

Empirical Evaluation of OpenCCM for Java-based Distributed, Real-time, Embedded Systems *

Shruti Gorappa and Raymond Klefstad

Department of Electrical Engineering and Computer Science
University of California, Irvine, CA 92697, USA
{sgorappa,klefstad}@uci.edu

ABSTRACT

Component technology can overcome many limitations of conventional Object Request Brokers (ORBs) in developing distributed, real-time, and embedded (DRE) applications. Component technology has particular advantages for building large-scale DRE systems. The CORBA Component Model (CCM) enables the composition and reuse of software components and the configuration of key non-functional aspects of DRE systems such as timing, fault-tolerance, and security. However, the CCM can introduce additional overhead to the runtime performance and code size of middleware. Hence, the overhead for using the CCM needs to be evaluated to determine if the CCM can be effectively employed in the design of high-reliability DRE applications. In this paper, we empirically evaluated the performance of OpenCCM, a Java-based implementation of the CCM standard, when configured with two Java ORBs: with ZEN, a real-time Java ORB, and with OpenORB, a desktop Java ORB. We measured throughput, latency, and jitter of method invocations for both ORBs configured with and without OpenCCM. We also measured the additional memory requirement introduced by the CCM implementation. We concluded that OpenCCM adds some overhead to both Java ORBs, affecting OpenORB's performance more than ZEN's. More development of the CCM may be necessary to bring its advantages to high-performance DRE systems.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Client/server, Distributed applications*; C.4 [Performance of Systems]: Measurement techniques

Keywords

Java-based middleware, benchmarking, CORBA, CCM

*This work was supported by Boeing DARPA contract Z20402 and AFOSR grant F49620-00-1-0330.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC '05, March 13-17, 2005 Santa Fe, New Mexico, USA.
Copyright 2005 ACM 1-58113-964-0/05/0003 ...\$5.00.

1. INTRODUCTION

Distributed, real-time, and embedded (DRE) systems are difficult to develop, maintain, and evolve, because they impose stringent requirements on performance and resource utilization. Applications for real-time systems must meet constraints for throughput, latency, and predictability, while applications deployed on embedded systems face memory footprint limitations.

Real-time CORBA [1] middleware can facilitate the development of DRE applications by abstracting low-level details of the operating systems away from the application and into the middleware. Real-time CORBA middleware provides mechanisms for controlling 1) non-functional aspects, such as resource allocation [2]; 2) real-time aspects, such as timing and fault-tolerance [3]; 3) flexible configuration of underlying protocols [4, 5]; and 4) platform-independence [6]. However, these aspects are often interwoven among various object implementations, resulting in code that can be hard to debug and reuse.

Component middleware technologies overcome some of these limitations of conventional middleware by providing higher-level mechanisms to compose and configure complex DRE applications, by configuring the non-functional aspects of DRE applications separately from the business code, thus enabling better reuse of software. Component models provide high-level abstractions for connecting data sources to sinks, and for configuring software component implementations on the various boards in the distributed system.

The CORBA Component Model (CCM) [7] is a specification developed by the Object Management Group (OMG) to enable component technology for CORBA-based middleware. The CCM is the first vendor-neutral open standard for *Distributed Component Computing*, supporting various programming languages, operating systems, and CORBA products seamlessly. The CCM abstracts details of the middleware, encapsulates business code into *components*, and provides mechanisms for the assembly of components and for the configuration of non-functional concerns such as fault-tolerance, security, and predictability.

While CCM facilitates development of DRE middleware, it also adds an extra layer of overhead that could affect performance. However, previous experiments showed that QoS-enabled component middleware implementations can offer performance and predictability similar to that of real-time CORBA middleware [8]. Specifically, the real-time performance of an example DRE system implemented in the The ACE ORB (TAO) [3], Real-time CORBA middleware written in C++, was similar to the performance of the same

system implemented with the Component-Integrated ACE ORB (CIAO) [9], a QoS-enabled implementation of CORBA 3.x CCM specification implemented atop TAO.

Since our research focuses on Java-based Real-time CORBA middleware, we tested whether OpenCCM [10], the first openly available Java implementation of the CCM specification, could be deployed on top of existing Java-based CORBA ORBs without unduly degrading performance. Specifically, we tested the effect of the overhead of CCM on the performance of the following two Java-based ORBs developed in previous research:

- ZEN [6], an open-source Java ORB with a modular, small-footprint design and real-time performance suitable for use in DRE systems, and
- OpenORB [11], an open-source Java ORB designed for desktop and enterprise systems, designed to be a foundation for component and component framework technology [12].

Since ZEN is being developed for DRE applications with real-time performance and memory constraints, it is particularly important to see if the CCM technology available can be used with ZEN without unduly degrading ZEN's size or performance. However, it is also interesting to compare how well a desktop Java ORB performs using the CCM. We chose OpenORB as a representative desktop Java ORB, partly since it is full-featured and well-documented, but especially since it was designed specifically to work with components.

The remainder of this paper is organized as follows. Section 2 details the advantages of using component technology for building DRE applications; Section 3 presents and interprets the results of the empirical measurements; Section 4 describes related work; and Section 5 presents concluding remarks.

2. ADVANTAGES USING THE CCM WITH CORBA

Distributed object middleware, such as CORBA, provides mechanisms to develop the functional parts of an application. However, objects in CORBA middleware systems have to be engineered from scratch. With no standard way to deploy object implementations, designers use ad hoc strategies to instantiate all objects in a system. Ad hoc implementations create tightly coupled objects with multiple dependencies, resulting in complicated, platform-dependent systems that are hard to debug, maintain, and extend.

Using *component architecture*, systems can be built through assembly instead of through conventional engineering. The CCM abstracts away the low-level details of the underlying middleware and presents the user with a "building-blocks" model for developing large-scale distributed applications. Objects are encapsulated in components with virtual boundaries. Components expose well-defined interfaces called *facets* that may be used by other components' *receptacles*. They can also behave as event sources or sinks. The events could be either time-triggered (by a Timer component) or event-triggered (by other components). The components are instantiated and executed within a well-defined environment called a container in which the component can run. The containers locate and connect components via their interfaces. The Component Implementation Framework generates a significant portion of the code and eases the job of the application

developer. The developer only has to define the relationships between the components and implement the business code for each component.

The CCM provides the following advantages over traditional CORBA middleware for large-scale DRE systems:

- **Assembly and deployment support:** The component model provides high-level mechanisms for the assembly of components. It provides support for managing commonly used CORBA services, such as transaction, security, persistent state, and event notification. For example, it greatly simplifies the connecting of event sources to event sinks. CCM also provides deployment support by examining the interdependencies of components and their interconnections. Capturing the component dependency information ensures that the components are instantiated in the correct order and automatically, providing a scalable model for developing complex DRE applications.
- **Separation of concerns and reuse:** In conventional CORBA middleware, non-functional aspects such as fault-tolerance, security, and predictability are specified along with the functional objects using interfaces. This method of configuration of aspects leads to intricate dependencies between objects and makes it difficult to reuse the objects with different configurations. In the component model, the components are enclosed in boundaries. Meta-data is used to define component dependencies and configure the non-functional requirements. This model makes it easier to reuse components and specify their runtime configuration. The component meta-data can also be used to configure advanced features such as real-time behavior [8].

In summary, the CCM extends the CORBA object model by defining a component architecture that provides support for assembly, deployment, and integration of common CORBA services, facilitating separation of concerns and reuse. The reuse of components is a valuable advantage, because the same component implementation can be configured with different non-functional aspects for different application requirements. The component architecture also makes it easier to maintain components over time, as compared to object-oriented middleware. Thus the CCM makes it easier, cheaper, and faster to build, maintain, and extend large-scale, distributed applications.

3. EMPIRICAL MEASUREMENTS

We empirically evaluated the overhead from using OpenCCM with two Java ORBs, ZEN and OpenORB, to see whether OpenCCM differentially affects the performance and size of different ORBs with different design goals. Since ZEN was designed and optimized for DRE systems, we expected ZEN's performance to be better and size smaller than OpenORB's. We also hypothesized that ZEN could incur less overhead from OpenCCM because of ZEN's optimizations at the algorithm level.

3.1 Design

This experiment tested two factors: ORB (ZEN and OpenORB) and OpenCCM (with and without). There were therefore four different configurations tested in this 2X2 factorial experiment:

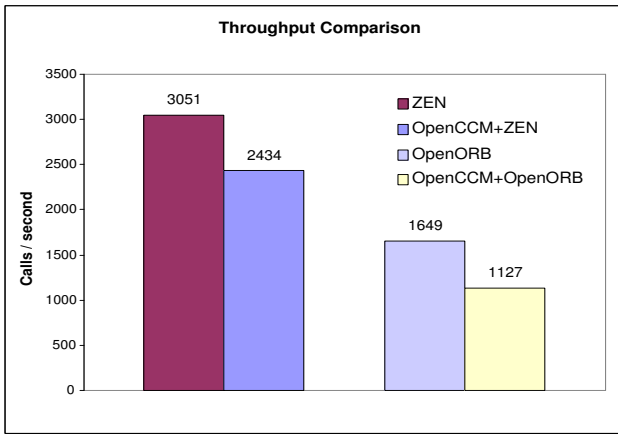


Figure 1: Throughput Comparison

1. ZEN,
2. ZEN+OpenCCM,
3. OpenORB, and
4. OpenORB+OpenCCM

Method call. In this experiment, a client, which consists of an object (using the ORB alone) or component (using OpenCCM with the ORB), makes method calls to a server. The method call is a `void ping()` invoked by a client on the server. The ORB implementation of this experiment consists of a simple server and client that execute and invoke the `ping` operation. The OpenCCM implementation consists of two components, one that *provides* the target interface and another that *uses* the interface. The method calls have a zero payload, so that only the time taken to make the call is considered.

Location. The method calls are made in locally located servers; i.e., both the client and server reside on the same machine. Local location was preferred over remote location, in order to eliminate any delays and variations caused by the network.

Sample size. 500 sets of invocations were performed for each of the configurations. In each set, the first 1000 invocations were not measured, in order to eliminate variation due to cold start. The next 1000 invocations were measured and analyzed. The large number of invocations renders any constant, small overhead in taking the timing measurement insignificant.

Testbed. The testbed was composed of a two-CPU 1.6GHz Pentium 4 computer running Red Hat Linux version 2.4.20-8smp. The test programs were developed using OpenCCM version 0.7, ZEN version 1.0, and OpenORB version 1.3.0.

3.2 Results: Throughput

Adding an extra level of abstraction, such as the CCM, to a software system may reduce the overall throughput of the system. Throughput was measured in number of calls per second on each of the four configurations.

Figure 1 shows that the throughput decreases as a result of using OpenCCM, which incurs the overhead of supporting CCM mechanisms and obtaining references to objects. Compared to the ORB, an operation invocation on OpenCCM incurs additional method calls. This overhead appears to

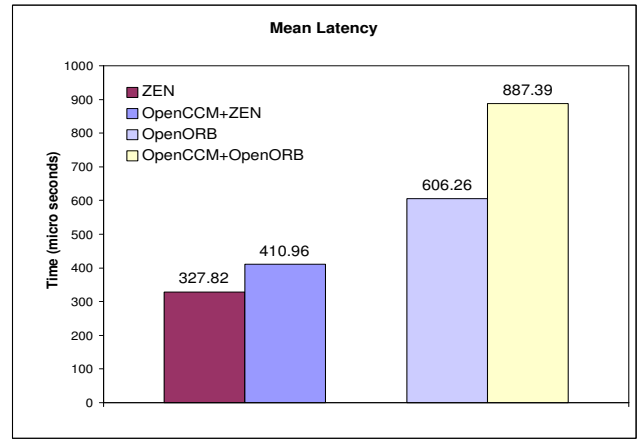


Figure 2: Latency Comparison

be constant for both ORBs, in terms of absolute number of calls: using OpenCCM reduced throughput by ≈ 500 -600 calls/second. There is also an obvious effect of ORB: ZEN has higher throughput than OpenORB.

3.3 Results: Latency

Using OpenCCM may increase the typical latency, or round-trip time of the `ping` method invocation. Figure 2 shows that ZEN's mean latency is significantly less than OpenORB's. In addition, OpenCCM increases the median latency for both ORBs, but increases OpenORB's latency more than ZEN's (46% vs. 25%).

3.4 Results: Jitter

Using OpenCCM may increase the jitter of a method invocation, particularly since OpenCCM is not designed for real-time performance. Two measurements were taken to provide an estimate of jitter, standard deviation and the maximum. As shown in Figure 3, ZEN has less jitter than OpenORB in both estimates. This is expected, since ZEN was designed to support real-time performance, whereas OpenORB was not. In addition, OpenCCM increases the jitter for both ORBs, but only in proportion to the mean latency. As measured by the standard deviation, OpenCCM appears to add a constant proportion to the jitter of both ORBs, about 25%; however, since the mean latencies increased by 25% or more, the jitter's increase is only proportional to overall slowing of the method call.

OpenCCM also appears to increase the maximum, but more so for OpenORB than for ZEN. This result shows that the algorithm-level optimizations in ZEN improve its predictability significantly for the worst-case. ZEN's high-level design eliminates sources of unpredictability at the ORB algorithm level [13]. Using the CCM exposes more potential sources of jitter because using the CCM requires use of more of the ORB features.

3.5 Results: Footprint

ZEN, OpenORB, and OpenCCM packages support numerous services and contain many classes which may not be required by every application. Hence, the on-disk memory footprint of the middleware does not provide an accurate indication of the amount of memory required for a particular application to execute. We therefore measured the size (and

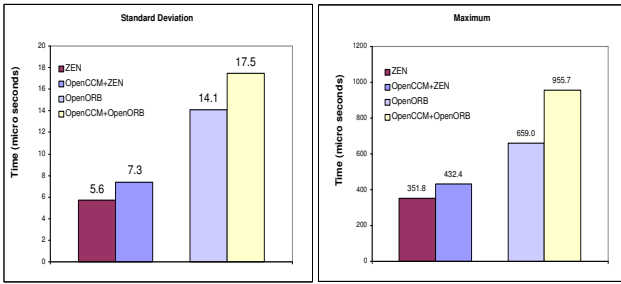


Figure 3: Jitter Comparison

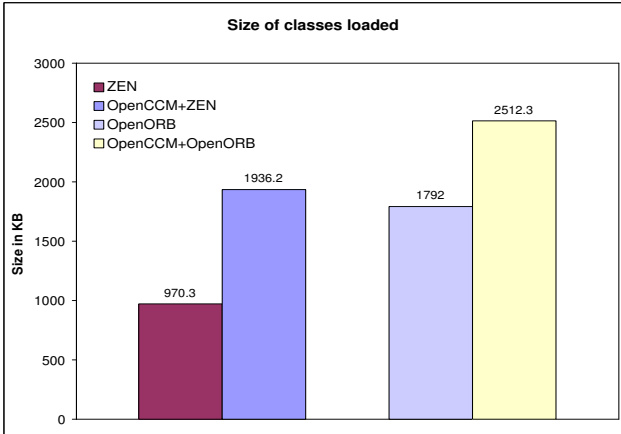


Figure 4: Footprint Comparison

number) of classes actually loaded into the JVM during runtime for each configuration. These measurements were obtained using the `jstat` monitoring tool provided with Sun’s JDK 1.5 to measure the class size used by each configuration (but not the size of the objects created at runtime).

Figure 4 shows the size of classes loaded during runtime for the four different configurations. (The results were equivalent when measured in number of classes.) As expected, ZEN has a much smaller footprint than OpenORB, since ZEN was designed to be highly modular to minimize footprint. In addition, OpenCCM adds considerably to the footprint of both ORBs, but more so to ZEN’s footprint. The additional classes can be accounted for by the various services that CCM inherently has to support, and by its container architecture.

3.6 Discussion

Using OpenCCM with ZEN causes a moderate performance overhead for throughput, latency, and jitter. This overhead may be a concern for DRE application developers. Unlike CIAO [14], OpenCCM is not a real-time implementation of the component model; it does not provide mechanisms to configure real-time aspects end-to-end. Our results indicate that further research and development of real-time configurability may be required before CCM technology can be widely used in Java-based DRE systems. A Real-time Java [15] implementation of the CCM could provide better predictability and resource management that is necessary for DRE systems.

OpenCCM requires more primary memory than CORBA-

based ORBs, since it must provide a CCM environment consisting of containers and component servers. This additional memory may be significant for small applications, but may be less of a concern for large-scale DRE applications involving many components. To reduce the static on-disc memory used by OpenCCM, the number of runtime classes can be reduced by better practices in modularization of code to reduce the dependencies between classes. Unused libraries, classes, and class elements may be removed by mechanisms such as code modularization and aspect-oriented programming [16].

4. RELATED WORK

A significant amount of work has already been done in the broad area of CORBA-based middleware benchmarking [17, 18]. Previous work has been done regarding optimizing CORBA-based middleware for DRE systems. [19, 20] discuss mechanisms to optimise CORBA middleware for low latency and footprint. [21] discusses techniques for minimizing memory footprint for DRE systems.

The CCM is an emerging standard and there have been recent efforts in empirically evaluating component middleware. CCMPerf [22, 23] is a middleware benchmarking suite for CIAO [24], a C++ implementation of CCM.

[8] presents empirical comparisons of configuring real-time aspects in TAO [3], a real-time ORB, and CIAO. This paper concludes that their component middleware implementations offer a performance that is comparable to that of TAO. Philippe Merle et. al. [10] are involved in benchmarking OpenCCM on various Java ORBs.

Other CORBA-based Java implementations have been developed, such as JacORB [25] and ORBacus [26]. Other available component technologies are Enterprise Java CORBA Component Model (EJCCM) [27], the MICO CORBA Component Project (MicoCCM) [28], and QoS enabled Distributed Components (Qedo) [29].

5. CONCLUSION

The CORBA Component Model offers significant advantages over CORBA alone, such as ease of configuration and development and efficient code reuse. CIAO, a C++ implementation of CCM built on top of TAO, did not add significant overhead to TAO’s performance. We empirically evaluated a Java implementation of the CCM, OpenCCM, and compared its performance on ZEN and OpenORB. The results indicated that OpenCCM reduces the throughput and increases latencies and jitter somewhat, for both ZEN and OpenORB. The increase in jitter may be unacceptable for some DRE systems. Since the jitter appears to be proportional to the mean performance, improving the typical performance should similarly improve jitter. OpenCCM also significantly increases the memory footprint for both ORBs, but especially for ZEN, since ZEN’s runtime class size is smaller due to its modular design. OpenCCM uses more of the ORB features than ZEN would normally require, thus neutralizing ZEN’s initial footprint advantage. This effect on footprint could be prohibitive for severely memory-constrained embedded devices but could be of less concern for large-scale applications. Research to develop a real-time specification for the CCM and to modularize its features could overcome these limitations.

6. ACKNOWLEDGMENTS

The authors would like to thank Susan Anderson Klefstad for her help with presentation and revision.

7. REFERENCES

- [1] Object Management Group. *Real-time CORBA Specification*, OMG Document formal/02-08-02 edition, August 2002.
- [2] Raymond Klefstad, Douglas C. Schmidt, and Carlos O’Ryan. The Design of a Real-time CORBA ORB using Real-time Java. In *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing*. IEEE, April 2002.
- [3] Douglas C. Schmidt, Bala Natarajan, Aniruddha Gokhale, Nanbor Wang, and Chris Gill. TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems. *IEEE Distributed Systems Online*, 3(2), February 2002.
- [4] Carlos O’Ryan, Fred Kuhns, Douglas C. Schmidt, Ossama Othman, and Jeff Parsons. The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware. In *Proceedings of the Middleware 2000 Conference*. ACM/IFIP, April 2000.
- [5] Douglas C. Schmidt Raymond Klefstad, Sumita Rao. Design and performance of a dynamically configurable, messaging protocols framework for real-time corba. In *Proceedings of the Distributed Object and Component-based Software Systems part of the Software Technology Track at the 36th Annual Hawaii International Conference on System Sciences*, pages 320, 10 pages, Big Island of Hawaii, January 6-9 2003.
- [6] Center for Distributed Object Computing. The ZEN ORB. www.zen.uci.edu, University of California at Irvine.
- [7] Object Management Group. *CORBA Components*, omg document formal/01-03-01 edition, 2002.
- [8] Nanbor Wang, Christopher Gill, Venkita Subramonian, and Douglas C. Schmidt. Configuring real-time aspects in component middleware. In *Distributed Objects and Applications*, 2004.
- [9] Institute for Software Integrated Systems. Component-Integrated ACE ORB (CIAO). www.dre.vanderbilt.edu/CIAO/, Vanderbilt University.
- [10] ObjectWeb. *OpenCCM: The Open CORBA Components Platform*. Presented at the Third ObjectWeb Conference on 20th and 21th November 2003.
- [11] The community openorb project. <http://openorb.sourceforge.net>, 2002.
- [12] <http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/meta.php>.
- [13] Arvind Krishna, Douglas C. Schmidt, Krishna Raman, and Raymond Klefstad. Optimizing the orb core to enhance real-time corba predictability. In *Distributed Objects and Applications (DOA) 2003*. DOA, 2003.
- [14] Nanbor Wang and Christopher Gill. Improving Real-Time System Configuration via a QoS-aware CORBA Component Model. In *Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2003*, Honolulu, HI, January 2003. HICSS.
- [15] The Real-Time for Java Expert Group. The Real-Time Specification for Java. <http://www.rtfj.org>.
- [16] F. Hunleth and R. Cytron. Footprint and feature management using aspect-oriented programming techniques. In *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems*, pages 38–45, 2002.
- [17] P. Tuma and A. Buble. Open corba benchmarking. In *Tuma P., Buble A.: Open CORBA Benchmarking, Proceedings of SPECTS 2001, USA*, 2001.
- [18] Adam Buble and Petr Tuma. On benchmarking object-oriented communication middleware. In *Proceedings of the Week of Doctoral Students (WDS), Faculty of Mathematics and Physics, Charles University, Prague*, 2000.
- [19] Aniruddha S. Gokhale and Douglas C. Schmidt. Measuring and optimizing CORBA latency and scalability over high-speed networks. *IEEE Transactions on Computers*, 47(4):391–413, 1998.
- [20] A. Gokhale and D. Schmidt. Optimizing a corba iiop protocol engine for minimal footprint multimedia systems. *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, 1998.
- [21] Mark Panahi, Trevor Harmon, and Raymond Klefstad. Adaptive techniques for minimizing middleware memory footprint for distributed, real-time, embedded systems. *IEEE Computer Communications Workshop (CCW)*, 2003.
- [22] Arvind S. Krishna, Nanbor Wang, Balachandran Natarajan, Anniruddha Gokhale, Douglas C. Schmidt, and Gautam Thaker. CCMPerf: A Benchmarking Tool for CORBA Component Model Implementations. In *Proceedings of the 10th Real-time Technology and Application Symposium (RTAS ’04)*, Toronto, CA, May 2004. IEEE.
- [23] A. S. Krishna, J. Balasubramanian, A. Gokhale, D. C. Schmidt, D. Sevilla, and G. Thaker. Empirically evaluating corba component model implementations. In *Proceedings of the OOPSLA 2003 Workshop on Middleware Benchmarking*, 2003.
- [24] N. Wang, D. Schmidt, and D. Levine. Optimizing the corba component model for high-performance and real-time applications. In ‘*Work-in-Progress*’ session at the *Middleware 2000 Conference, ACM/IFIP.*, April 2000.
- [25] Software Engineering, Freie Universitat Berlin Systems Software Group, and Xtradyne Technologies AG. *JacORB*.
- [26] IONA. Orbacus. www.orbacus.com.
- [27] Enterprise java corba component model. <http://www.cpi.com/ejccm/>.
- [28] MICO. The mico corba component project. <http://www.fpx.de/MicoCCM/>.
- [29] Tom Ritter, Marc Born, Thomas Unterschutz, and Torben Weis. A qos metamodel and its realization in a corba component infrastructure. In *HICSS, Hawaii*, 2003.