Contents

About the Author

Greg Utas received his honors BSc in Computer Science from the University of Western Ontario (Canada). He joined Nortel Networks in 1981, where he served as the principal software architect for various switching products. As chief software architect of GSM NSS Development, he led a team of 50 designers who redesigned the product's call processing software using object-oriented techniques. For this work, he received the Nortel Technology Award for Innovation and became the first software architect at Nortel's director level. In March 2002, Greg joined Sonim Technologies as chief software architect, responsible for the design of push-to-talk services for wireless networks. He recently left Sonim to become a consultant specializing in the design of carrier-grade software.

Greg has presented papers at the International Switching Symposium, the International Workshop on Feature Interactions in Telecommunications and Software Systems, and at ChiliPLoP, a patterns conference. He has also authored a patterns paper in *IEEE Communications* and contributed a chapter to the book *Design Patterns in Communication Systems*.

Readers can e-mail Greg at *greg@carriergradesoftware.com*. He plans to post discussions related to the book at *www.carriergradesoftware.com*.

Preface

This book is about programming techniques for building carrier-grade software systems. Carrier-grade software is required in products that support mission-critical applications. These products include routers, switches, servers, and gateways found in communication networks. Firms that operate such networks are known as carriers, so the term carrier-grade refers to a product that meets their stringent quality requirements. However, the need for carrier-grade software is now emerging in other products, such as high-end web servers, storage area network equipment, and video-on-demand servers.

Carrier-grade software must meet extreme availability, reliability, and scalability requirements. Consequently, it employs many techniques that are not common practice in the computing industry. These techniques have been proven in products designed by firms such as Lucent, Nortel, and Ericsson. However, they have never been documented in a comprehensive manner. Instead, they are lore among the software engineers who develop products that depend on them. This book attempts to elucidate what is currently a black art.

TARGET AUDIENCE

This book is primarily intended as a guide for software engineers who work on products that face carrier-grade requirements. Managers and testers of such products will also find it useful. It could also serve as a reference text for advanced software engineering courses on highly available, distributed systems that support connection-oriented protocols.

This book will also help those who perform due diligence on products that claim to be, or need to become, carrier grade. This form of due diligence often neglects to assess a product's software architecture, even though this is a primary determinant of whether a product is, or can become, carrier grade. To serve this purpose, the last chapter of this book defines maturity levels for carrier-grade software. Development teams can also use these levels for self-assessment purposes.

MOTIVATION

The impetus for this book began at the 1998 ChiliPLoP conference in Arizona. At that conference, I participated in the TelePLoP hot topic, which was mostly attended by software architects working in the telecom industry. The other group members were Ward Cunningham, Dennis DeBruler, David DeLano, Jim Doble, Bob Hanmer, John Letourneau, and Greg Stymfal. Jim Coplien organized the hot topic. Linda Rising was the conference chair and therefore could not attend our sessions, but she later edited *Design Patterns in Communications Software* [RIS01], which included patterns written by TelePLoP participants. However, few of these patterns were programming patterns, and there were many carrier-grade topics which they did not cover.

We all felt that a comprehensive book on carrier-grade software techniques was needed. Nothing close to one existed. It was as if we all lived in pre-literate societies, where knowledge was handed down in an oral tradition. We could all recite cases in which teams developing new products were unfamiliar with techniques that we took for granted. In other cases, teams exposed to these techniques dismissed them in favor of others that were common practice in the computing industry and, therefore, surely better than bizarre ideas being touted by dinosaurs. Both situations led to products that failed to meet carrier-grade requirements. These products then had to be cancelled or rewritten, or their customers had to suffer until they could replace them. We believed that a book that included the rationale for our techniques could help to avert such unfortunate outcomes.

EXAMPLES

For many reasons, most of the examples in this book come from the telecom field. It is the most mature domain when it comes to experience in constructing carrier-grade software. Conveniently, readers

also have a general familiarity with the domain, given that everyone makes telephone calls. It is also an area in which I have worked as a software architect for over twenty years, so it offers me a ready source of examples that I know to be realistic rather than hypothetical.

The book contains a number of code examples. They are intended as sketches to illustrate the concepts described in the text. They are *not* carrier-grade in every detail, but they *do* convey the essence of what has been used in carrier-grade systems.

Typefaces are used as follows:

- Bold face (e.g. **five nines**) introduces technical terms.
- Small caps (e.g. OBJECT POOL) highlight names of techniques and patterns.
- COURIER (e.g. `vptr`) is used for code examples and types.
- COURIER ITALICS (e.g. *`setitimer`*) denote operating system calls.

Acknowledgments

For the past ten years, I worked for Subramaniam 'Loga' Logarajah in his role as VP of Engineering for various products that faced carrier-grade requirements. It is rare to find people in this type of role who demonstrate Loga's degree of commitment to software architecture. I wish to thank Loga for his support and trust, which allowed me the opportunity to acquire the experience needed to write this book.

After completing the first draft of this book, I received many thoughtful comments from reviewers. These ran to almost fifty pages, encompassing both style and technical comments that significantly improved the final draft. For their helpful and timely feedback, I wish to thank Dennis DeBruler, Jim Doble, Steven Fraser, Bob Hanmer, Stephanie Ho, Kim Holmes, Chris Honkakangas, Jagdish Patel, and Linda Rising.

Carrier-Grade
Software Techniques
Purposes
[a] availability
[c] capacity
[p] productivity
[r] reliability
[s] scalability
A → B A is a prerequisite for B
Technique use judiciously
Object Orientation [pr]
Object Class [pr]
Singleton [c]
Flyweight [c]
Registry [p]
Polymorphic Factory [p]
Cached Result [c]
Object Pool [ac]
Object Nullification [a]
Embedded Object [c]
Object Template [c]
Quasi-Singleton [c]
Object Morphing [c]
Scheduling
Thread Class [arp]
Cooperative Scheduling [acpr]
Run-to-Completion Timeout [a]
Run-to-Completion Cluster [pr]
Thread Pool [c]
Proportional Scheduling [ar]
Fault Avoidance [ar]
Defensive Coding
Stack Overflow Protection
Low-Order Page Protection
User Spaces
Write-Protected Memory
Fault Detection [ar]
Leaky Bucket Counter
Watchdog
Heartbeating
Messaging [s]
TLV Message [cpr]
Parameter Typing [pr]
Parameter Template [r]
Parameter Fence [r]
Parameter Dictionary [c]
Half-Sync/Half-Async [cpr]
Reliable Delivery [cpr]
Message Attenuation [cr]
In-Place Encapsulation [c]
Stack Short-Circuiting [c]
Message Cascading [c]
Message Relaying [c]
Eliminating I/O Stages [c]
Prefer Push to Pull [c]
No Empty Acks [c]
Polygon Protocol [c]
Callback [c]
Shared Memory [c]
Overload Controls [ar]
Finish What You Start
Discard New Work
Ignore Babbling Idiots
Throttle New Work
Fault Recovery [a]
Safety Net
Audit
Initialization Framework [p]
Escalating Restarts
Binary Database
Checkpointing [a]
Application Checkpointing
Object Checkpointing
Memory Checkpointing
Virtual Synchrony
Failover [a]
Load Sharing
Cold Standby
Warm Standby
or Hot Standby
Software Installation [a]
Hitless Patching
Protocol Backward Compatibility
Rolling Upgrade
Object Reformatting
Hitless Upgrade
Distribution [as]
Heterogeneous Distribution
Homogeneous Distribution
Hierarchical Distribution
Symmetric Multi-Processing
Half-Object Plus Protocol [c]
Field Debugging [pr]
Software Warning Log
Software Error Log
Object Dump
Flight Recorder
Object Browser
Function Tracer
Message Tracer
Tracepoint Debugger
System Operation [ar]
Configuration Parameters
Logs
Alarms
Maintenance
Operational Measurements
Capacity Management [c]
Set the Capacity Benchmark
Transaction Profiler
Function Profiler
Thread Profiler
Software Optionality [cp]
Conditional Compilation
Software Targeting
Run-Time Flags