DOWNLOAD PDF REAL TIME DESIGN PATTERNS BRUCE POWEL DOUGLASS

Chapter 1 : Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems | R

Bruce Powel Douglass is the Chief Evangelist for i-Logix, a leading producer of tools for real-time systems development. He contributed to the original specification of the UML and to the UML as one of the co-chairs of the Object Management Group's Real-Time Analysis and Design Working Group.

Preface Goals Real-time and embedded systems "RTE systems" must execute in a much more constrained environment than "traditional" computer systems, such as desktop and mainframe computers. RTE systems must be highly efficient, optimally utilizing their limited processor and memory resources, and yet must often out-perform systems with significantly more compute power. In addition, many RTE systems have important safety critical and high reliability requirements due to their use in systems such as avionics flight control, nuclear power plant control, life support, medical instrumentation, just to name a few. The creation of RTE systems to meet these functional and quality of service requirements requires highly experienced developers with decades of experience. Yet, over the years, as these developers development of such systems, they see the same kinds of problems recurring over and over - not exactly the same problems, but common threads. The very best developers abstract these problems and their solutions, into generalized approaches that have been consistently effective. These generalized approaches are called design patterns. They are often best applied at the level of the system or software architecture - the sum of design decisions that affect the fundamental organization of the system. Real-Time Design Patterns is an attempt to capture in one place a set of architectural design patterns that are useful in the development of RTE systems. Audience The book is oriented towards the practicing professional software developer and the computer science major, in the junior or senior year. This book could also serve as an undergraduate or graduate level text, but the focus is on practical development rather than a theoretical introduction. The book assumes a reasonable proficiency in at least one programming language and a basic understanding to the fundamental concepts of object orientation, the Unified Modeling Language UML , and real-time systems. Organization The first section consists of three chapters. Chapter 1 provides a very brief review of the major concepts in the Unified Modeling Language. Chapter 2 introduces the fundamental concepts of architecture, as they are defined in the Rapid Object-oriented Process for Embedded Systems ROPES , including the primary division of architecture into logical design-time and physical run-time aspects, and the 5 important architectural views. In the third chapter, the book gets into a discussion of design patterns and their role in defining architecture. Because it is difficult to discuss architecture in a process-free environment, the ROPES process, and the key technologies it tries to optimize, are introduced to provide a background in which design patterns may be effectively discussed. Once process has been introduced, design patterns are next. Their various aspects are explained and the fundamental organization of design patterns used in this book is provided. The chapter finishes with a discussion of how design patterns can be applied in the development of real systems. Section 2 contains the architectural design patterns which reify the ways that large scale system components are organized and structured to optimize some set of general system criteria. The patterns in the second section are organized around the architectural concept they address. Chapter 4 is dedicated to high-level structural patterns - focused around what is called the Subsystem or Component architecture. Because concurrency and resource management is so crucial to real-time and embedded systems, Chapter 5 focuses on the common patterns of concurrency. Memory management is crucial for many systems in this domain, and is the subject of Chapter 6. We see even more general resource management patterns in Chapter 7. Chapter 8 presents a number of common distribution architecture patterns that define how objects can be distributed across multiple address spaces and computers. Finally, Chapter 9 provides a number of patterns that deal with building safe and reliable architectures. Two appendices appear at the end of the book. The first is simply a summary of the UML graphical notation and the second is an index of the patterns by name. Other additional tools potentially useful for developers of real-time developers are also provided. The Papers chapter contains some papers on various topics as well as

some useful OMG specifications. More Information Additional information on the UML, object-oriented technology and the development of real-time systems can be found at www. Many other well-written and useful books on the UML and software engineering are similarly available. Most other publications about software patterns such as [1] have not addressed real-time systems per se in any depth, or have focused on the narrower and more advanced topic of real-time middleware [2] , or have been application domain specific [3]. This book offers a significant benefit to the practice of real-time computing, because software patterns and the UML enable potentially lower software costs in many systems. Real-time software spans the entire range of complexity and costs. In other real-time systems, the software is so large and complex that regardless of the hardware costs, the software costs are a major part of the system costs for example, software in a military or commercial aircraft. Barry Boehm, in his recent book updating the ubiquitous Cocomo software cost model [4], assigns an effort multiplier of 1. Most real-time software lies between these two extremes, and it is that mainstream audience of practitioners who will benefit the most from this book. Historically, developers of real-time software have lagged behind other developers in using the most contemporary software engineering methodologies. There are several reasons for this. One is, as mentioned above, that some real-time software is so simple that only the most elementary methodologies are needed. A more common reason is that many real-time systems with non-trivial software suffer from hardware capacity constraints due to size, weight, power, and so on. Software structured for purposes such as re-usability, modularity, or flexibility does tend to consume additional time or space resources. This is sometimes compensated for by the fact that commodity computing system hardware cost is always declining and its performance is always increasing. But in many real-time systems, hardware cost is still an easily measured quantitative factor that is thought to outweigh the hard-to-measure qualitative factors of software quality and costs. Yet another reason is that real-time software practitioners are frequently application experts who are not always educated enough in modern software engineering to understand and employ it properly. New computer science and engineering graduates rarely enter the real-time field, because their formal education has not exposed them to much if any significant realistic real-time practice real-time is a uniquely disadvantaged aspect of computer science and engineering in this respect , and what little real-time theory they may have learned is still of very limited practical relevance. This book provides an introduction to software patterns and the UML-by one of the most authoritative contributors to those topics-as applied to mainstream real-time software, in a manner that is easily understood by practitioners in that field without prerequisite knowledge. Another prospective benefit for many real-time software designers of becoming familiar with software patterns and the UML is that these issues are of rapidly increasing importance to building larger scale, more dynamic and complex, and more distributed real-time computing systems. Such systems offer highly significant albeit as yet not always fully appreciated added value to many enterprises, and hence offer perhaps the most challenging and rewarding career development opportunities in the field of real-time computing systems. This book is an excellent starting point toward that future. Douglas Jensen Natick, Massachusetts July Doug Jensen is widely recognized as one of the pioneers of real-time computing systems, and especially of dynamic distributed real-time computing systems. He has over three decades of hardware, software, and systems research and technology development experience in military and industrial real-time computing, and was on the faculty of the Computer Science Department of Carnegie Mellon University for eight years. He is currently in a senior technical leadership position at The MITRE Corporation, where he conducts research and technology transition on real-time computing systems for projects of strategic national interest. Upper Saddle River, NJ: Prentice Hall, January Elements of Reusable Object-Oriented Software. Design Patterns for Avionics Control Systems, http:

DOWNLOAD PDF REAL TIME DESIGN PATTERNS BRUCE POWEL DOUGLASS

Chapter 2 : CiteSeerX â€" Real-Time Design Patterns

Auto Suggestions are available once you type at least 3 letters. Use up arrow (for mozilla firefox browser alt+up arrow) and down arrow (for mozilla firefox browser alt+down arrow) to review and enter to select.

Bruce has also written previous editions of Real-Time UML and earlier books on programming in Pascal and Basic and on numerical analysis. Publications and Webcasts Metrics: Metrics discussed include project, design, and modeling metrics. Bruce Powel Douglass explores how agile methods can be applied to the development of real-time systems. Agile Development for Safety-Critical Systems: Dobbs February www. The Rational Aerospace Solution embodies an integrated tool environment leveraged by integrated best practices and dynamic project governance to enable engineering teams to achieve both their economic and technical goals. Based on the Rational Best Practices Library, the Integrated Software Development Process for DOB ISDP incorporates best practices for model-based software development in addition to the standard software engineering disciplines of quality assurance configuration management. These practices provide guidance for engineers performing these tasks with dynamic links to the DOB standard, checklists, task descriptions, and other forms of guidance. These task definitions support automatic enactment in the Jazz-based platform of the Rational Aerospace Solution, enabling simple assignment, tracking, and dashboard-based project governance. Together, the tooling, practice definition, and process enactment provide a powerful environment for realizing the benefits of model-based development. Agile for Safety-Critical Systems: Quality Assurance Practices Dr. Dobbs January www. By attending this webcast, you will learn: Taylor Bruce Powel Douglass is a writer with an attitude. He is passionate and enthusiastic about real-time systems, safety and reliability, and direct execution of models. This first article identifies the major notational and semantic features of UML without a great deal of formality. The second part of this series shows how UML applies to a real-time system development problem. Part three wraps up the series on UML with a focus on architectural, mechanistic, and detailed design. The basics of safety and capturing of fault metadata for analysis.

DOWNLOAD PDF REAL TIME DESIGN PATTERNS BRUCE POWEL DOUGLASS

Chapter 3 : IBM Meet the experts - United States

Real-time programming guru Bruce Powell Douglass collects the best design patterns from this unique, and rapidly growing, area of programming, and presents them in an instructional format that teaches the reader the 'what, when, and how' of leveraging the significant power of these proven design solutions.

He can be reached at bpd ilogix. An up-and-coming trend in the real-time and embedded development environments is the move toward object technology. While object technology offers improved abstraction and encapsulation mechanisms, better reuse, and far better scalability over traditional structured methods, questions arise when applying this technology to designing and implementing predictable real-time systems. Some of the design issues to consider to get predictable real-time and embedded-system designs are the concurrency model, data sharing and task synchronization, and capturing and testing the timeliness of systems. Object model examples are provided to illustrate the pedagogical points of the actual application of these techniques to the construction of real-time and embedded systems. Real-Time Systems and Predictability An important aspect of embedded computing is the interaction with the external environment. In addition to logical correctness of computations, which is the main focus in desktop computing, embedded devices have various timing requirements that must be met by the deployed system. The nature of the timing requirements varies widely depending on the specific embedded device. The metrics for real-time systems include: Predictably fast response to urgent events. The device should perform its intended function in a predictable manner, while supporting other functions. High degree of schedulability. The timing requirements of the system must be satisfied at high degrees of utilization. We use the term "schedulability" broadly to also include transmissions over a network. Stability under transient overload. When the system is overloaded by events and it is impossible to meet all the deadlines, the deadlines of selected critical tasks must still be guaranteed. As a result of strong application demand, established theories in the field of real-time systems, developed and published over a number of years, are becoming commercially available solutions. We apply one of the key theories -- Rate Monotonic Analysis -- to an example later in this article. Figure 1 represents the high-level deployment diagram. This system is responsible for responding to one process input temperature within the reactor. Handling multiple process inputs can be achieved using the same principles outlined here. The temperature needs to be sampled, digitized, and filtered using a smoothing filter, and converted to engineering units suitable for further computation. The system also monitors an emergency shutdown switch, which is operated manually. A digital control action to shut down the reaction must be initiated for the following conditions within the specified times: Within ms of the magnitude of the temperature exceeding degrees C. Within ms of the operator requesting a manual shutdown via the switch. Alarms of high temperature must be reported to the operator within ms. Current temperature of the reactor must be displayed in a numeric field on the operator station console with no more than one-second lag. Temperature must be accumulated, acquired at Hz sample rate, displayed in a waveform window with no more than ms lag time, and stored to a black box recorder. Jitter must be less than 10 ms between drawn waveform points to ensure sufficient smoothness. Classes form the fundamental structural unit of object-oriented systems. A class is, roughly speaking, the type of an object, just as int is the type of the object " The class diagrams identify the classes themselves and their arrangements into collaborations via association relations among them and into taxonomic hierarchies via generalization relations among them. Figure 2 shows a class diagram for our reactive controller example. The objective is to identify the external interfaces clearly. The use- case diagram looks like Figure 3. There is no direct, obvious mapping of the use- case model to the class model. In general, a use-case is realized by a collaboration of objects working together. This is complicated by the fact that an object can participate in multiple collaborations. The ovals are the use-cases -- chunks of functionality visible to one or more actors that do not reveal or imply the internal structure necessary to implement that functionality. Actors are objects that are outside the scope of the system but that interact with the system.

Behavioral models, particularly sequence diagrams, are used to elaborate the details of the system-actor interaction. UML provides different ways to represent behavior. The behavior of reactive classes are modeled using statecharts, a visual formalism originated by David Harel. A state is a condition of existence of an object or, more generally, of a UML metaclass, called a "classifier" that persists for a significant period of time. An object transitions among states in response to received events occurrences in time and space that are of interest to the object. Transitions are shown by directed arcs connecting the states. Statecharts also include the ability to nest states deal with states at multiple levels of abstraction and to divide the behavior of an object up into independent regions called "and-states" and-states are separated by dashed lines. Figure 4 is a statechart for the Sensor class of the control example. Behavior of collaborations may be defined by statecharts associated with the use-case or, more commonly, scenarios shown by sequence diagrams. Figure 5 is an example sequence diagram. Temporal Modeling Temporal models represent the system timing characteristics. Examples of such characteristics are: The objective of temporal modeling of real-time systems is to present a consistent view of the system design such that its timing predictability can be readily determined using efficient and proven scientific techniques. These techniques include mathematical methods based upon RMA for analyzing the worst-case timing behavior of the system, and discrete-event simulation for determining the average-case behavior of the system. A combination of these techniques presents a methodology that can be used to capture both the hardware and software characteristics of distributed real-time systems, analyze and predict the worst-case timing behavior of the system using RMA, and study the average-case behavior of the system using discrete-event simulation techniques. The RC system is specified from the point of view of its external interfaces. A number of timing requirements are identified explicitly. The first three can be considered hard real-time requirements, because any single failure to meet any of these would result in the system failing in a potentially spectacular fashion. The remaining requirements may be considered "soft" because occasionally missing the specified requirement by a small amount is unlikely to have severe consequences. Figure 5 shows how the interactions of the system with the actors is depicted with a sequence diagram. The instance lines map one-to-one with the structural pieces on the use- case diagram.

Chapter 4 : Resource Patterns | Dr Dobb's

Bruce Powel Douglass, PhD Telelogic, Inc. calendrierdelascience.com Real-Time Design Patterns Identify design patterns that optimize the system (architectural) or.

Chapter 5 : Bruce Powel Douglass (Author of Real-Time UML)

Bruce Powel Douglass, Ph.D. Page 1 Real-Time Design Patterns Bruce Powel Douglass, Ph.D. Chief Methodology Scientist I-Logix Some of the contents of this paper are adapted from the author' s book Real-Time UML.

Chapter 6 : Real-Time Design Patterns : Bruce Powel Douglass :

Bruce Powel Douglass, who has a doctorate in neurocybernetics from the USD Medical School, has over 35 years of experience developing safety-critical real-time applications in a variety of hard real-time environments.

Chapter 7 : Bruce Douglass | About

Bruce Powel Douglass, Fine grained patterns for real-time systems, UML for real: design of embedded real-time systems, Kluwer Academic Publishers, Norwell, MA, Xin Ben Li, Feng Xia Zhao, Formal development of a washing machine controller by using formal design patterns, Proceedings of the 3rd WSEAS international conference on Computer.

DOWNLOAD PDF REAL TIME DESIGN PATTERNS BRUCE POWEL DOUGLASS

Chapter 8 : Real Time UML : Bruce Powel Douglass :

By Bruce Powel Douglass, June 01, This excerpt from Bruce Powel Douglass's "Real-Time Design Patterns" investigates resource patterns, such as the Critical Section Pattern, the Priority Inheritance Pattern, and the Highest Locker Pattern.

Chapter 9 : Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems | InformIT

In this book, Bruce Douglass illustrates for the first time how two important contemporary software engineering advancesâ€”patterns and the UMLâ€”can be applied advantageously to the concepts and techniques traditionally used in mainstream real-time software.