"TaskWare" – A New Tool for Task Scheduling for Distributed System Design

Dr. David Rhodes, Dr. Benjamin Epstein

OpCoast LLC, 2530 Hooper Ave., Brick, NJ, 08723

Keywords: Task scheduling, co-design, allocation, resource contention, object-oriented design, CASE, analysis of alternatives, task graph, task modeling, UML

ABSTRACT

We describe herein *TaskWare®*, a novel tool for the object-oriented design and simulation of large-scale software systems. The tool takes a high-level view of systems through the application of *task modeling*, enabling a performance assessment of software-based systems containing thousands of software elements distributed across hundreds of hardware platforms, the latter of which intercommunicate via generalized internetworking. Realistic effects of process activation, communication overhead, resource contention, resource failure, and other effects found in large-scale systems can be analyzed with our approach. In addition, we include the *analysis of alternatives*, as expressed through deployment studies (i.e., assignment of software processes to specific processor hardware) and software system design studies (e.g., use of different software design approaches for implementing the same functionality). Although TaskWare remains work in progress, we describe the tool's current status and capabilities.

1. OVERVIEW OF TASKWARE

TaskWare is a Web-based tool that applies task modeling to facilitate the quantitative analysis, design and optimization of distributed software systems prior to full-scale development. Much of our effort towards its development has leveraged our previous work in the task modeling area (see [1] through [5]) as well as by utilizing other published techniques. While task-scheduling and allocation have been the subject of many research projects (for an overall summary of the field, see ref. [6]), little of this technology has transitioned to commercial, easy-to-use tools.

Figure 1 describes the overall multi-tiered architecture of TaskWare. The Web-based orientation of the tool enables users to access it from anywhere to facilitate multi-member teams in design sharing and concurrent system development. The tiered architecture allows for separate development of TaskWare's elements, extensions to the system, interfaces to off-the-shelf design tools, and porting of the toolset to different computing platforms.

Where CASE-based and other existing tools operate well at later stages in the computing system development cycle (such as at the target-system code generation phase), TaskWare is aimed at the early-stages of the development process. By providing analytical assessment of various design implementation choices prior to any code implementation, TaskWare provides the feedback necessary to make and guide critical, high-cost impact, early-stage implementation choices. Likewise, TaskWare addresses the problem of later-stage system deployment by optimally allocating processing tasks to processing elements. This can be useful when legacy code must be optimally integrated into a newly designed distributed system environment.

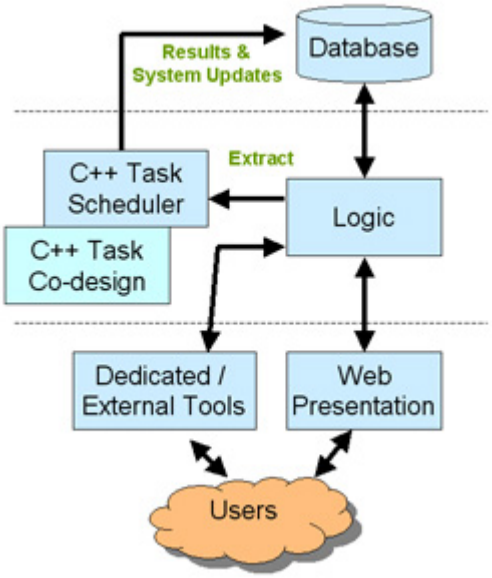

Figure 1. TaskWare implementation overview.

The current implementation of TaskWare uses MySQL [7] as the database for storing elements of the design in consideration, PHP [8] for server-side scripting with the GD [10] module for dynamic graphics creation (operating on an Apache Web server [9]). These components, along with a C++ task scheduling tool derived from earlier work [1] and [11], form the TaskWare prototype. TaskWare's present Web implementation includes database and PHP scripts that provide viewing and smart edit capabilities. The database currently uses 28 tables, although nine of them are

considered the 'primary' information tables covering the parameters of: Processor Type, Interface Type (IFace), Media, Task , Message (exchanged between tasks), TaskGraph (a high level design of Tasks and Messages), Object (a container or grouping of Tasks corresponding to an object in object oriented programming languages or a component in UML), RunTimeTable, and Study (an instance of an interconnection of Processors and their IFaces, Media, and TaskGraphs).

Entries in the nine major tables—each representing a significant hardware, software or system definitional element—can have arbitrary attributes attached to them. An attribute is a name/value pair where values can be defined to be of type number or string. TaskWare supports a single inheritance system that can be used to form a class hierarchy among elements. For example, the definition of the 'Pentium 4 PC' and 'Super PC' Processors can derive definitions from 'An old Pentium PC.'

A basic task scheduler analysis module has been interfaced with the TaskWare database described above. Its code is written in C++ and derives from our prior task scheduling and co-design work [1]. The scheduler uses the MySQL API to interface with the database and begins running and extracting information from Studies based on user commands provided via a Web interface. For a given Study, the scheduler extracts the processor/interfaces, media, task graphs, run time table and configuration (mapping of objects to processors) data. Note that the task graph portion of the data includes tasks and their inter-messaging as well as the overall task graph execution requirement (e.g. one-time or periodic). To support object-oriented software systems, we allocate objects to processors and not individual tasks. Allocation of an object implies that all of its tasks are similarly allocated of course.

Based on the task allocation, inter-task messaging routing is determined. At the present time, this is very simple and can only handle a 'single hop' but multiple hops will be addressed in future versions. In this single-hop form, a single 'communication task' is used to model the effect of communication. The duration of the task is the message size (set with the messageSize attribute on the Message) divided by the Media bandwidth (set with the bandwidth attribute on the Media).

Given the above input information, the task scheduler executes and determines the start and end times for each task instance (including communication tasks). This is the act of determining the task schedule and is the prime function of a task scheduler. Currently, we have provisions in the database for defining deadlines on tasks, which can be soft or hard. A hard deadline is one that must be met while a soft deadline is one which uses an expression evaluation to contribute to a penalty function. The combination of penalty contributions and meeting of hard deadlines form a metric that can then be used in system optimization. At present, deadlines are not evaluated.

2. EXAMPLE

Figure 2 depicts a simple task graph entitled 'WebUse.' This graph illustrates a single HTTP request and response between a Web client (e.g., browser) and Web server. It is composed of two objects, called 'ClientWare' and 'ServerWare' that intercommunicate via messages 'wm1' and 'wm2' of types 'HTTPreq' and 'HTTPresponse' requiring 72 and 4900 bytes respectively. The ClientWare object uses two tasks, called 'C1' and 'C2', which are of task types HTTPreq and HTTPrecv respectively. Note C1 and C2 can be viewed as representing object "operations" in UML parlance, or "methods" in object-oriented programming convention—here the task 'C2' is an instance (use) of the method HTTPrecv in the object ClientWare. The server object 'ServerWare' utilizes a single task called 'S1' of type 'HTTPresp'. The overall task graph is meant to represent a single Web page client request (from C1), server response (by S1) and client receipt (by C2).

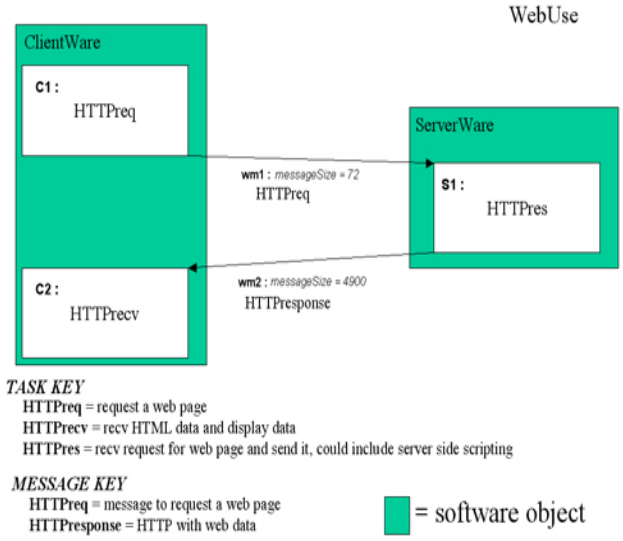

Figure 2. The 'WebUse' task graph.

3. DEPLOYMENT ALTERNATIVES

When alternative systems are studied wherein each alternative utilizes the same software tasks and structure, but where tasks (and their object containers) are allocated to processors differently, the resulting alternatives are called **deployment alternatives**. That is, the software structure and associated tasks have not changed, but the processors that tasks are executing on have changed. By re-allocating processors, changes to the message routing in the system also occur.

Figure 3 shows 'Study4' which uses two task graphs: a Web1 task graph (as an instance of the WebUse task graph given above) which executes every 50 seconds, and the second task graph SSL1 (instance of SSLauth)

which executes every 200 seconds. As can be seen, the deployment is such that the client object of Web1 is allocated to PC Station 1 (which is an instance of the Pentium 4 processor) and the Web1 server object likewise is allocated to Server 1 (an instance of a Sun Server). As for the SSL1 task graph, the client object is allocated to PC Station 2 (also a Pentium 4 processor) and both the SessionMgr and SSLauthorizer objects are deployed to Server 1. The interface called 'eth0' for each of the processors is connected to the LAN 105 media, of type Ethernet. Names such as 'LAN 105', 'Pentium 4 PC', etc. are of course free choices made by the user as they define the system. Alternative deployment configurations to Study4 may also be tried, such as by allocating the SessionMgr and SSLauthorizer objects to Server 2 (not shown).

Unlike deployment alternatives, software-system alternatives utilize different tasks, task graphs, media, etc., where each alternative software system design presumably attempts to accomplish the same job or function. These alternatives must be captured as separate studies in TaskWare.

4. ANALYSIS RESULTS

Our prototype task scheduling application has been developed to interact directly with the system definition database and return results to it. All user interaction is via a Web interface (see Figure 4). The 'Run Task Schedule(…)' button triggers the scheduler, while passing parameters regarding which analysis is to be performed. After successful completion, the tool writes the status along with detailed schedule information back to the database and some other results. Shown is the 'demand rate', which is the number of tasks launched per unit time. Note that 'Server 2' is not used in configuration 1 of Study4 so its demand rate is 0. The 'Graph Task Schedule(…)' button can then create a graphic of the schedule.

An example of TaskWare's Web-based graphics output is shown in Figure 5. Time increases in the negative y-axis and the x-axis depicts each resource (processors or media) in the study. TaskWare colorizes (not visible here) the tasks based on which task graph they are contained in. Recall that each of these task graphs is set to periodically repeat, in this case at intervals of 50 and 200 seconds respectively. Note that use of the LAN 105 media resource is also part of the schedule, although the short messages in this study result in only short bursts of use.

Since the tasks of SSLauth are essentially serial there isn't much advantage to using the second server as is demonstrated here. The present TaskWare prototype supports non-preemptive scheduling, but if the processor where defined to use preemptive scheduling (and the tool updated to support it), then one task could interrupt another task if it had higher priority. Our co-design work ([1] – [5]) has developed preemptive schedule analysis and a data arrival model for process activation overheads and message processing; however, a simpler scheduler was adapted for demonstration purposes. Of course, the example presented herein is very simple.

More advanced metrics remain to be implemented in TaskWare. These might include: average and worst case processor demand rates, resource contention metrics, interface queue depths, critical path assessment, etc. Moreover, hard and soft deadline evaluations remain to be implemented as well.

ACKNOWLEDGEMENT

This work was supported by the US Navy SPAWAR, SBIR Phase I contract N65538-04-M-0102, *Task Scheduling for Distributed System Design*, (Topic N04-069), 4 Nov. 2004.

REFERENCES

[1] David L. Rhodes, Wayne Wolf, "RAGS: Real-Analysis, ALAP Guided Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, No. 8, August 2001, pp. 931-41.

[2] David L. Rhodes, Wayne Wolf, "Co-Synthesis of Heterogeneous Multiprocessor Systems Using Arbitrated Communication," *IEEE Int. Conf. Computer-Aided Design (ICCAD'99)*, Nov. 1999, pp. 339-42, San Jose, CA.

[3] David L. Rhodes and Wayne Wolf, "Overhead Effects in Real-time Preemptive Schedules," *IEEE/ACM 7th Int. Workshop on Hardware / Software Co-Design (CODES'99)*, 3-5 May 1999, Rome, Italy, pp. 193-7.

[4] Robert P. Dick, David L. Rhodes and Wayne Wolf, "TGFF: Task Graphs For Free," *IEEE Int. Workshop on Hardware/Software Co-Design (CODES/CASHE'98)*, 15-18 March 1998, Seattle, WA, pp. 97-101.

[5] David L. Rhodes, Wayne Wolf, "Allocation and Data Arrival Design of Hard Real-time Systems," *IEEE Int. Conf. on Computer Design (ICCD '97)*, 12-15 Oct. 1997, Austin, TX, pp. 393-9.

[6] H. El-Rewini, T. G. Lewis, H. H. Ali, *"Task Scheduling in Parallel and Distributed Systems,"* Prentice Hall, 1994.

[7] http://www.mysql.com/

[8] http://www.php.net/

[9] http://www.apache.org/

[10] http://www.boutell.com/gd/

[11] Objective Force Warrior: C4ISR Architecture Study, October 2003.

BIOGRAPHIES

David Rhodes is the founder of OpCoast LLC, a small business dedicated to R&D in advanced network design. Dr. Rhodes specialties include large scale design and simulation of distributed systems, network protocol designs, and task modeling. He received the PhD and MS degrees in Electrical Engineering from Princeton University, and the BS in EE from Rutgers University.

Benjamin Epstein handles Special Projects at OpCoast. His specialties include wireless simulation and design, multi-level security for tactical systems, and electronic surveillance for law enforcement and intelligence agencies. He holds a PhD in Bioengineering from the Univ. of Pennsylvania, MBA from NY University, and a BS in EE from the University of Rochester.

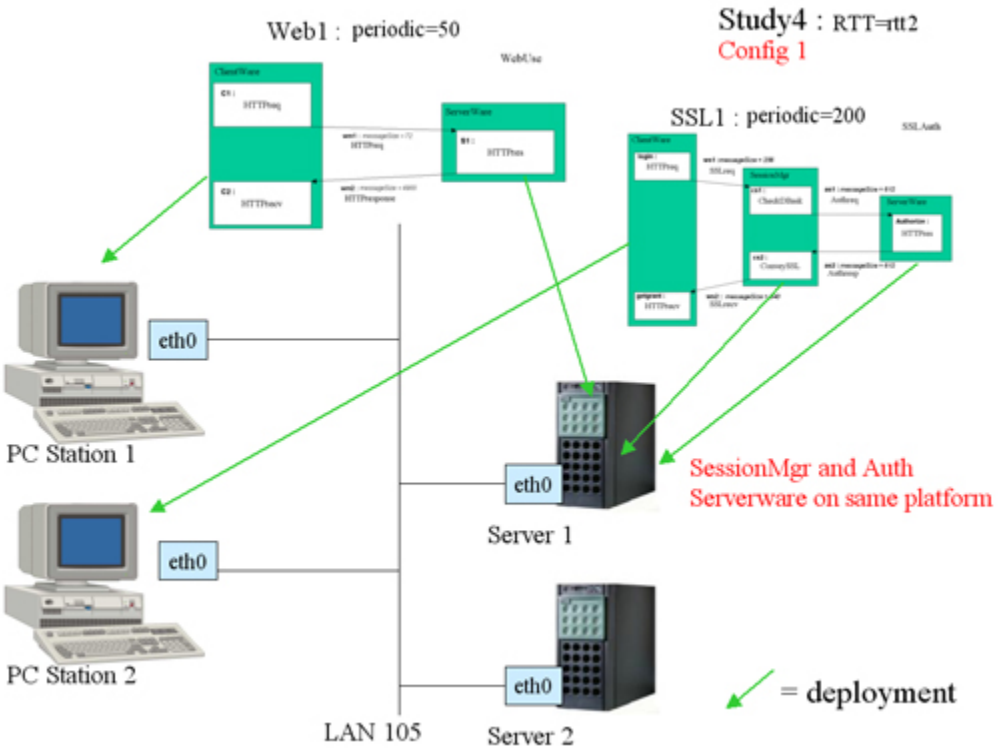

Figure 3. Study4 in Configuration 1.

Software Configuration (Config #1)

Run Task Schedule(1,Study4) | Graph Task Schedule(1,Study4)

TaskWare v0.3

PC Station 1: *Demand Rate* = 0.042
PC Station 2: *Demand Rate* = 0.012
Server 1: *Demand Rate* = 0.039
Server 2: *Demand Rate* = 0
LAN 105: *Demand Rate* = 0.054
Normal Tool Completion

Description	Task Graph	Object	Mapped to Processor
conf11	Web1	ClientWare	PC Station 1
conf12	Web1	ServerWare	Server 1
conf13	SSL1	ClientWare	PC Station 2
conf14	SSL1	ServerWare	Server 1
conf15	SSL1	SessionMgr	Server 1

Software Configuration (Config #2)

Run Task Schedule(2,Study4) | Graph Task Schedule(2,Study4)

TaskWare v0.3

PC Station 1: *Demand Rate* = 0.042
PC Station 2: *Demand Rate* = 0.012
Server 1: *Demand Rate* = 0.033
Server 2: *Demand Rate* = 0.006
LAN 105: *Demand Rate* = 0.066
Normal Tool Completion

Figure 4. Snapshot of the Web page for triggering the schedule analyzer. Note the configuration table of Config#2 is not shown in the figure.

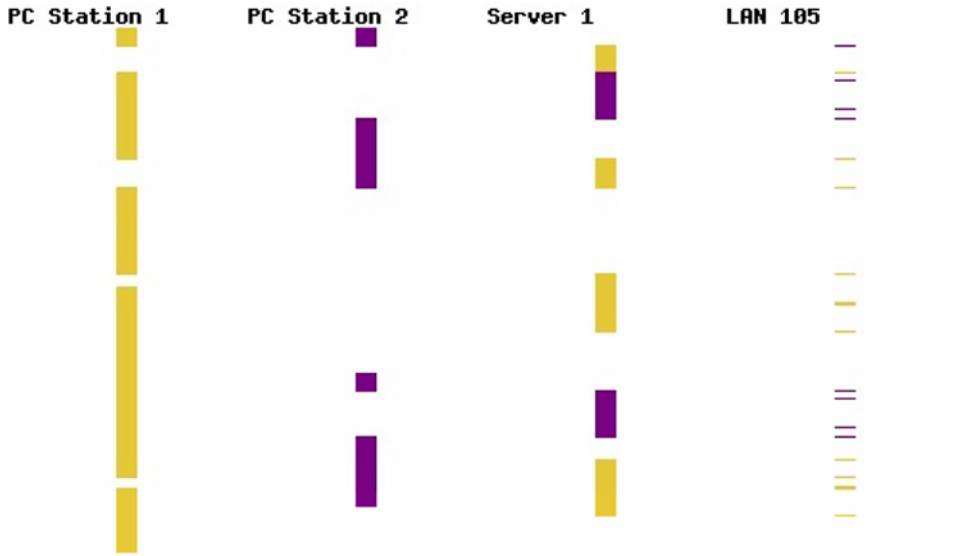

Figure 5. Part of the Study4 Configuration 1 task schedule generated by TaskWare.