PATTERN-ORIENTED SOFTWARE ARCHITECTURE

Patterns for Resource Management

Volume 3

Michael Kircher,
Siemens AG Corporate Technology, Munich, Germany

Prashant Jain,
IBM Research Labs, Delhi, India

JOHN WILEY & SONS, LTD

Resource Acquisition Patterns

The *Lookup* (21) pattern describes how to find and access resources, whether local or distributed, by using a lookup service as a mediating instance.

The *Lazy Acquisition* (38) pattern defers resource acquisitions to the latest possible time during system execution in order to optimize resource use.

The *Eager Acquisition* (53) pattern describes how run-time acquisition of resources can be made predictable and fast by eagerly acquiring and initializing resources before their actual use.

The *Partial Acquisition* (66) pattern describes how to optimize resource management by breaking up acquisition of a resource into multiple stages. Each stage acquires part of the resource, dependent upon system constraints such as available memory and the availability of other resources.

Resource Lifecycle Patterns

The *Caching* (83) pattern describes how to avoid expensive re-acquisition of resources by not releasing the resources immediately after their use. The resources retain their identity, are kept in some fast-access storage, and are re-used to avoid having to acquire them again.

The *Pooling* (97) pattern describes how expensive acquisition and release of resources can be avoided by recycling resources that are no longer needed. Once the resources are recycled and pooled, they lose their identity and state.

The *Coordinator* (111) pattern describes how to maintain system consistency by coordinating the completion of tasks involving multiple participants, each of which can include a resource, a resource user and a resource provider. The pattern presents a solution such that in a task involving multiple participants, either the work done by all of the participants is completed or none are. This ensures that the system always stays in a consistent state.

The *Resource Lifecycle Manager* (128) pattern decouples the management of the lifecycle of resources from their use by introducing a separate Resource Lifecycle Manager, whose sole

responsibility is to manage and maintain the resources of an application.

Resource Release Patterns

The *Leasing* (149) pattern simplifies resource release by associating time-based leases with resources when they are acquired. The resources are automatically released when the leases expire and are not renewed.

The *Evictor* (168) pattern describes how and when to release resources to optimize resource management. The pattern allows different strategies to be configured to determine automatically which resources should be released, as well as when those resources should be released.

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

Wiley Series in Software Design Patterns

The WILEY SERIES IN SOFTWARE DESIGN PATTERNS is designed to meet the needs of today's software architects, developers, programmers and managers interested in design patterns. Frank Buschmann, the Series Editor, as well as authors, shepherds and reviewers will work collaboratively within the patterns community to strive for high-quality, highly researched, thoroughly validated, classic works which document accepted and acknowledged design experience. Priority will be given to those titles that catalog software patterns and pattern languages with a practical, applied approach in domains such as:

- Distributed systems
- Real time systems
- Databases
- Business information systems
- Telecommunications
- Organizations
- Concurrency
- Networking

Books in the series will also cover conceptual areas of how to apply patterns, pattern language developments and architectural/component-based approaches to pattern-led software development.

TITLES PUBLISHED

- PATTERN-ORIENTED SOFTWARE ARCHITECTURE, Volume 1

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal

0 471 95869 7	476pp	1996	Hardback

- PATTERN-ORIENTED SOFTWARE ARCHITECTURE, Volume 2

Douglas Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann

0 471 60695 2	636pp	2000	Hardback

- A PATTERN APPROACH TO INTERACTION DESIGN

Jan Borchers

0 471 49828 9	250pp	2001	Hardback

- SERVER COMPONENT PATTERNS

Markus Völter, Alexander Shmid, Eberhard Wolff

0 471 84319 5	462pp	2002	Hardback

- ARCHITECTING ENTERPRISE SOLUTIONS

Paul Dyson, Andy Longshaw

0 470 85612 2	384pp	2004	Hardback

- PATTERN-ORIENTED SOFTWARE ARCHITECTURE, Volume 3

Michael Kircher, Prashant Jain

0 470 84525 2	312pp	2004	Hardback

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

Patterns for Resource Management

Volume 3

Michael Kircher,
Siemens AG Corporate Technology, Munich, Germany

Prashant Jain,
IBM Research Labs, Delhi, India

JOHN WILEY & SONS, LTD

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England
Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wileyeurope.com or www.wiley.com

Other Wiley Editorial Offices
John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA
Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA
Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany
John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia
John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809
John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library
ISBN 0-470-84525-2

Typeset in Bookman Light 10/13 point by WordMongers Ltd, Treen, Cornwall TR19 6LG, England
Printed and bound in Great Britain by Biddles Ltd, King's Lynn, England
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

For Christa, my parents, and grandparents

Michael Kircher

For Ruchi, Aanya, and my parents

Prashant Jain

Table of Contents

Foreword by Frank Buschmann

I never thought this would happen: a new volume of the POSA series is published and nobody from the original 'Party of Five' is on the author list. Yet I am very proud that I was wrong! I am proud because this book shows that a new generation of smart software engineers has grown up and is actively contributing their experience to the patterns community. I am proud because the book demonstrates that the pattern idea is still thriving. And I am proud, of course, because our original POSA vision from 1996 is still inspiring patterns authors to build on our material and enhance, mature, and evolve it.

The topic chosen for this POSA volume is key for the success of almost every software system: resource management. This may appear obvious, but only a decade ago many developers thought resource management was of concern only in the embedded systems domain. In the desktop and enterprise spheres only a few developers paid attention to the topic. Why care about resources like memory, for example—if you run out of it, you can simply upgrade your machine. I must admit that in the earlier days of my career I had a similar perspective: resources were just 'there' and unlimited. How wrong I was! Fortunately I shot myself in the foot and quickly learned my lesson.

Today few developers ignore the importance of resource management. With the advent of component-based systems, application servers, and the growing complexity and size of the applications running on our computers, it is realized that managing resources carefully can make a significant difference to the quality of a software system. Resources such as memory, CPU power, threads, connections, and so on are limited even on the biggest servers. Yet these servers are expected to provide a high quality of service to all their users, even if they are accessed by zillions of users simultaneously. This conflict is only resolvable if the server's resources are managed explicitly and with care. However, the term 'resource' is not limited to low-level

things like memory or connections. In today's networked computing world, resources can also include components and services that are used remotely by client applications. Multiple client applications often compete for access to these components and services. Ensuring that all clients are served sufficiently well is the subject of appropriate resource management.

However, acknowledging the importance of resource management and doing it well are two different things. Effective resource management is both difficult and challenging. If you do it well, your applications will be efficient, stable, scalable, predictable, and accessible. If you do it badly, your applications will at best provide very limited operational quality. At worst they will just fail—it is as simple as that. Allowing a container to do resource management is not always a solution, because many software systems cannot afford such infrastructures. Even if using a container is a feasible option, you need to understand how its resource management works to be able to build high quality systems. Many applications using containers fail simply because of lack of understanding.

How can you acquire this understanding? What sources can you mine for the challenges in resource management, the solutions to master these challenges, and the do's and don't associated with the solutions? One such source is the book you are holding, the third volume of the POSA series. It presents experiences and solutions in resource management gained over many years. All such experiences and solutions have proved their quality in active duty in countless well-known applications and middleware. Capturing these experiences and solutions as patterns makes them accessible to every software developer. Novices can learn about the fundamental concerns and solutions in resource management, while experts can cross-check and evaluate alternative solutions and read about details of a particular solution. I am unaware of any other literature on resource management that is equally comprehensive.

As said at the beginning: I am proud of this book. If you read it, you will know why.

Frank Buschmann
Siemens AG, Corporate Technology

Foreword by Steve Vinoski

If you've been a software developer for any appreciable length of time, you have almost certainly experienced what I call the 'stroll down computer memory lane'. This event occurs regularly when developers get together in a social setting, such as having lunch together. It starts out innocently enough, with one of the developers describing a recent run-in with an especially difficult problem. Not to be outdone, another developer chimes in with a story detailing the conquest of an even worse problem. From there, each new story attempts to outdo the last, until only the old-timers are left speaking about primitive machines that had to be programmed with punch cards or toggle switches and had only a few bytes of RAM. I am waiting for the day that during a stroll down computer memory lane, someone tries to convince me that back when he or she started programming, there were only zeros to program with, and no ones!

Developers are able to compare stories in the manner described above because programming inherently requires many trade-offs. Applications have to share computing resources with other applications. Memory space and disk storage are not infinite. CPUs can process only a certain number of instructions per second. Disk and device I/O can take a relatively long time. Establishing database connections and network connections can be expensive in terms of time and resources. Throughout the history of electronic computing, tremendous advances have been made in hardware, operating systems, middleware, and applications. Unfortunately, despite these advances, programming remains very much an art of choosing the right trade-offs to maximize the effectiveness of the overall computing solution.

All applications manage resources of some kind, such as memory, connections to networks or databases, or threads. However, there is a marked difference between applications that are written and tuned to manage resources well and those that aren't. Ignoring resource

management is not a problem for simple applications that run only occasionally for brief periods of time, such as basic command-line utilities or configuration GUIs. However, a lack of focus on resource management is a recipe for failure when developing long-running applications that must be robust, high performance, and scalable, such as Web servers, application servers, and notification systems.

The patterns in this book are about resource management. Generally, resources can be acquired, managed, and released, and the patterns presented here specifically target those three areas. These are the primary areas that can have a profound influence on the overall performance, size, scalability, and durability of a running application. For example, many applications acquire memory from the heap. For an application like a Web or application server, every individual heap memory allocation made in the request-handling code path decreases the server's performance and scalability, due to the cost of invoking the heap manager and the cost of acquiring and releasing mutexes needed to protect the heap for concurrent access. Such an application might instead apply a pattern such as Partial Acquisition to eagerly acquire as many resources as possible before entering the request code path, and apply the Caching or Pooling patterns to keep the resources around for the next request, rather than releasing them and later reacquiring them. Even experienced developers can be pleasantly surprised by the degree to which applying the right combination of resource management patterns can positively impact the performance and scalability of an application.

This book continues with the POSA tradition of emphasizing practical solutions. I especially like the fact that each pattern in this book includes a fairly detailed section describing implementation issues, as well as a section that provides an extensive list of known applications of the pattern. In addition, there are two chapters describing detailed case studies that show not only how the patterns might be applied to real-world applications, but also how the patterns relate to each other within such applications. Patterns are, after all, descriptions of approaches that have been proven to work in real applications, and the fact that the authors have so effectively tied them back to their sources and influences ensures that the patterns maintain that important connection with the real world.

Software patterns generally help us decide what trade-offs to make in our architectures and designs, and where to make them. Programming is, after all, a human endeavor, and patterns raise the levels of abstraction and communication that teams use to socialize their architectures and designs. As a long-time middleware architect, designer, and implementer, I have at one time or another successfully applied each of the patterns that Prashant and Michael present here. Unfortunately, I did so before the approaches were captured as patterns, which means that my teammates and I spent many hours devising our own variations, implementing them, measuring them to determine their trade-offs, and refining and tuning them based on our measurements. Now, because Michael and Prashant have clearly and comprehensively documented these approaches as a pattern language, you and your teammates can more easily discuss them and understand what they're good for, determine the best circumstances under which to apply them, and understand the forces you need to concern yourselves with when implementing them.

Armed with the knowledge of these resource management patterns and good performance measurement tools, you'll be pleasantly surprised at how quickly and easily you can take an application with only mediocre performance and scalability and turn it into something that races along instead. With these kinds of skills, your stories at the next stroll down computer memory lane will impress even the old-timers.

Steve Vinoski
Chief Engineer, Product Innovation
IONA Technologies

About This Book

This is a book about patterns for resource management in software systems. The patterns provide solutions for problems that are typically encountered by software architects and developers when trying to provide an effective and efficient means of managing resources in a software system. Efficient management of resources is critical in the execution of any kind of software. From embedded software in a mobile device to software in a large enterprise server, it is important that resources, such as memory, threading, files, or network connections, are managed efficiently to allow the systems to function properly and effectively.

The first volume of the Pattern Oriented Software Architecture (POSA) series [POSA1] introduced a broad-spectrum of general-purpose patterns in software design and architecture. The second volume of the series [POSA2] narrowed the focus to fundamental patterns for building sophisticated concurrent and networked software systems and applications. This volume uses patterns to present techniques for implementing effective resource management in a system.

The patterns in this book are covered in detail and make use of several examples. As with the previous POSA volumes, the book provides directions to the readers on how to implement the patterns presented. Additionally, the volume presents a thorough introduction to resource management, and two case studies, in which the patterns are applied to two different domains. The patterns presented are independent of any implementation technique, such as .NET, Java or C++, even though the examples are given in Java and C++. The patterns are grouped by different areas of resource management, and hence address the complete lifecycle of resources: resource acquisition, resource lifecycle and resource release.

The patterns in the book provide an extensive coverage of the sphere of resource management. We began documenting these patterns

several years ago based on our experiences of building many different software systems. Most of the patterns have been presented or workshopped at leading conferences. However, what we felt was missing was an effort to pull the patterns together in the form of a pattern language and present it in such a way that the pattern language can be applied to multiple domains.

The scope of resource management is vast. The challenges that are faced by system designers and developers dealing with the management of resources are constantly changing as new technologies emerge. We anticipate that additional patterns in resource management will be discovered and documented with time. The *Concluding Remarks* chapter of this book talks about what lies ahead in the effort to evolve the resource management pattern language.

Intended Audience

This book is for all software architects, designers, and developers. They can use the patterns described in the book to solve the challenges in resource management that every typical software system faces.

The book will also be useful to students of computer science, as it can provide them with a broad overview of the available best practices in resource management.

Structure of the Book

The book is divided into two parts. The first part provides an introduction to the topic of resource management and the patterns in resource management. The patterns presented in the first part have been grouped into three chapters, *Resource Acquisition*, *Resource Lifecycle*, and *Resource Release*, to correspond with the typical lifecycle of resources. The second part of the book applies the patterns to two case studies.

While the first part of the book looks at resource management from a problem domain perspective, the second part of the book does so from an application domain perspective. The patterns in this book are not isolated. In fact, throughout our coverage of resource management

patterns, we make extensive references to other related and relevant patterns. For each such pattern, we have included thumbnail descriptions in the *Referenced Patterns* chapter.

The book contains many examples of the use of the patterns. While the patterns use individual examples, each case study chapter uses a single example in a particular domain to tie all the patterns together. This acts as a running example by presenting problems in that particular domain and uses the individual patterns together to address the problems presented. This approach allows us to prove the broad applicability of the patterns while still showing how they connect together.

The first chapter, *Introduction*, formally introduces the topic of resource management in software systems and defines its scope. The chapter describes why managing resources in software systems effectively and efficiently is challenging. It also introduces patterns and shows how they can be used to address the challenges of resource management.

Chapter 2, *Resource Acquisition*, describes patterns that address the forces affecting the acquisition of resources. Resources must be acquired before they can be used. However, acquisition of resources should not degrade system performance, nor should it produce any bottlenecks. For large resources or resources that become available only in parts, the adaptation of the resource acquisition strategy is essential.

Chapter 3, *Resource Lifecycle*, describes patterns that resolve forces affecting the lifecycle of resources. Resources can have very different lifecycles. For example, some resources are both heavily and frequently used, while others may only be used once. When a resource is no longer needed, it can be released. However, determining when to release a resource is not trivial. Explicit control of the lifecycle of a resource can be tedious and error-prone. To address this problem, automated resource management techniques are needed. In addition, in certain architectures such as distributed systems, multiple resources have to work together to achieve a higher common goal. As every resource can potentially be managed by its own thread of control, the collaboration and integration of multiple resources needs to be coordinated.

Chapter 4, *Resource Release*, describes patterns that deal with the efficient and effective release of resources. Resources no longer needed should be returned to the resource environment to optimize system performance and allow other users to acquire them. However, if a released resource needs to be used again by the same resource user, the resource must be re-acquired, impacting performance. The challenge is to find the right balance and to determine when to release resources. Furthermore, explicit invocation of resource management operations, such as the release of a resource, is tedious. How can the management effort be kept at a minimum while ensuring high efficiency and scalability?

In Chapter 5, *Guidelines for Applying Resource Management*, we present guidelines for applying resource management, which describe a recipe for applying the resource management pattern language to a particular domain effectively.

Chapter 6, *Case Study: Ad Hoc Networking*, shows how an ad hoc networking application can be built and its resource management requirements addressed using the patterns we describe.

Chapter 7, *Case Study: Mobile Network*, ties all the patterns into a pattern language, and uses that pattern language to address the requirements of a case study in the telecommunications domain.

In Chapter 8, *The Past, Present, and Future of Patterns*, Frank Buschmann continues the tradition of looking back at the forecasts about 'where patterns are going' made in the earlier POSA volume. He further analyzes where patterns are now, and makes predictions about the future of patterns.

Chapter 9, *Concluding Remarks*, completes the main content of the book. This chapter includes an analysis of what future work in the area of resource management might include.

Referenced Patterns documents all the patterns we reference as thumbnails, and *Notations* contains a key to all the notations that we use in this book.

For supplementary material, we encourage you to visit our Web site at `http://www.posa3.org`. The site contains all the source code that we present in the patterns, together with updates to the patterns themselves. The site also contains any additions to the resource management pattern language that have been made over time.

If you have any comments, constructive criticism, or suggestions for improvement, please send them to us via e-mail to authors@posa3.org.

Guide to the Reader

This book is constructed so that you can read it from cover to cover. If you know what you want to achieve, however, you may want to choose your own route through the book. In this case, the following hints can help you decide which topics to focus on and the order in which to read them:

- To see how individual patterns can be used in practice and how they work together, start with the problem and solution sections of the patterns, then continue with the case studies in Chapters 6 and 7.
- To get a feeling for the broad applicability of the pattern language, look for the abstracts and known uses of each pattern in Chapters 2 to 4.
- To see how the resource management pattern language fits into existing patterns work, especially those that touch on the area of resource management, refer to Section 1.5, *Related Work*.

We summarize the pattern language inside the front and back covers. Inside the front cover, we present a quick overview of the pattern language by listing the abstracts of all patterns. Inside the back cover, we present a pattern map illustrating the relationships between the patterns.

Acknowledgements

It is a pleasure for us to thank the many people who supported us in writing this book. Foremost, we would like to thank our shepherd, Charles Weir, and the reviewers of this book: Cor Baars, Frank Buschmann, Fabio Kon, Karl Pröse, Christa Schwanninger, Michael Stal, Christoph Stückjuergen, Bill Willis, and Egon Wuchner. Special thanks to the Silicon Valley Patterns Group, with outstanding contributions from Trace Bialik, Jeffrey Miller, Russ Rufer, and Wayne Vucenic. Collaborating on the Internet using Wiki to obtain

feedback from the Silicon Valley Patterns Group proved to be very effective.

We wish to thank our EuroPLoP and PLoP shepherds who have extensively reviewed individual patterns: Pascal Costanza, Ed Fernandez, Alejandra Garrido, Bob Hanmer, Kevlin Henney, Irfan Pyarali, Terry Terunobu, John Vlissides, and Uwe Zdun. We also wish to thank the reviewers of individual chapters: Roland Grimminger, Kevlin Henney, Michael Klug, Douglas C. Schmidt, Peter Sommerlad, and Markus Völter. Kevlin also supported us by providing many of the chapter citations, reviewing the correct usage of UML in our diagrams, and giving recommendations on the use of white space.

We also wish to thank Kirthika Parameswaran, with whom we worked on the idea of implementing JinACE [KiJa04], a Jini-like [Sun04c] framework implemented using C++. Our work inspired us to dig deeper into the concept of ad hoc networking and ultimately discover the patterns behind it. We want to thank her for the brainstorming sessions via long e-mail exchanges.

We are grateful to Douglas C. Schmidt, who encouraged us to view our research on JinACE from a pattern language perspective. The patterns that subsequently emerged motivated us to start working on the resource management pattern language.

Further, we wish to thank the pattern team at Siemens AG, Corporate Technology: Martin Botzler, Frank Buschmann, Michael Stal, Karl Pröse, Christa Schwanninger, Dietmar Schütz, and Egon Wuchner.

For the wonderful support on this book, we would like to thank our contacts at John Wiley & Sons: Gaynor Redvers-Mutton, Juliet Booker and Jonathan Shipley. Steve Rickaby, our copy editor, did a great job in helping us polish the book. It was a pleasure working with him.

Finally, we wish to express our special thanks to Frank Buschmann. He not only served as a reviewer of the book, and contributor of the chapter on *The Past, Present, and Future of Patterns*, but also gave us inspiration and encouragement to help us complete the book.

About The Authors

Michael Kircher

Michael Kircher is currently working as Senior Software Engineer at Siemens AG Corporate Technology in Munich, Germany. His main fields of interest include distributed object computing, software architecture, patterns, eXtreme Programming, and management of knowledge workers in innovative environments. He has been involved as a consultant and developer in many projects, within various Siemens business areas, building software for distributed systems. Among those were the development of software for UMTS base stations, postal automation systems, and operation and maintenance software for industry and telecommunication systems.

During his studies he was a member of the research group lead by Douglas C. Schmidt, where he was involved in the development of the Real-Time CORBA implementation TAO (The ACE ORB). That was when he discovered the world of patterns. In the course of implementing an efficient multi-threaded dispatching mechanism for TAO, he co-authored his first pattern with Irfan Pyarali, Leader/ Followers.

In recent years he has published at numerous conferences on topics such as patterns, software architecture for distributed systems, and eXtreme Programming. Together with Prashant, he organized several workshops on the topics covered by this book at conferences such as OOPSLA and EuroPLoP.

In his spare time Michael likes to combine family life with enjoying nature on foot and by bike. The best place for him to relax is in his custom-made hide, while watching wildlife accompanied by his hunting dog Ella.

Prashant Jain

Prashant Jain is currently working as a Technical Staff Member at IBM Research Labs in Delhi, India. His main fields of interest include distributed systems, e-commerce, software architecture, patterns and eXtreme programming. At IBM he has been doing research on emerging technologies in the area of e-commerce.

Prashant obtained his Masters degree in Computer Science from Washington University in St. Louis, U.S.A. It was there that he developed a keen interest in design patterns, and in 1996 co-authored his first pattern with his advisor, Douglas C. Schmidt. Since then he has been actively involved in the pattern community by means of authoring pattern submissions and organizing pattern workshops at conferences such as OOPSLA and EuroPLoP.

His passion for travel has seen Prashant living and working in countries that include India, Japan, the USA, and Germany. His professional experience includes working for companies such as Siemens AG, Fujitsu Network Communications Inc., and Kodak Health Imaging Systems Inc. He has also been actively involved in the Centre for Distributed Object Computing at Washington University.

In his spare time, Prashant enjoys travelling, dining out, watching movies and swimming. But his most cherished activity is engaging in logical conversations with his four-year old daughter Aanya, which often leave him speechless.

1 Introduction

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

Douglas Adams

A *resource* is an entity that is available in limited supply such that there exists a requestor, the *resource user*, that needs the entity to perform a function, and there exists a mechanism, the *resource provider*, that provides the entity on request. In the context of software systems, a resource can include, among other things, memory, synchronization primitives, file handles, network connections, security tokens, database sessions, and local as well as distributed services. A resource can be anything from a heavyweight object such as an application server component [VSW02] to a fine-grained lightweight object such as a file handle.

Determining what is a resource can sometimes be challenging. For example, in programming environments an image, such as a JPEG or

GIF file, is often referred to as a resource. In reality, however, there are no acquisition and release semantics defined for an image—instead, it is the data that makes up the image that constitutes a resource. Therefore a more accurate representation would be to treat the memory an image is using as a resource, which needs to be acquired when the image is loaded and released when the image is no longer needed.

There are numerous ways of categorizing resources. In the simplest, resources can be viewed as either *reusable* or *non-reusable*. Reusable resources typically are acquired from a resource provider, used and then released. Once the resources have been released they can be acquired and used again. An example of a reusable resource is memory that is allocated by and released to the operating system. Other examples of reusable resources include file handles and threads. Reusable resources are the most important form of resource, because the resource provider typically only has a limited number of resources, and therefore reusing resources that are not consumed makes logical sense. In contrast, non-reusable resources are consumed, and therefore once acquired are either not released, or their release is implicit. An example of a non-reusable resource is processing time in a computing grid [Grid04]—once the processing time is acquired and used, it is gone and cannot be returned.

A different method of categorizing resources is based on how the resources are accessed or used. A resource, once acquired, can be concurrently used either by multiple users or a single user. Examples of resources that are concurrently accessible by multiple users include services, queues, and databases. If a resource that can be concurrently accessed by multiple users has changeable state, then access to that resource needs to be synchronized. On the other hand, if a resource that can be concurrently accessed by multiple users does not have state, then no synchronization is needed. An example of a resource that does not need synchronization is a stateless session bean of a J2EE EJB [Sun04b] application server. On the other hand, a network communication socket is an example of a resource that needs synchronization. A resource that can be concurrently accessed by multiple users need not be acquired explicitly by each user. Instead, resource users can share references to the resource, such