Using Khazana to Support Distributed Application Development *

Sai Susarla Anand Ranganathan Yury Izrailevsky John Carter

UU-CS-TR-99-008

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

Abstract

One of the most important services required by most distributed applications is some form of shared data management, e.g., a directory service manages shared directory entries while groupware manages shared documents. Each such application currently must implement its own data management mechanisms, because existing runtime systems are not flexible enough to support all distributed applications efficiently. For example, groupware can be efficiently supported by a distributed object system, while a distributed database would prefer a more low-level storage abstraction. The goal of Khazana is to provide programmer's with configurable components that support the data management services required by a wide variety of distributed applications, including: consistent caching, automated replication and migration of data, persistence, access control, and fault tolerance. It does so via a carefully designed set of interfaces that support a hierarchy of data abstractions, ranging from flat data to C++/Java objects, and that give programmers a great deal of control over how their data is managed. To demonstrate the effectiveness of our design, we report on our experience porting three applications to Khazana: a distributed file system, a distributed directory service, and a shared whiteboard.

*This research was supported in part by the Defense Advanced Research Projects Agency, monitored by the Department of the Army under contract number DABT63–94–C–0058, and the Air Force Research Laboratory, Rome Research Site, USAF, under agreement number F30602–96–2–0269. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Email: {sai,anand,izrailev,retrac}@cs.utah.edu
Khazana URL: http://www.cs.utah.edu/projects/khazana

1 Introduction

Distributed systems involve complicated applications with complex interactions between disparate components. The environment in which these applications operate introduces additional challenges in terms of fault tolerance and security. As a result, researchers have developed a wide variety of systems to ease the chore of building distributed applications. The earliest distributed systems provided support for interprocess communication via message passing [8, 27] or remote procedure calls [3], but provided little support for transparent distribution of data and execution or for fault tolerance. More sophisticated systems have provided such support via a variety of basic abstractions, including *distributed files* [5, 12, 23, 9, 28], *distributed objects* [20, 21, 19], and *distributed shared memory* (DSM) [1, 7, 22, 24]. Each of these models is useful for certain types of applications. For example, systems like Petal [23] that support a flat persistent storage abstraction are ideal for supporting distributed file systems and distributed directory services, systems with fairly simple persistent coarse-grained data structures. In contrast, distributed object systems such as CORBA [19] are useful for hiding the complexities of client-server systems, while distributed shared memory systems like Treadmarks [1] are useful for running shared memory codes on top of distributed systems.

Currently, distributed applications must implement their own data management mechanisms, because no existing runtime system can support the very different needs of each application efficiently. This approach has the advantage of allowing each system to optimize its data management mechanisms to suit its specific needs. However, it requires a great deal of redundant programmer effort to develop and maintain each such set of ad hoc mechanisms. It also makes it difficult to share state *between* applications or reuse code devel-