Distributed Systems Architecture

PrefaceTo understand anything, you should not try to understand everything. — Aristotle The whole is greater than the sum of the parts; the part is greater than a fraction of the whole. — Aristotle Architecting is a challenging process of abstraction, composition, modularity, and simplification to create an architecture specification. An architecture specification captures the essence and definition of the system: understanding, parts, and the relationships among the parts. An architecture specification defines how a system solves a business problem within the scope of the business. — Putman Leave the beaten track occasionally and dive into the woods. You will be certain to find something that you have never seen before. — Alexander Graham Bell There are large gaps in the theory and practice of software architecture and engineering. Much is published about the representation of a software architecture, such as the Unified Modeling Language (UML), but little is available about the specification for a software architecture. Software engineering methods of domain engineering, process modeling languages, and well-formed patterns of reasoning aid in the specification of an architecture. The Reference Model of Open Distributed Processing (RM-ODP) defines the standard reference model for distributed software systems architectures, based on object-oriented techniques, accepted at the international level. RM-ODP is a standard adopted by the International Standards Organization (ISO) and the International Telecommunications Union (ITU). RM-ODP is embedded and used actively in mission-critical systems industries such as in telecommunications, in health care, on Wall Street (financial services industry), in various Government systems (Logistics), in European Government Agencies such as UK Aviation control systems, as a foundation for the Object Management Group (OMG) Object Management Architecture (OMA), for defining enterprise architectures, and for defining software architectures. The software systems architecture work that is emerging, and is focused either at the component level or at the systems level, provides a key resource for architecting. This is enhanced by the architecting techniques of RM-ODP. This book assembles these great ideas, explains what they mean, and shows how to use them for practical benefit, along with real-world case study examples. By using the RM-ODP specification constructs, associated languages, architecture patterns of reasoning, semantic behavior specification, and conformance testing abilities, readers will be able to architect their specific systems based on the RM-ODP specification foundations, and specify architectures that work. One of the purposes of this book is to provide the approach to using the RM-ODP foundations in architecting and specifying a distributed processing system that addresses such key properties as interoperability, dependability, portability, integration, composability, scalability, transparency, behavior specification, quality of service, policy management, federation, and conformance validation. Another purpose of this book is to explain the underlying foundations for creating an architectural specification. These foundations come not only from RM-ODP, but also from the current work in software systems architecture. Another purpose is to guide the reader to understand the importance and benefits of creating an architecture specification for an enterprise. Yet another purpose is to provide the reader with the principles to construct software systems architecture (at both introductory and in-depth levels). By applying the proven techniques of RM-ODP for what

makes a good architecture, readers will be able to build their own tailored architectures, and clearly represent them in UML or some other tool, with an understanding of the underlying principles. Practitioners of RM-ODP have found that the standard is extremely beneficial in guiding architecture definition and providing standard terminology/principles for distributed object applications and infrastructures from an enterprise perspective. Outstanding Features This book is intended to provide valuable insight into successful architecture specification by describing an unprecedented foundation to accomplish this task, describing the use of the foundation, explaining the relationships of the concepts of architecting, explaining the relationships of the concepts of distributed processing, and identifying the right methods and possible tools for architecting. All material for the book has been derived from actual experiences. A medical case study is used throughout the book in ever increasing detailed specification. This medical case study is based on actual experience of the author. In addition, many metamodels are provided to represent the concepts of RM-ODP. All of these metamodels are contributions from the author. This is information that readers can use and apply in their architecting today. RM-ODP provides a reference framework, grammars, methods of abstraction and composition, and separation of concerns to achieve an architecture specification of the system. RM-ODP provides a framework for this separation, using viewpoints, as well as separating out certain decisions (e.g., product decisions) until later. Further, the reference model provides a set of definitions, which always aids in communicating with others. There is little in the literature about RM-ODP or architecture specification, and certainly not a book dedicated as a tutorial of these subjects. Now there is. In summary, this book offers the following: How to manage the architecting process in the lifecycle of a system How to solve many architecture reuse and cost-effectiveness problems How to create a business specification How to understand and use the concepts of distributed processing in an architecture How to architect effectively How to specify an architecture How to understand and specify semantic behavior and nonfunctional properties of a system (the "ilities") How to provide the right level of detail in an architecture specification How to ensure the implementation conforms to the architecture specification How to use RM-ODP effectively How to use popular tools, such as UML, to describe an architecture A definitive tutorial of RM-ODP Audience This book is designed for: Those in the Distributed Software Systems Architecture community who are interested in a methodology for using proven architecture principles. Professional software architects who are looking for new ideas about architecting a system. Within this book, the reader will find discussions of the techniques for architecting, for creating an architecture specification, and RM-ODP's relationship to other architecture frameworks. Program managers interested in how to create a cost-effective architecture within their enterprise that focuses on the needs of the enterprise and solves an enterprise problem. They will learn how do to do this through an overview of RM-ODP, the program benefits for using it, and where RM-ODP fits in the system lifecycle process. Systems engineers interested in the lifecycle approach to enterprise architecture specification. Experienced engineers interested in expanding their understanding of how to create a valid architecture specification and gain an understanding of the distributed processing system concepts, why certain constructions are valid and why some are not, what is to be specified and how, and some new ideas and approaches to architecting a system. The reader will be able to develop a collection of useful distributed

processing architecting techniques that expand upon the current software systems architecture capabilities. Developers interested in the practice of architecture specification and aligning current technology to achieve a workable system, while allowing evolutionary changes in technology solutions. Researchers interested in solutions and aids for furthering the research work in architecture specification. Individuals in the software community who are generally interested in the application of an architecture method. Readers will find examples of the applications of RM-ODP and specific analysis techniques. The expected audience will be novice and mid-level program managers, software engineers, those in the IEEE, DoD, research communities, consortia, and general architecture readers. This book can be used as a textbook and reference book for studies in the methods of architecture; for graduate studies in software architecture specification; for training information about software architecture and RM-ODP; for further education of consultants, integration specialists, and acquisition managers who need to approve and fund such work; and for researchers who are expanding the discipline of software architecture. The inclusion of RM-ODP will bring to the U.S., principally, the outstanding work that was accomplished by the international standards working group. In brief, the RM-ODP principles form a solution set and foundation for all software architecting endeavors. It is the formalized framework for this topic, and at the International Standard (IS) level of acceptance. It forms a solution set and foundation for reuse of design patterns to provide cost-effective software architecture. It is the process for this topic, but has never before been described in a book. Many program managers (who typically set the stage as to the methodology of choice for a project), software engineers, and researchers in academia and in DARPA are unaware of the power and solutions provided by the standard, or the process of identifying and instantiating reuse of all the expensive assets of architecture. Many do not realize that there is a language for specifying software-intensive distributed processing, and that language is precisely and rigorously defined in RM-ODP for reuse. Those debating definitions for architecture, system, interface, and others can reuse the internationally agreed upon definitions. Finally, with the inclusion of RM-ODP and its relationship to other architecture frameworks, it is expected that many software engineers will benefit from reading this work, since it will be the first time these subjects are discussed in print. How to Use This Book This book is divided into four parts, aimed at increasing levels of detail. Part One provides an overview of the field of software architecture, an RM-ODP primer for managers, and an RM-ODP primer for architects. Part Two provides an in-depth study of RM-ODP and how to use it. Areas of importance and utility from RM-ODP are highlighted. Ambiguity in RM-ODP is highlighted. Warnings in the use of RM-ODP are highlighted. Part Three provides a discussion of the principal architecture patterns of use, arranged by topic. Several of these patterns of use come from emerging work under the initiative of RM-ODP, as well as lessons learned from the practice of RM-ODP. These patterns of reasoning used by the architect are founded on the principals of RM-ODP, as discussed in Part Two of the book. Part Four concludes with relating RM-ODP to other architecture methods. It also provides emerging technologies to further the patterns of reasoning for use in architecting, and a set of architecting heuristics. The information contained in this book is organized in a manner that provides clear insight into the world of distributed software-intensive processing architecture for designers and developers who are familiar with information systems

technology, but want to know more about how to build a good architecture. Starting with a tutorial about software architecture, and then a tutorial about the standard for software architecture, the reader need not be an expert in the area of international standards, RM-ODP, software architecture, or specific technologies. The book goes on to address the needs of the variety of readers for which it is intended. Each chapter in the book provides an overview of the subject of the chapter, as well as a summary. For those who wish a broad brush exposure to RM-ODP, the primers of Part One provide this, as well as the overviews and summaries in each chapter of interest. As each chapter progresses, in Parts Two and Three, more and more in-depth detail is provided. The readings of these chapters are aimed at those who wish to know the technical details of a topic. There are two case studies used throughout the book, at various levels of detail. The primary case study is a Hospital enterprise, based upon the author's experience with the medical profession. A secondary case study is an airline reservation system, also based upon the author's experience. These case studies are used to describe the concepts of RM-ODP, and to show how they might be used.

Delta-4 is a 5-nation, 13-partner project that has been investigating the achievement of dependability in open distributed systems, including real-time systems. This book describes the design and validation of the distributed fault-tolerant architecture developed within this project. The key features of the Delta-4 architecture are: (a) a distributed object-oriented application support environment; (b) built-in support for user-transparent fault tolerance; (c) use of multicast or group communication protocols; and (d) use of standard off the-shelf processors and standard local area network technology with minimum specialized hardware. The book is organized as follows: The first 3 chapters give an overview of the architecture's objectives and of the architecture itself, and compare the proposed solutions with other approaches. Chapters 4 to 12 give a more detailed insight into the Delta-4 architectural concepts. Chapters 4 and 5 are devoted to providing a firm set of general concepts and terminology regarding dependable and real-time computing. Chapter 6 is centred on fault-tolerance techniques based on distribution. The description of the architecture itself commences with a description of the Delta-4 application support environment (Deltase) in chapter 7. Two variants of the architecture - the Delta-4 Open System Architecture (OSA) and the Delta-4 Extra Performance Architecture (XPA) - are described respectively in chapters 8 and 9. Both variants of the architecture have a common underlying basis for dependable multicasting, i. e.

This second edition of Distributed Systems, Principles & Paradigms, covers the principles, advanced concepts, and technologies of distributed systems in detail, including: communication, replication, fault tolerance, and security. Intended for use in a senior/graduate level distributed systems course or by professionals, this text systematically shows how distributed systems are designed and implemented in real systems.

LOCUS, a distributed version of the popular operating system Unix, provides an excellent solution. It makes a collection of computers, whether they are workstations or mainframes, as easy to use as a single computer by providing a set of supports for the underlying network that is virtually invisible to users and - applications programs. Computer systems consisting of many machines will be the norm within a few years. However, making a collection of machines appear as a single, coherent system - in

which the location of files, servers, programs, or users is invisible to users who do not wish to know - is a very difficult problem. LOCUS, a distributed version of the popular operating system Unix, provides an excellent solution. It makes a collection of computers, whether they are workstations or mainframes, as easy to use as a single computer by providing a set of supports for the underlying network that is virtually invisible to users and - applications programs. This "network transparency" dramatically reduces the cost of developing and maintaining software, and considerably improves the user model of the system. It also permits a variety of system configurations, including diskless workstations, full duplex I/O to large mainframes, transparently shared peripherals, and incremental growth from one workstation to a large network including mainframes with no effect on applications software required to take advantage of the altered configurations. In addition to transparent, distributed operation, LOCUS features also include high performance and reliability; full Unix compatibility, support for heterogeneous machines and systems, automatic management of replicated file storage; and architectural extensions to support extensive interprocess communication and internetworking. Contents The LOCUS Architecture • Distributed Operation and Transparency • The LOCUS Distributed Filesystem • Remote Tasking • Filesystem Recovery • Dynamic Reconfiguration of LOCUS • Heterogeneity • System Management • Appendixes: LOCUS Version Vector Mechnism • LOCUS Internal Network Messages The LOCUS Distributed System Architecture is included in the Computer Systems series, edited by Herb Schwetman.

The new edition of this bestselling title on Distributed Systems has been thoroughly revised throughout to reflect the state of the art in this rapidly developing field. It emphasizes the principles used in the design and construction of distributed computer systems based on networks of workstations and server computers.

Explains fault tolerance in clear terms, with concrete examples drawn from real-world settings Highly practical focus aimed at building "mission-critical" networked applications that remain secure

Although much has been made of the impact XML is having on Web development, the most significant changes brought about by XML have been in the way distributed systems store and exchange information. XML Distributed Systems Design offers in-depth architectural models for devising open-ended systems and provides templates for complex data interchange and mining theories as related to XML. XML Distributed Systems Design addresses core XML technologies such as XSL, DTD, XML Query, Data Warehouses, Data Mining, Distributed Systems Architecture, Web-based system design, Distributed Systems Framework, SOAP, SAX and using XML enabled tools for development and problem solving. Close attention is given to the way XML changes existing development patters and paradigms. In addition, the book presents the new patterns and strategies emerging in XML system design.

The communications-served data-processing system. Today's teleprocessing systems. System trends. Evolution of configuration and function distribution. Improving line utilization. System objectives summary. The architectural layers. Basic concepts of systems network architecture. Higher-level services of sna network. Data flow control. Transmission control. Path control. Data link control. Overview of operations. Putting it together. Finite state architecture. Reliability and security control. Advanced functions.

Multidomain networks. Routing techniques. Interfacing to new data networks.

Fueled by ubiquitous computing ambitions, the edge is at the center of confluence of many emergent technological trends such as hardware-rooted trust and code integrity, 5G, data privacy and sovereignty, blockchains and distributed ledgers, ubiquitous sensors and drones, autonomous systems and real-time stream processing. Hardware and software pattern maturity have reached a tipping point so that scenarios like smart homes, smart factories, smart buildings, smart cities, smart grids, smart cars, smart highways are in reach of becoming a reality. While there is a great desire to bring born-in-the-cloud patterns and technologies such as zero-downtime software and hardware updates/upgrades to the edge, developers and operators alike face a unique set of challenges due to environmental differences such as resource constraints, network availability and heterogeneity of the environment. The first part of the book discusses various edge computing patterns which the authors have observed, and the reasons why these observations have led them to believe that there is a need for a new architectural paradigm for the new problem domain. Edge computing is examined from the app designer and architect's perspectives. When they design for edge computing, they need a new design language that can help them to express how capabilities are discovered, delivered and consumed, and how to leverage these capabilities regardless of location and network connectivity. Capability-Oriented Architecture is designed to provide a framework for all of these. This book is for everyone who is interested in understanding what ubiquitous and edge computing means, why it is growing in importance and its opportunities to you as a technologist or decision maker. The book covers the broad spectrum of edge environments, their challenges and how you can address them as a developer or an operator. The book concludes with an introduction to a new architectural paradigm called capability-based architecture, which takes into consideration the capabilities provided by an edge environment. .

In 1992 we initiated a research project on large scale distributed computing systems (LSDCS). It was a collaborative project involving research institutes and universities in Bologna, Grenoble, Lausanne, Lisbon, Rennes, Rocquencourt, Newcastle, and Twente. The World Wide Web had recently been developed at CERN, but its use was not yet as common place as it is today and graphical browsers had yet to be developed. It was clear to us (and to just about everyone else) that LSDCS comprising several thousands to millions of individual computer systems (nodes) would be coming into existence as a consequence both of technological advances and the demands placed by applications. We were excited about the problems of building large distributed systems, and felt that serious rethinking of many of the existing computational paradigms, algorithms, and structuring principles for distributed computing was called for. In our research proposal, we summarized the problem domain as follows: "We expect LSDCS to exhibit great diversity of node and communications capability. Nodes will range from (mobile) laptop computers, workstations to supercomputers. Whereas mobile computers may well have unreliable, low bandwidth communications to the rest of the system, other parts of the system may well possess high bandwidth communications capability. To appreciate the problems posed by the sheer scale of a system comprising thousands of nodes, we observe that such systems will be rarely functioning in their entirety.

REST continues to gain momentum as the best method for building Web services, and this down-to-earth book delivers techniques and examples that show how to design and implement integration solutions using the REST architectural style.
Distributed systems intertwine with our everyday lives. The benefits and current shortcomings of the underpinning technologies are experienced by a wide range of people and their smart devices. With the rise of large-scale IoT and similar distributed systems, cloud bursting technologies, and partial outsourcing solutions, private entities are encouraged to increase their efficiency and offer unparalleled availability and reliability to their users. The Research Anthology on Architectures, Frameworks, and Integration Strategies for Distributed and Cloud Computing is a vital reference source that provides valuable insight into current and emergent research occurring within the field of distributed computing. It also presents architectures and service frameworks to achieve highly integrated distributed systems and solutions to integration and efficient management challenges faced by current and future distributed systems. Highlighting a range of topics such as data sharing, wireless sensor networks, and scalability, this multi-volume book is ideally designed for system administrators, integrators, designers, developers, researchers, academicians, and students.
Future requirements for computing speed, system reliability, and cost-effectiveness entail the development of alternative computers to replace the traditional von Neumann organization. As computing networks come into being, one of the latest dreams is now possible - distributed computing. Distributed computing brings transparent access to as much computer power and data as the user needs for accomplishing any given task - simultaneously achieving high performance and reliability. The subject of distributed computing is diverse, and many researchers are investigating various issues concerning the structure of hardware and the design of distributed software. Distributed System Design defines a distributed system as one that looks to its users like an ordinary system, but runs on a set of autonomous processing elements (PEs) where each PE has a separate physical memory space and the message transmission delay is not negligible. With close cooperation among these PEs, the system supports an arbitrary number of processes and dynamic extensions. Distributed System Design outlines the main motivations for building a distributed system, including: inherently distributed applications performance/cost resource sharing flexibility and extendibility availability and fault tolerance scalability Presenting basic concepts, problems, and possible solutions, this reference serves graduate students in distributed system design as well as computer professionals analyzing and designing distributed/open/parallel systems. Chapters discuss: the scope of distributed computing systems general distributed programming languages and a CSP-like distributed control description language (DCDL) expressing parallelism, interprocess communication and synchronization, and fault-tolerant design two approaches describing a distributed system: the time-space view and the interleaving view mutual exclusion and related issues, including election, bidding, and self-stabilization prevention and detection of deadlock reliability, safety, and security as well as various methods of handling node, communication, Byzantine, and software faults efficient interprocessor communication mechanisms as well as these mechanisms without specific constraints, such as adaptiveness, deadlock-freedom, and fault-tolerance virtual channels and virtual networks load distribution problems

synchronization of access to shared data while supporting a high degree of concurrency

In the race to compete in today's fast-moving markets, large enterprises are busy adopting new technologies for creating new products, processes, and business models. But one obstacle on the road to digital transformation is placing too much emphasis on technology, and not enough on the types of processes technology enables. What if different lines of business could build their own services and applications—and decision-making was distributed rather than centralized? This report explores the concept of a digital business platform as a way of empowering individual business sectors to act on data in real time. Much innovation in a digital enterprise will increasingly happen at the edge, whether it involves business users (from marketers to data scientists) or IoT devices. To facilitate the process, your core IT team can provide these sectors with the digital tools they need to innovate quickly. This report explores: Key cultural and organizational changes for developing business capabilities through cross-functional product teams A platform for integrating applications, data sources, business partners, clients, mobile apps, social networks, and IoT devices Creating internal API programs for building innovative edge services in low-code or no-code environments Tools including Integration Platform as a Service, Application Platform as a Service, and Integration Software as a Service The challenge of integrating microservices and serverless architectures Event-driven architectures for processing and reacting to events in real time You'll also learn about a complete pervasive integration solution as a core component of a digital business platform to serve every audience in your organization.

Distributed systems intertwine with our everyday lives. The benefits and current shortcomings of the underpinning technologies are experienced by a wide range of people and their smart devices. With the rise of large-scale IoT and similar distributed systems, cloud bursting technologies, and partial outsourcing solutions, private entities are encouraged to increase their efficiency and offer unparalleled availability and reliability to their users. Applying Integration Techniques and Methods in Distributed Systems is a critical scholarly publication that defines the current state of distributed systems, determines further goals, and presents architectures and service frameworks to achieve highly integrated distributed systems and presents solutions to integration and efficient management challenges faced by current and future distributed systems. Highlighting topics such as multimedia, programming languages, and smart environments, this book is ideal for system administrators, integrators, designers, developers, researchers, and academicians.

Distributed systems have helped application development teams deal with failures, downtime, and poor scaling, but these systems bring technical challenges of their own. With this unique cookbook, system architects will get a detailed understanding of reactive systems, along with proven recipes for dealing with different architectural issues. Each self-contained chapter covers the architecture of an entire reactive system, and--since these systems share many of the same architectural issues--each chapter also focuses on a particular area, such as delivery semantics or monitoring & tracing, with detailed solutions for problems that commonly arise. Learn the architecture and implementation tips for an entire reactive microservices-based system in each chapter Understand the challenges of long-term running and evolution of your distributed system Explore different failure modes of distributed systems and the approaches to address them Learn about proper site reliability and production readiness

Data is at the center of many challenges in system design today. Difficult issues need to be figured out, such as scalability, consistency,

reliability, efficiency, and maintainability. In addition, we have an overwhelming variety of tools, including relational databases, NoSQL datastores, stream or batch processors, and message brokers. What are the right choices for your application? How do you make sense of all these buzzwords? In this practical and comprehensive guide, author Martin Kleppmann helps you navigate this diverse landscape by examining the pros and cons of various technologies for processing and storing data. Software keeps changing, but the fundamental principles remain the same. With this book, software engineers and architects will learn how to apply those ideas in practice, and how to make full use of data in modern applications. Peer under the hood of the systems you already use, and learn how to use and operate them more effectively Make informed decisions by identifying the strengths and weaknesses of different tools Navigate the trade-offs around consistency, scalability, fault tolerance, and complexity Understand the distributed systems research upon which modern databases are built Peek behind the scenes of major online services, and learn from their architectures

The primary audience for this book are advanced undergraduate students and graduate students. Computer architecture, as it happened in other fields such as electronics, evolved from the small to the large, that is, it left the realm of low-level hardware constructs, and gained new dimensions, as distributed systems became the keyword for system implementation. As such, the system architect, today, assembles pieces of hardware that are at least as large as a computer or a network router or a LAN hub, and assigns pieces of software that are self-contained, such as client or server programs, Java applets or pro tocol modules, to those hardware components. The freedom she/he now has, is tremendously challenging. The problems alas, have increased too. What was before mastered and tested carefully before a fully-fledged mainframe or a closely-coupled computer cluster came out on the market, is today left to the responsibility of computer engineers and scientists invested in the role of system architects, who fulfil this role on behalf of software vendors and in tegrators, add-value system developers, R&D institutes, and final users. As system complexity, size and diversity grow, so increases the probability of in consistency, unreliability, non responsiveness and insecurity, not to mention the management overhead. What System Architects Need to Know The insight such an architect must have includes but goes well beyond, the functional properties of distributed systems.

Intended as a textbook for undergraduate students of computer science, computer science and engineering, and information technology for a course on distributed systems/operating systems, this up-to-date text provides a thorough understanding of the fundamental principles and technologies pertinent to the design and construction of the distributed systems. Beginning with an introduction to the subject, the book discusses the techniques of software and network architectures and presents the issues pertaining to the handling and accessing of resources. This also focuses on major application areas. Finally, the book provides the examples for explaining the concepts discussed. The book would also be useful to postgraduate students of computer science, computer science and engineering, and information technology as well as to postgraduate students of computer applications. The book can also be used by software engineers, programmers, analysts, scientists and researchers for reference. New to This Edition This second edition highlights some of the latest distributed system technologies. It includes discussions on: • Cloud Computing • Social Networks • Big Data In addition to this, It presents some current key software tools, viz. BitTorrent, Amazon Dynamo, Amazon DynamoDB, Apache Cassandra, Apache Server, Apache Zookeeper, Google BigTable and others. Key Features • Introduces Internet, The World Wide Web, Web services and network technologies, viz. WAN, LAN and MAN. • Discusses software development tools, like PVM, MPI, DCE, CORBA and the Globus toolkit. • Provides discussions on network protocol suites, i.e. TCP/IP, SMTP and HTTP. • Deals with grid computing, wireless computing and client-server model. • Presents applications of NFS, Coda, Microsoft SQL Server, Oracle, Amoeba, Chorus, Mach, Windows NT and Orbix technologies. • Emphasizes the

programming languages, like Ada, C++ and Java.

Concentrating upon the design and usage of global security systems, this technical manual covers a wide diversity of application areas. Emphasizes the design and implementation of the comprehensive integrated security system already in operation at a number of key sites. Gives a broad overview of existing international standards and examines the latest research results along with practical products.

This book is written for computer programmers, analysts and scientists, as well as computer science students, as an intro duction to the principles of distributed system design. The emphasis is placed on a clear understanding of the concepts, rather than on details; and the reader will learn about the struc ture of distributed systems, their problems, and approaches to their design and development. The reader should have a basic knowledge of computer systems and be familiar with modular design principles for software development. He should also be aware of present-day remote-access and distributed computer applications. The book consists of three parts which deal with prin ciples of distributed systems, communications architecture and protocols, and formal description techniques. The first part serves as an introduction to the broad meaning of "distributed system". We give examples, try to define terms, and discuss the problems that arise in the context of parallel and distributed processing. The second part presents the typical layered protocol architecture of distributed systems, and discusses problems of compatibility and interworking between heterogeneous computer systems. The principles of the lower layer functions and protocols are explained in some detail, including link layer protocols and network transmission services. The third part deals with specification issues. The role of specifications in the design of distributed systems is explained in general, and formal methods for the specification, analysis and implementation of distributed systems are discussed.

Modern-day projects require software and systems engineers to work together in realizing architectures of large and complex software-intensive systems. To date, the two have used their own tools and methods to deal with similar issues when it comes to the requirements, design, testing, maintenance, and evolution of these architectures. Software and Systems Architecture in Action explores practices that can be helpful in the development of architectures of large-scale systems in which software is a major component. Examining the synergies that exist between the disciplines of software and systems engineering, it presents concepts, techniques, and methods for creating and documenting architectures. The book describes an approach to architecture design that is driven from systemic quality attributes determined from both the business and technical goals of the system, rather than just its functional requirements. This architecture-centric design approach utilizes analytically derived patterns and tactics for quality attributes that inform the architect’s design choices and help shape the architecture of a given system. The book includes coverage of techniques used to assess the impact of architecture-centric design on the structural complexity of a system. After reading the book, you will understand how to create architectures of systems and assess their ability to meet the business goals of your organization. Ideal for anyone involved with large and complex software-intensive systems, the book details powerful methods for engaging the software and systems engineers on your team. The book is also suitable for use in undergraduate and graduate-level courses on software and systems architecture as it exposes students to the concepts and techniques used to create and

manage architectures of software-intensive systems.

This book teaches you how to evaluate a distributed system from the perspective of immutable objects. You will understand the problems in existing designs, know how to make small modifications to correct those problems, and learn to apply the principles of immutable architecture to your tools. Most software components focus on the state of objects. They store the current state of a row in a relational database. They track changes to state over time, making several basic assumptions: there is a single latest version of each object, the state of an object changes sequentially, and a system of record exists. This is a challenge when it comes to building distributed systems. Whether dealing with autonomous microservices or disconnected mobile apps, many of the problems we try to solve come down to synchronizing an ever-changing state between isolated components. Distributed systems would be a lot easier to build if objects could not change. After reading The Art of Immutable Architecture, you will come away with an understanding of the benefits of using immutable objects in your own distributed systems. You will learn a set of rules for identifying and exchanging immutable objects, and see a collection of useful theorems that emerges and ensures that the distributed systems we build are eventually consistent. Using patterns, you will find where the truth converges, see how changes are associative, rather than sequential, and come to feel comfortable understanding that there is no longer a single source of truth. Practical hands-on examples reinforce how to build software using the described patterns, techniques, and tools. By the end, you will possess the language and resources needed to analyze and construct distributed systems with confidence. The assumptions of the past were sufficient for building single-user, single-computer systems. But as we expand to multiple devices, shared experiences, and cloud computing, they work against us. It is time for a new set of assumptions. Start with immutable objects, and build better distributed systems. What You Will Learn Evaluate a distributed system from the perspective of immutable objects Recognize the problems in existing designs, and make small modifications to correct them Start a new system from scratch, applying patterns Apply the principles of immutable architecture to your tools, including SQL databases, message queues, and the network protocols that you already use Discover new tools that natively apply these principles Who This Book Is For Software architects and senior developers. It contains examples in SQL and languages such as JavaScript and C#. Past experience with distributed computing, data modeling, or business analysis is helpful.

Many applications follow the distributed computing paradigm, in which parts of the application are executed on different network-interconnected computers. The extension of these applications in terms of number of users or size has led to an unprecedented increase in the scale of the infrastructure that supports them. Large-Scale Distributed Computing and Applications: Models and Trends offers a coherent and realistic image of today's research results in large scale

distributed systems, explains state-of-the-art technological solutions for the main issues regarding large scale distributed systems, and presents the benefits of using large scale distributed systems and the development process of scientific and commercial distributed applications.

The eagerly awaited Pattern-Oriented Software Architecture (POSA) Volume 4 is about a pattern language for distributed computing. The authors will guide you through the best practices and introduce you to key areas of building distributed software systems. POSA 4 connects many stand-alone patterns, pattern collections and pattern languages from the existing body of literature found in the POSA series. Such patterns relate to and are useful for distributed computing to a single language. The panel of experts provides you with a consistent and coherent holistic view on the craft of building distributed systems. Includes a foreword by Martin Fowler A must read for practitioners who want practical advice to develop a comprehensive language integrating patterns from key literature.

Salary surveys worldwide regularly place software architect in the top 10 best jobs, yet no real guide exists to help developers become architects. Until now. This book provides the first comprehensive overview of software architecture's many aspects. Aspiring and existing architects alike will examine architectural characteristics, architectural patterns, component determination, diagramming and presenting architecture, evolutionary architecture, and many other topics. Mark Richards and Neal Ford—hands-on practitioners who have taught software architecture classes professionally for years—focus on architecture principles that apply across all technology stacks. You'll explore software architecture in a modern light, taking into account all the innovations of the past decade. This book examines: Architecture patterns: The technical basis for many architectural decisions Components: Identification, coupling, cohesion, partitioning, and granularity Soft skills: Effective team management, meetings, negotiation, presentations, and more Modernity: Engineering practices and operational approaches that have changed radically in the past few years Architecture as an engineering discipline: Repeatable results, metrics, and concrete valuations that add rigor to software architecture

Systems Programming: Designing and Developing Distributed Applications explains how the development of distributed applications depends on a foundational understanding of the relationship among operating systems, networking, distributed systems, and programming. Uniquely organized around four viewpoints (process, communication, resource, and architecture), the fundamental and essential characteristics of distributed systems are explored in ways which cut across the various traditional subject area boundaries. The structures, configurations and behaviours of distributed systems are all examined, allowing readers to explore concepts from different perspectives, and to understand systems in depth, both from the component level and holistically. Explains key ideas from the ground up, in a self-contained style, with material carefully sequenced to make it easy to absorb and follow. Features a detailed case study that is designed to

serve as a common point of reference and to provide continuity across the different technical chapters. Includes a ‘putting it all together’ chapter that looks at interesting distributed systems applications across their entire life-cycle from requirements analysis and design specifications to fully working applications with full source code. Ancillary materials include problems and solutions, programming exercises, simulation experiments, and a wide range of fully working sample applications with complete source code developed in C++, C# and Java. Special editions of the author’s established ‘workbenches’ teaching and learning tools suite are included. These tools have been specifically designed to facilitate practical experimentation and simulation of complex and dynamic aspects of systems.

While there is a lot of appreciation for backend and distributed systems challenges, there tends to be less empathy for why mobile development is hard when done at scale. This book collects challenges engineers face when building iOS and Android apps at scale, and common ways to tackle these. By scale, we mean having numbers of users in the millions and being built by large engineering teams. For mobile engineers, this book is a blueprint for modern app engineering approaches. For non-mobile engineers and managers, it is a resource with which to build empathy and appreciation for the complexity of world-class mobile engineering. The book covers iOS and Android mobile app challenges on these dimensions: Challenges due to the unique nature of mobile applications compared to the web, and to the backend. App complexity challenges. How do you deal with increasingly complicated navigation patterns? What about non-deterministic event combinations? How do you localize across several languages, and how do you scale your automated and manual tests? Challenges due to large engineering teams. The larger the mobile team, the more challenging it becomes to ensure a consistent architecture. If your company builds multiple apps, how do you balance not rewriting everything from scratch while moving at a fast pace, over waiting on "centralized" teams? Cross-platform approaches. The tooling to build mobile apps keeps changing. New languages, frameworks, and approaches that all promise to address the pain points of mobile engineering keep appearing. But which approach should you choose? Flutter, React Native, Cordova? Native apps? Reuse business logic written in Kotlin, C#, C++ or other languages? What engineering approaches do "world-class" mobile engineering teams choose in non-functional aspects like code quality, compliance, privacy, compliance, or with experimentation, performance, or app size?

Computer Systems Organization -- Computer-Communication Networks.

The key area of open communications in distributed computing systems is explained in this authoritative text. International standards and management strategies are explained in the context of both global and local network developments.

This book is a guide to creating a software architecture comprised of distributed components. While it is based on OMG's CORBA standard, the principles also apply to architecture built with other technology, such as Microsoft's DCOM.

Designing Distributed Control Systems presents 80 patterns for designing distributed machine control system software architecture (forestry

machinery, mining drills, elevators, etc.). These patterns originate from state-of-the-art systems from market-leading companies, have been tried and tested, and will address typical challenges in the domain, such as long lifecycle, distribution, real-time and fault tolerance. Each pattern describes a separate design problem that needs to be solved. Solutions are provided, with consequences and trade-offs. Each solution will enable piecemeal growth of the design. Finding a solution is easy, as the patterns are divided into categories based on the problem field the pattern tackles. The design process is guided by different aspects of quality, such as performance and extendibility, which are included in the pattern descriptions. The book also contains an example software architecture designed by leading industry experts using the patterns in the book. The example system introduces the reader to the problem domain and demonstrates how the patterns can be used in a practical system design process. The example architecture shows how useful a toolbox the patterns provide for both novices and experts, guiding the system design process from its beginning to the finest details. Designing distributed machine control systems with patterns ensures high quality in the final product. High-quality systems will improve revenue and guarantee customer satisfaction. As market need changes, the desire to produce a quality machine is not only a primary concern, there is also a need for easy maintenance, to improve efficiency and productivity, as well as the growing importance of environmental values; these all impact machine design. The software of work machines needs to be designed with these new requirements in mind. Designing Distributed Control Systems presents patterns to help tackle these challenges. With proven methodologies from the expert author team, they show readers how to improve the quality and efficiency of distributed control systems.

Middleware is the bridge that connects distributed applications across different physical locations, with different hardware platforms, network technologies, operating systems, and programming languages. This book describes middleware from two different perspectives: from the viewpoint of the systems programmer and from the viewpoint of the applications programmer. It focuses on the use of open source solutions for creating middleware and the tools for developing distributed applications. The design principles presented are universal and apply to all middleware platforms, including CORBA and Web Services. The authors have created an open-source implementation of CORBA, called MICO, which is freely available on the web. MICO is one of the most successful of all open source projects and is widely used by demanding companies and institutions, and has also been adopted by many in the Linux community. * Provides a comprehensive look at the architecture and design of middleware the bridge that connects distributed software applications * Includes a complete, commercial-quality open source middleware system written in C++ * Describes the theory of the middleware standard CORBA as well as how to implement a design using open source techniques

Distributed file systems are network file systems where the server can be distributed across several physical computer nodes. File systems that share access to the same block storage are shared disk file systems.