrated that it allows easy mapping of real world problems into our language. It eliminates most of the complexities often associated with programming parallel applications. We have also used a first prototype of our code generation system to translate CWave into Promela which allows us to verify correctness properties of the programs.

Visual Threads: The Benefits of Multithreading in Visual Programming Languages

Christian Mueller-Planitz
cmp@cs.utah.edu

Robert R. Kessler
kessler@cs.utah.edu

Department of Computer Science
3190 M.E.B.
University of Utah
Salt Lake City, UT 84112
U.S.A.

Abstract

After working with the CWave visual programming language, we discovered that many of our target domains required the ability to define parallel computations within a program. CWave has a strongly hierarchical model of computation, so it seemed like adding the ability to take a part of the hierarchy and execute it in parallel would provide a good way of solving the problem. This led us to the concept of the Visual Thread and its associated components. Effectively, the Visual Thread allows the programmer to specify a part of the hierarchy and execute that part in parallel with the rest of program. We have used this implementation in several domains and demonstrated that it allows easy mapping of real world problems into our language. It eliminates most of the complexities often associated with programming parallel applications. We have also used a first prototype of our code generation system to translate CWave into Promela which allows us to verify correctness properties of the programs.

1. Introduction

CWave is a dynamically typed, multithreaded visual programming language that runs on Microsoft Windows-95™ and Windows NT™ platforms. It makes use of state-of-the-art OLE™/ActiveX™ technology and provides seamless integration to other Win32 applications. It has been developed by the Component Software Project (CSP) research group in the Department of Computer Science at the University of Utah in cooperation with the CSP group at Hewlett Packard Research Labs in Palo Alto, California.

CWave has been designed as a general framework [EPHRAIM 90], [KOELMA 92] to handle several very different domains. The general goal is that the CWave user in a domain should be technically experienced in that domain, but not necessarily a computer programmer. In particular, CWave has been used in the following areas:

- Measurement devices - it has been used to directly provide the operations of individual low-level sensors and actuators;
- Measurement systems - with its ability to interact with numerous measurement devices and also interface to the enterprise, CWave has been used to generate combined measurement systems;
- Hardware simulation - a package of components has been developed that provides a simple, gate-level simulator;
- Wizard - it has been used as a visual wizard for constructing systems in a large component-based software engineering system;
- Web programming - CWave offers support for a Webserver that can be used to dynamically create Web pages and to visually construct CGI equivalent code;
- Software design and verification - We have used CWave to design parallel programs that were automatically translated and verified with the 'Spin' protocol verifier.

Due to the open architecture and easy extensibility of CWave we continue to seek new domains including some newly started work in robotics and robot sensor calibration. Thus, our primary goal is to develop an easy to use, customizable and extensible tool that can be used by non-computer scientists to develop solutions for their specific needs.

We start with a description of CWave, designed to provide sufficient detail to provide context for the description of '*Visual Threads.*' We follow with a discussion of additional features associated with the

threads that provide a powerful, yet easy to use method of parallel programming. Then we have a description of our techniques for program correctness verification and with an implementation of the Dining Philosopher's problem using Visual Threads.

2. General overview of CWave

The basic CWave metaphor [MEYER 93], [GLINERT 90] is to present the programmer with a palette of both predefined and user-created components as well as a drawing surface. The programmer chooses a component, drags it from the palette to the surface, sets properties of the component and then 'wires' component connections together to create an executable program. The execution model is similar to static data flow with a hierarchical structure for handling complexity. Program development proceeds by 'watching' the execution of the program and observing values that are produced along the way.

Figure 1 shows a sample program and the feature-rich CWave design environment which is used to develop, run, and visualize CWave programs. It's a multiple document, multiple view Windows NT application with support for zooming, a built-in debugger, and component management. The window on the left-hand side is a hierarchical tree view of currently loaded components. In the middle is the drawing area with a CWave program, which consists of four components (CONNECT_TO_MOTOR, TURN_ON, WAIT, TURN_OFF) wired to perform their actions in sequence. The browser is an HTML browser that points to the HTML documentation of the selected component.

In the example program in

Figure 1, each of the components control physical devices of a Lego model. We also have currently defined around 50 components that range in complexity from simple logic and arithmetic functions (AND, OR, PLUS, MINUS, ...) to more complex functions like CREATE_THREAD or invoke an OLE server to even more advanced components like SERVE_WEB_PAGE.

Components are grouped together in libraries which can be loaded at runtime. CWave comes with a well defined component programming interface which allows a programmer to create his/her own component library. For performance reasons component libraries (as well as the CWave framework) are written in C++. Additionally, CWave provides wrapper libraries to call Visual Basic™, Visual J++™ (Microsoft's implementation of Java™) and any other language that produces OLE capable code.

Components can be nested arbitrarily deep by using an EMPTY component as a container for other components which corresponds to a function in conventional programming languages.

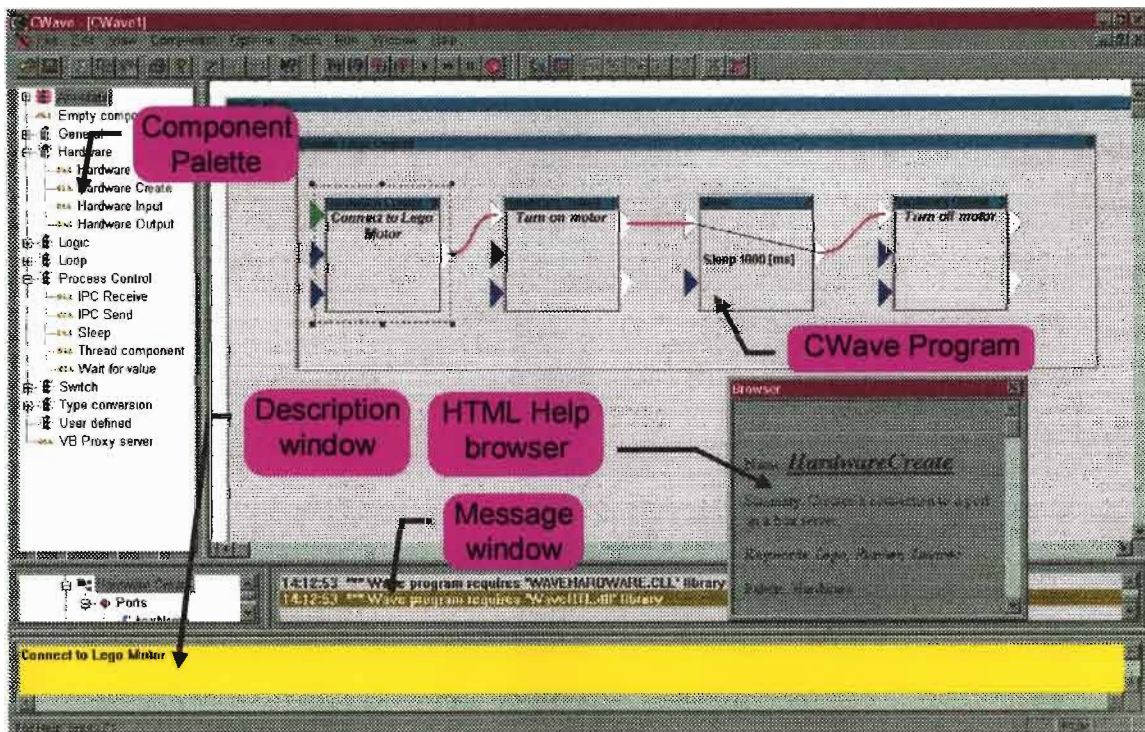

Figure 1: Sample program that shows the CWave environment.

Similar to components, CWave supports a set of predefined data types which can be extended by loading libraries at runtime. Currently, INVALID, INTEGER, FLOAT, STRING and POINTER are the primitive data types. In addition, CWave supports a bounded buffer (QUEUE), a single dimension array of any data type and DISPATCH which is used to pass OLE objects through wires.

Figure 2 shows part of a more involved program that depending on a pushed button will make a two-motor, mobile device turn a 12 inch circle or run in a square. This program demonstrates a for loop, multiple threads, and IPC communication. Finally, Figure 3 below shows one of the more complicated programs that we have written, which is a simulation of an automated system for doing automobile inspection and emissions testing.

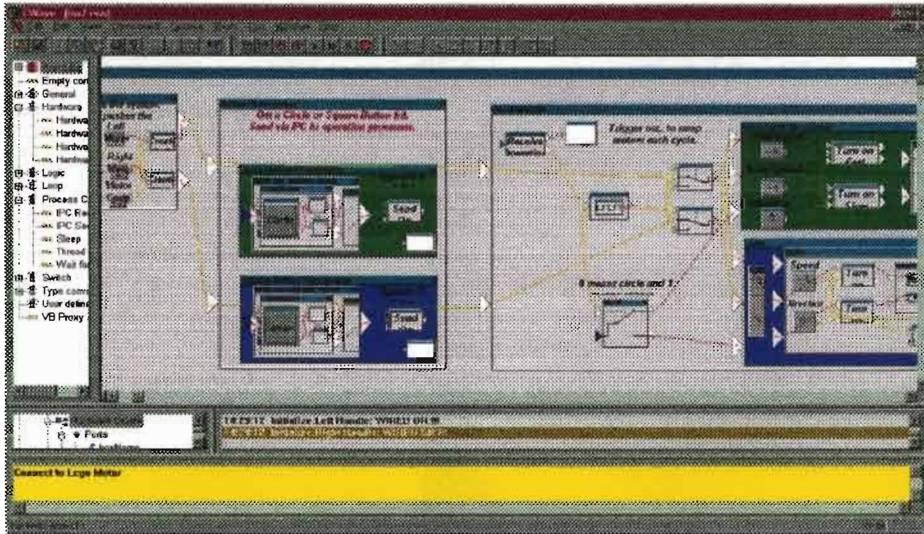

Figure 2: Program to tell a mobile Lego robot to move in a circle or square shape.

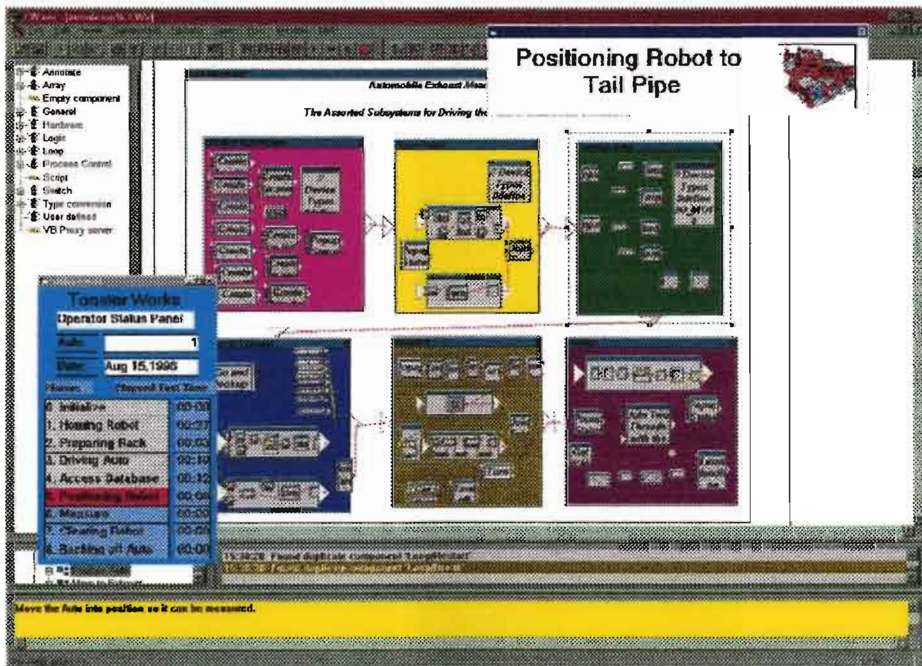

Figure 3: Automobile inspection and emission simulation program.

3. Parallel programming in CWave

With CWave being used for a broad class of domains, many of which have aspects of the "real world" [KESSLER 92], [GRISS 96], it became clear that we needed support for parallel processing within a program. We had already created components that supported broadcast and point-to-point distributed computing using "Inter Process Communication" (IPC) with pipes and mailboxes. Thus, our effort focused on developing extensions to CWave that supported parallel execution within a program. The most important of these extensions are visual threads and queues. However, before going into threads, we must first discuss the execution model.

3.1. Execution model

The execution model is similar to static data flow with a hierarchical structure for handling complexity. Similar to PROMELA [HOLZMAN 91] input ports to components act like a guard: if an input is not available, the component can't execute and won't produce a value which will most likely prevent execution of the next component.

CWave allows the user to specify optional default values (constants in other programming languages) on every input port. If a source of an input chooses not to generate a value, then the component will still execute using the default value as an input (provided that the default value is not of the special type INVALID which dynamically can prevent the component from executing).

CWave uses static data flow analysis to determine execution order at design time. Following this predetermined order, each component is executed if its input values are present. As mentioned earlier, the EMPTY component introduces hierarchy and is a container for other components. When it is the empty component's turn to execute, the component tries to execute all of its subcomponents before the execution continues with the next sibling. Thus an EMPTY component acts like a `<begin .... end>` statement in Pascal and gives the programmer implicit control over the execution order. If two output ports are connected to one input port ("wired-OR") the execution order of both sources is not deterministic. The programmer can force determinism by having one of the sources produce the special INVALID data type which will not overwrite any of the other data types.

3.2. Multithreading

Multithreading is accomplished primarily with the visual THREAD component. At first glance, the thread looks just like an EMPTY component. It has no substructure and no predefined input or output ports, but they can be added at design time in order to pass values asynchronously into and out of the thread (Figure 4). Like the EMPTY component, the THREAD component introduces hierarchy, allowing the programmer to specify components that will execute "inside" of the thread.

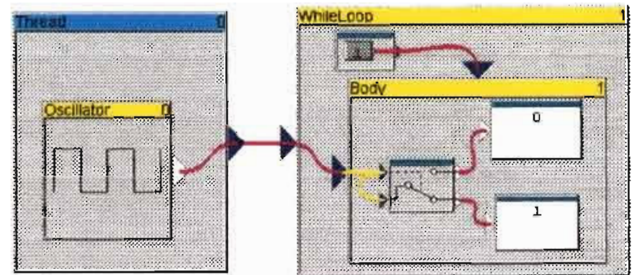

Figure 4: Asynchronous message passing

As with all other components, a thread component waits for its chance to execute until it is hierarchically its turn. Unlike the other components, when the outer thread reaches the THREAD component, a thread fork is performed allowing the THREAD component to execute independently of the outer thread. The outer thread proceeds to execute the next sibling of the THREAD component, while the THREAD component proceeds to execute its nested components using the same exact execution model. Graphically, when a component is executing, its title bar is highlighted so the programmer can see where execution is occurring. With the THREAD component, multiple locations may in parallel indicate that they are executing. Figure 5 shows an example of two threads executing, one generating the natural numbers, while the other is generating 0 and 1 in sequence.

Each thread object includes a number of capabilities that the user can customize. One feature allows the programmer to control how fast to execute the thread. When watching a thread execute, the slower speed allows the programmer to more easily track which component inside of the thread is currently executing. It is also very useful when writing producer/consumer types of applications and you want to throttle one or the other.

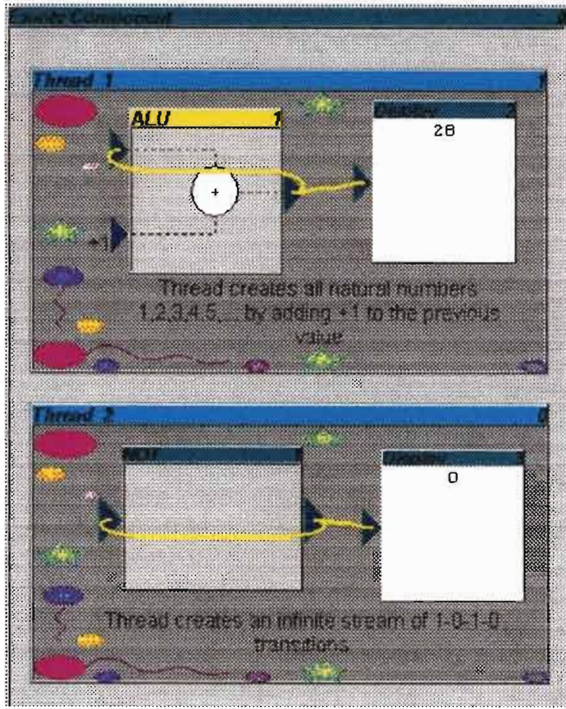

Figure 5: Demonstrates two independent threads.

The next important feature is the "Run only once" control. When enabled, the thread will be forked, its contents will be executed following the standard execution model and then the thread will be terminated. Often when executing a thread only once, you want the ability to synchronize an outer thread to wait for the thread to terminate. This can be accomplished with the WAIT_FOR_VALUE component which is used to "join" an outer thread with the results of the thread component. The WAIT_FOR_VALUE component and CWave's wired OR capability allows you to express situations where several threads are forked and you wait for the first one to produce a value, with the remaining values being ignored.

Finally, the last feature is the "SingleStep mode" capability. This is used for debugging purposes. When enabled, the outer thread will NOT fork, but instead will enter the thread and continue executing. This is handy when you want to verify the execution of a particular thread without other threads executing to distract you in your effort to verify that a particular thread is performing the correct operations.

Figure 6). We were able to verify the correctness of CWave programs by passing CWave generated Promela code directly to the 'Spin' verifier which checks for properties such as deadlock, livelock, and starvation. See HTTP://EASY.CS.UTAH.EDU/CWave/CodeGen/index.htm for further details.

3.3. Queues

As we previously mentioned, CWave offers support for passing values in and out of threads asynchronously as well as joining the results of two threads. Our first version only had threads and joins and no other features. Lab assignments of students using CWave in the Fall Quarter '96 showed that although asynchronous passing of values is powerful, it created many problems because of the student's lack of experience in parallel programming resulted in their being surprised when their program worked only some of the time.

Therefore, we decided to implement visual synchronization primitives that are easy to understand and remove the complexity of asynchronous values. We considered creating handles to native Win32 semaphores and passing these handles through wires to semaphore request- and release objects. We abandoned this model because it was too difficult to use. Instead we implemented a bounded buffer data type called QUEUE that meets the needs for simple synchronization. The model we use is that if an element is in the queue then the thread that references the value can proceed otherwise it blocks. In programs that are '*producer-consumer,*' the producer places values into the queue and the consumers take values out of the queue. These can occur at completely arbitrary speeds, allowing fast consumers to block when processing data from slow producers or slow consumers to execute at full speed when processing data from fast producers. Note - since our Visual Queue data type is protected by semaphores, one can implement simple semaphore type control with a single element queue. Our experience has shown that the QUEUE primitive covers most of the needs of the users of the system.

3.4. Correctness verification

Promela is a validation modeling language used by *Spin*, [HOLZMAN 91] a software verification tool for concurrent systems. Each Promela program consists of processes, channels and variables. The execution of every statement is conditional on its executability, much like our components must have values on all their input ports to execute. During Fall '96 we did experiments with translating CWave programs into Promela (see

Experiments have shown that a CWave programmer can easily generate Promela programs with more than 40 processes. The infinite nesting capabilities of CWave makes it possible to duplicate these existing processes again and again until the limits of Promela are approached.

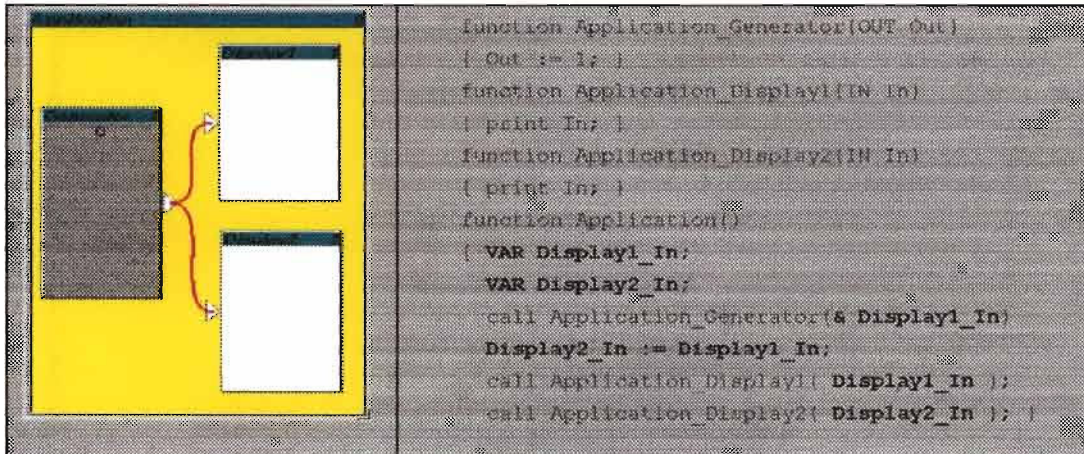

Figure 6: CWave to pseudo code which is expanded by macros into Promela

Naturally the generated code is not as efficient as if it would have been handcoded by an experienced Spin programmer but it offers a novice user an intuitive way to use a model checker and to verify his/her programs.

4. Example - 'Dining Philosophers

CWave with its threads and queues are perfect for implementing and visualizing standard synchronization problems like "*Dining Philosophers*" or "*Bounded Buffers*." These are standard problems that are taught in every operating systems class. Often students do not get practical experience implementing these due to the lack and/or difficulty of using threads in existing operating systems. Even if threads are available they have problems to debug the programs because existing debuggers for languages like C/C++ provide no visual clues about acquired locks, which thread is executing and why a program deadlocks.

We have used CWave to implement some of these problems. '*Producer/Consumer*' is trivial because of the visual queue primitive.. '*Dining Philosophers*' is more interesting and was programmed within minutes and is shown in Figure 7.

5. Related Work

There are many 'Visual Programming Languages' currently in use. Some of them are "despite their names" not visual but rather textual languages[1] and provide only a graphical GUI builder to make programming easier.

The systems that are most closely related to CWave are the graphically-based programming environments HP VEE (Visual Engineering Environment) [HENSEL 93] and National Instruments LABView that are both oriented towards test and measurements. Although CWave has grown to handle additional domains, it is useful to see how it compares with the other systems.

HP VEE	Domain specific / higher-level	Objective C / Byte code	Tolerant of inputs	VEE DLLs C/C++ via wrappers	lots / lots
LabView	Less domain specific/ lower level granularity	C / C	Specific types	external C code	lots / lots
CWave	Less domain specific / both (3-tier) Multithreading Visual Threads Visual Queues	C++ / "Open Framework"	C++ dynamic typing extensible	DLL, OLE, Automation, ActiveX script, Java, Visual Basic	toys / some

[1] e.g. Visual Basic or Visual C++

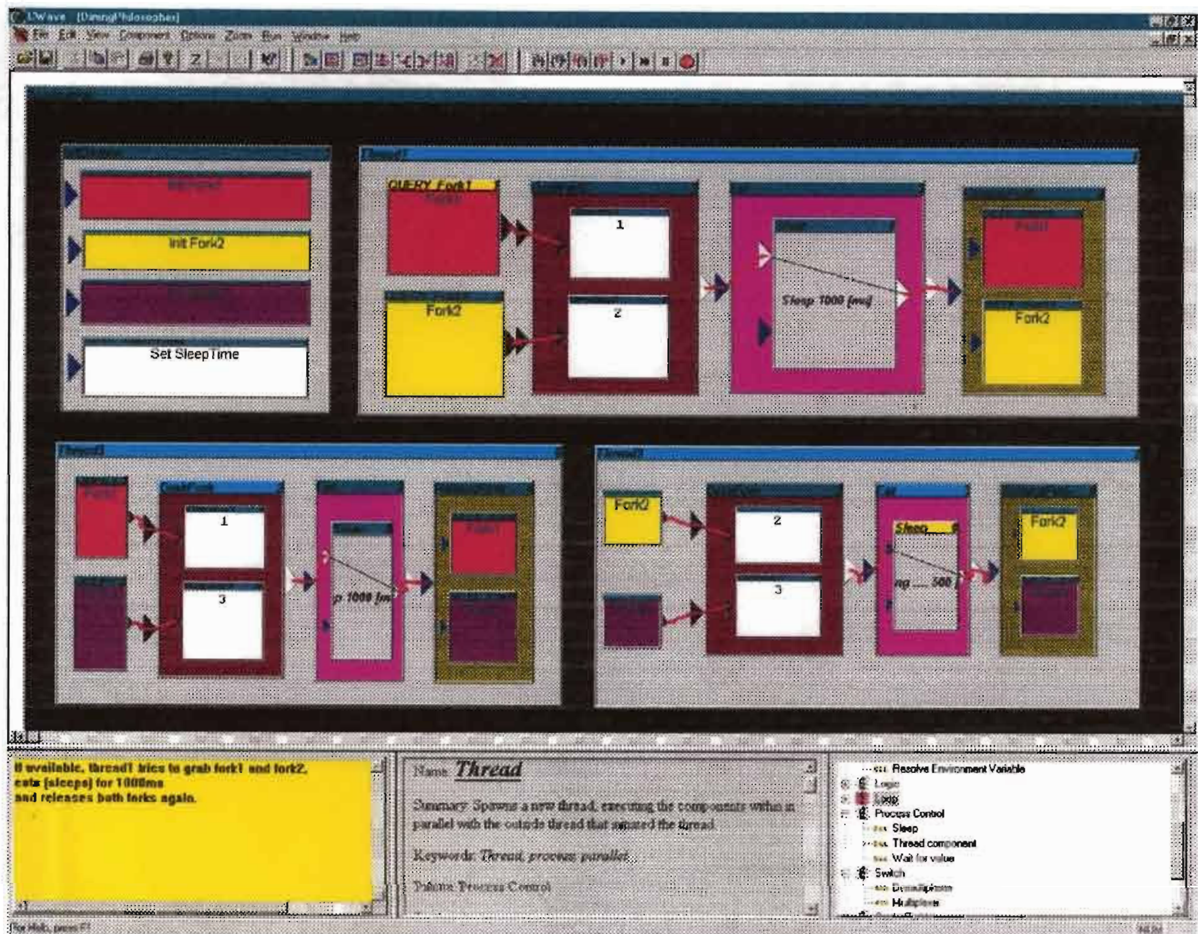

Figure 7: Dining Philosophers - note Thread2 is currently in the critical section while Thread1 is blocked

6. Conclusions

CWave has been used to simulate hardware (logic functions, gates, ...) as well as to design, debug and visualize software. The range of possible applications for CWave is broad:

- CWave can be used as a tool in a class teaching concurrency problems. With the appropriate code generators a concurrent CWave program can be converted into Promela and can be checked for correctness properties (e.g. deadlock and starvation).
- CWave can be used as a glue language for hooking together existing OLE applications. The fact that changes do not require recompilation gives opportunity for experiments like *'what happens if I do XYZ'*
- CWave can also be used as a teaching tool in hardware design classes. The standard release comes with components that do logic functions.

Our primary development goal was to make design of software as easy as connecting 'boxes' (components) together and to avoid cryptic C/C++ syntax.

Our addition of parallel primitives has enabled in-process parallel programming that is as easy as dragging a Visual thread component onto the drawing surface. The programmer does not have to struggle with huge programs that do various complex system calls like:

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    DWORD dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddress,
    LPVOID lpParameter,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId );
```

While talking to non-computer scientists we came up with the conclusion that there is a real need for making parallel programming more intuitive and easier. What is

most important is the overall design, the ability to maintain code and how much detail and knowledge of the language and of the system is required to come up with a working parallel solution.

Making use of CWave provides benefits even for experienced parallel programmers. Once a parallel CWave program has been developed it can be called from other languages. In other words CWave can be used for what it is good for: rapid application development of parallel programs. We believe that the open architecture of CWave provides a good basis for further extensions and research into adding parallelism to visual programs. CWave is available for free on our Website.

7. Acknowledgements

We thank the contributions of the Utah CSP group, in particular Nathan Dykman and Michelle Miller and our HP colleagues Martin Griss and Lorna Zorman. We hearby acknowledge Lorna for contributing the table comparing CWave with VEE and Labview.

8. References

[GLINERT 90] Glinert, E.P., Ed., "Visual Programming Environments: Paradigms and Systems and Visual Programming Environments: Applications and Issues", IEEE Computer Society Press, Los Alamitos, 1990.

[CHANG 90] Chang, Shi-Kuo "Principles of Visual Programming Systems," Prentice Hall, Englewood Cliffs, NJ. 1990

[EPHRAIM 90] Ephraim, P, "Visual Programming Environments"., Computer Society Press, 1990.

[KOELMA 92] Koelma, D., R. van Balen, and A. Smeulders, "SCIL-VP: a multi-purpose visual programming environment," Proceedings of the 1992ACM/SIGAPP Symposium on Applied Computing, 1188-1198, 1992.

[MEYER 93] Meyer, Bernd, Deklarative "Spezifikation visueller Sprachen durch graphische Beispiele oder: Ein Bild sagt mehr als tausend Formeln," 23. GI Jahrestagung, (Horst Reichel, ed.), Dresden, Germany, Springer Verlag, Berlin, 316-321, October 1993.

[BURNETT 95] Burnett, M., M. Baker, C. Bohus, P. Carlson, S. Yang, and P. van Zee, "Scaling Up Visual Programming Languages, Computer," 45-54, March 1995.

[KESSLER 92] Kessler and Evans, "DPOS: A Metalanguage and Programming Environment for Parallel Processors," International Journal on Lisp and Symbolic Computation, Vol 5:1/2, January 1992

[HOLZMAN 91] Holzmann, Gerard "Design and Validation of Computer Protocols," Prentice Hall, 1991.

[HENSEL 93] Hensel, R, "Cutting your Test Development Time with HP-VEE - An Iconic Programming Language," Hewlett-Packard Professional Books, Prentice-Hall, 1993.

[GRISS 96] Griss, M. and Kessler, R. "Building Object-Oriented Instrument Kits," Object Magazine, April 1996

9. Appendix - CWave Architecture

Figure 8 shows the architecture of the CWave frame work. On the left hand side is the CWave development environment which makes use of the Microsoft Foundation class (MFC). Linked into the development environment is the CWave runtime library which is shared with the component libraries. All calls from the development environment go through the runtime library. Although it is not recommended it is possible to run CWave programs without the graphical user interface by loading the runtime library which in turn loads the component libraries from any other Windows application.

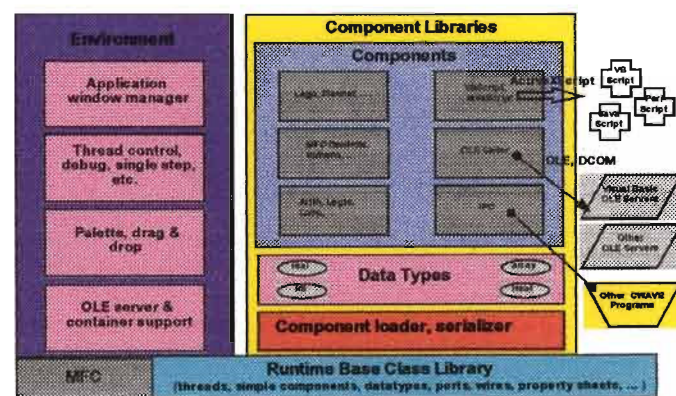

Figure 8: CWave architecture