Using Patterns to Capture Architectural Decisions

Neil B. Harrison, *Utah Valley State College*
Paris Avgeriou, *University of Groningen*
Uwe Zdun, *Vienna University of Technology*

By providing information about a decision's rationale and consequences, architecture patterns can help architects better understand and more easily record their decisions.

Throughout the software design process, developers must make decisions and reify them in code. The decisions made during software architecting are particularly significant in that they have system-wide implications, especially on the quality attributes. However, architects often fail to adequately document their decisions because they don't appreciate the benefits, don't know how to document decisions, or don't recognize that they're making decisions. This lack of thorough documentation can significantly disrupt the system when decisions made later, during subsequent development iterations, conflict with the original architectural decisions.

Researchers are investigating various methods and tools to help software architects effectively document their decisions. However, such tools' utility has yet to be fully validated. Documenting architectural decisions thus remains difficult and we continue to lose important information.

Architecture patterns address these documentation challenges by capturing structural and behavioral information and encouraging architects to reflect on decisions in a way that doesn't interfere with the natural architectural design process. Architecture patterns are easy to use, giving developers a rich set of information about rationale, consequences, and related decisions. Essentially, they offer reusable knowledge for the architect's toolkit. Here, we discuss the relation between patterns and decision making and describe how architects can use patterns to capture certain architectural decisions in practice.

Problem overview

Most software architecture documentation describes the system's structure from different views.[1] Ideally, this documentation also records decisions that architects made while designing the system. Recording only the decision does little good, however; for the documentation to be truly useful, architects must also capture the alternatives considered, their expected consequences, and the rationale—that is, the reasons for selecting a particular alternative. Our discussion of decision documentation here refers not just to the decision but rather to all of its aspects. Unfortunately, when architects document their decisions, this wider definition is what they most neglect.

0740-7459/07/$25.00 © 2007 IEEE

Current research trends in software architecture focus on architectural decisions[2] as first-class entities and capture their explicit representation in the architectural documentation. Such documentation extends the typical views of a software system's architecture—such as the interacting components and connectors—with explicit representations of the architectural decisions that convey the rationale underlying a particular design.[3]

Knowledge vaporization

The ultimate goal of documenting architectural decisions is to alleviate a major problem in the field: architectural knowledge vaporization.[4] This knowledge vaporizes because architects fail to record their decisions, so significant information about a software system's architecture is unavailable during the development and evolution cycles. These decisions can't be explicitly derived from the architectural models. And, because they exist merely as tacit knowledge in the heads of architects or other stakeholders, they inevitably dissipate. As the well-known saying in software architecture goes, "If something is not written down, it does not exist."

Knowledge vaporization has consequences across the software industry, including expensive system evolution, lack of stakeholder communication, limited reusability of architectural assets, and poor traceability between the requirements, the architecture, and the implementation.

Documentation challenges

If recording architectural decisions is to become standard practice, then documenting decisions must be easy and somewhat automatic. To this end, researchers are investigating conceptual models, methods, processes, and tools for documenting decisions.[5–7] However, in practice, architects still fail to document their decisions for many reasons. The most significant include the following:

- The substantial effort required to document and maintain architectural decisions seems greater than the perceived benefit.
- Architects sometimes make decisions without realizing it or without reflecting explicitly upon them, so they don't know what to document.
- Rather than disrupt the creative flow of design, architects defer decision documentation until the architecture is essentially complete; at that point, they've often forgotten many decisions and the rationale behind them.
- Architects don't know how to document their decisions.

Clearly, such difficulties make the process of documenting architectural decisions problematic, leading to loss of valuable architectural knowledge.

The patterns solution

Architecture patterns are solutions to general architectural problems that developers have verified through multiple uses and then documented. They thus offer an effective way to capture some of the most significant design decisions and provide appropriate alternative solutions for various design challenges. Pattern documentation includes the pattern's usage context—that is, a recurring design problem solved by a recurring solution that resolves both the problem's general challenges and the solution's consequences.

Patterns help mitigate the four primary documentation challenges as follows:

- Architecture patterns include general structural and behavioral information, making it easier and faster to document architectural decisions.
- In applying architecture patterns, architects make decisions that encourage them to both reflect on those decisions and consider related issues.
- Pattern selection is indispensable to the architecting process, so architects can record related decisions with little effort. Pattern usage thus fits within the natural flow of the architecture design process.
- Patterns follow an easily understood form, which is highly compatible with proposed description templates for architectural decisions.

Patterns and decisions: A comparison

As the following descriptions and comparisons show, architecture patterns and architectural decisions have much in common.

Patterns: Coupling structure and consequences

Patterns are solutions to recurring problems.

Ideally, architecture documentation records decisions that architects made while designing the system.

Architecture patterns often dictate a particular high-level, modular system decomposition.

A pattern describes a problem, its context, and a generic solution. Many developers use patterns to document solutions to software problems. The best-known software patterns describe solutions to object-oriented design problems,[8] but patterns have also been used in many aspects of software design, coding, and development.

Architecture patterns are similar to OO design patterns in that both provide a problem's solution in context. Rather than directly producing code, however, architecture patterns work at the architectural design level, describing an abstract, high-level system structure and its associated behavior. Architecture patterns often dictate a particular high-level, modular system decomposition.

One of architectural patterns' key benefits is that they capture the system's general architectural structure—which is typically well known and easily recognized—and couple it with consequences that are often not as readily recognized. This is particularly useful when attempting to reconstruct architectural decisions: the system's structure indicates the (explicit or implicit) architecture pattern. The pattern description, in turn, indicates consequences of the architectural decision (especially with respect to quality attributes). These consequences are, in effect, less apparent decisions derived from the primary decisions. Developers can use this valuable knowledge to evaluate an architecture, although they can more precisely measure a pattern's actual impact on quality attributes through thorough analysis, such as quantitative performance analysis. The particular pattern variant used also indicates whether alternative variants or related patterns might be applied.

Common architecture patterns include

- Layers, which decomposes the system into a hierarchy of partitions, each of which exchanges services with adjacent partitions;
- Pipes and Filters, which encapsulates data-stream processing steps into filter components; and
- Blackboard, which centralizes the data from independent computation processes.[9]

Paris Avgeriou and Uwe Zdun offer a comprehensive overview of currently identified architecture patterns.[10] Another way to describe a system's architectural structure is through its architectural style,[11] which Mary Shaw first described in 1988.[12] Most agree that architectural styles and patterns are essentially the same concept.[1,10,13] Here, we refer to both as "patterns."

Decisions: Capturing key information

An architectural decision is a decision that affects the system architecture. Jan Bosch proposed that a decision consists of requirements and a solution, and that each design decision addresses some system requirements while leaving others unresolved.[4]

According to Bosch, design decisions might

- add components to the architecture,
- impose functionality on existing components,
- add requirements on components' expected behavior, or
- add constraints or rules on part or all of the software architecture.

He goes on to state that an architectural decision can represent many solution structures, including an architectural style or pattern.

A crucial consideration of design decision documentation is which information to collect. That is, what critical information about a decision should you convey to other architects and developers? Key information includes the issue being designed, the decision made, the alternatives considered, and the reasoning behind the decision. Anton Jansen and Jan Bosch characterize this information as a problem, motivation, cause, context, potential solutions, and decision.[3] Jeff Tyree and his colleagues describe this and other important information about decisions and give a sample template for recording them.[14]

A second important consideration is to determine what kinds of information comprise architectural decisions. Philippe Kruchten[2] describes several types of design decisions:

- Existence decisions relate to the behavior or structure in the system's design or implementation.
- Non-existence decisions describe behavior that is excluded from the system.
- Property decisions state an enduring, overarching system trait or quality, which might include design guidelines or constraints.
- Executive decisions are those driven by external forces, such as financial imperatives.

Another consideration here is the important distinction between two knowledge types:[15]

- *Application-generic knowledge* is an architect's implicit knowledge, gained through previous experiences in one or more domains (such as architectural patterns, tactics, or reference architectures).
- *Application-specific knowledge* involves all the decisions made during a particular system's architecting process, as well as the architectural solutions that implemented the decisions.

These two knowledge types are related in that application-generic knowledge is used to make decisions for a single application, and thus constructs application-specific knowledge.

As we noted earlier, a key difficulty with architectural decisions is in getting people to record the critical information surrounding a decision, rather than just recording the decision itself. To this end, researchers are developing tools to make the recording process as easy and unobtrusive as possible.[6,7] In addition to tools that explicitly document architectural decisions, model-driven software development researchers have developed tools for defining architectural metamodels with constraints and model-checking features. We can easily extend MDSD tools to metamodels for architectural decisions (for example, following the templates described in the next section) and use them to define and automatically check formalizable constraints that result from an architectural decision. This hypothesis, however, remains to be tested in practice; we're not yet aware of any MDSD tools that can effectively record architectural decisions.

The pattern–decision relationship

Architecture patterns and architectural decisions are complementary concepts. Using a pattern in system design is, in fact, selecting one of the alternative solutions and thus making the decisions associated with the pattern in the target system's specific context. For example, an architect designing a user interface structure might consider two alternative patterns: Model-View-Controller and Presentation-Abstraction-Control. The MVC pattern divides the application into components that contain the core functionality and data (the model), the views presented to the user, and the user-input controller. The PAC pattern creates a hierarchy of cooperating agents, each of which manages its own data display. The PAC pattern is extensible but less efficient than MVC. So, in deciding which pattern to use, the architect must consider the target system's performance and extensibility needs.

The major difference between architecture patterns and architectural decisions is in the scope of information each contains. Each architectural decision document describes an individual decision about the target system. In contrast, patterns describe solutions that have proven successful in multiple applications. Thus, architectural decisions are specific, but tentative; patterns are proven, but general. When designing systems, architects consider patterns as alternative solutions. In relation to the two knowledge types described earlier, architectural decisions comprise application-specific knowledge, whereas architecture patterns comprise application-generic knowledge.

Although patterns and decisions have different origins, we can investigate their relation by comparing how they're documented. Architectural decisions include the issue to be decided, the alternative solutions, the decision made, and the reasons for the decision. Similarly, a pattern describes the issue (in a problem section) and the decision (in a solution section). Alternative solutions are motivated by forces (different variants of the solution) and justified in a rationale section. Table 1 shows the typical sections in patterns documentation (Frank Buschmann and colleagues offer examples in their book[9]), architectural decision documentation,[15] and their correspondence.

As table 1 shows, patterns and architectural decisions also differ in their documentation format. Although they have many of the same sections, pattern descriptions focus on timeless, generic knowledge (and hence have a name, examples, and known uses), whereas decision templates focus on concrete knowledge relating to a specific situation (and hence contain elements such as status and notes).

Another interesting aspect is how the two facilitate solution selection. In the patterns realm, architects can derive alternative solutions in two ways. First, as table 1 shows, an individual pattern can provide alternative solutions by resolving the forces in different ways using different variants. The Pipes and Filters pattern, for example, might appear in different variants such as purely sequential, forks/joins, feedback loops, and so on. Second, two or more patterns can be complementary in a specific decision topic. For example, when deciding

Table 1

Pattern and architectural decision documentation

Pattern section	Decision section	Comments
Name		Patterns represent generic knowledge, so pattern names give the pattern a recognizable, reusable name to facilitate communication. Because decisions are knowledge-specific to the current situation, they're not intended to serve as a "language" among the architects/developers.
Problem	Issue	The pattern's problem statement roughly corresponds to the issue requiring a decision. In both cases, it expresses a stakeholder's concern that must be addressed.
Category	Group	Some pattern authors categorize their patterns in some scheme; correspondingly, decisions are grouped. The decision groups are usually rather clear because they're rooted in a concrete decision process, whereas pattern categories are often rather abstract.
	Status	Status information, such as pending, decided, or approved, refers to concrete realization of a decision. As generic knowledge, a pattern doesn't need such a section.
Context	Assumptions, constraints	A pattern's context and a decision's assumptions and constraints both set the scene and characterize the situation in which the pattern can be used or the decision is applied.
Solution variants according to forces	Positions	A decision's positions are the alternatives that have been considered to tackle the issue. This roughly corresponds to two parts of the pattern text: the forces describe various concerns that can lead to different solutions; the variants of the solution represent alternatives in solving the problem by balancing the concerns in a different way.
Solution	Decision	A pattern's solution describes the generic solution to the recurring design problem covered by the pattern. This corresponds to the concrete decision that resolves the issue of a decision.
Rationale	Argument	A pattern describes the generic rationale of applying the pattern's solution in relation to the forces. Similarly, a decision's argument section explains why the decision was made.
Resulting context/ consequences	Implications	A pattern's resulting context section describes the context that is created by applying the pattern. A pattern's consequences section describes the consequences of its application. These sections correspond to a decision's implications.
Example, known uses		Known uses are the sources from which the pattern has been mined; examples show how to apply the pattern's generic solution in a specific way. Because decisions are concrete knowledge, neither known uses (there's only one) nor examples are needed.
Related patterns	Related decisions, requirements, artifacts, or principles	A pattern's solution often leads to a context in which other related patterns can be applied. This corresponds directly to the related decisions, requirements, artifacts, or principles of a decision.
	Notes	In decision templates, notes can be taken during the decision process as part of the communication between stakeholders. Even though a lot of communication usually occurs when patterns are written, notes aren't explicitly recorded but informally captured in other sections of the pattern template or in the verbose text in the other pattern sections.

on interacting components' distributed communication, you might choose the Client-Server, Peer-to-Peer, or Publish-Subscribe pattern or combine two or all three.

As table 1 shows, patterns can support traditional architectural documentation. The patterns provide application-general knowledge in the areas of assumptions, constraints, positions, implications, and related decisions. The architect might wish to augment this information with application-specific knowledge; in this case, the pattern serves as a reminder of issues to consider. In some cases, a pattern contains nearly all the desired decision documentation (albeit at a general level). In such cases, the architect must document little beyond the decision itself. So, using patterns in decision documentation can minimize the efforts necessary to document extra information, such as design considerations, consequences, and so on.

When using pattern-oriented knowledge, it's important to understand the consequences of applying the pattern on functional and (especially) nonfunctional system aspects. When you decide to use a pattern, you decide to accept its consequences. The Layers pattern, for example, partitions software in a way that often results in many function calls, which might decrease performance. In deciding to use this pattern, you must consider its performance impact. However, because the Layers pattern supports security levels in the application, you might use it if

you want to adopt a particular security model and implementation. This brings up an important advantage of using patterns with respect to decision-making: A decision's consequences are rarely fully understood or even anticipated. Because patterns are based on extensive prior experience, the consequences are generally well understood and described in the pattern documentation. Thus, pattern usage can help you understand the consequences beforehand and document them for future reference.

Using patterns: Practical considerations

Architectural design is an especially challenging decision-making process because it involves frequent trade-offs: A given structure often satisfies a few requirements at others' expense. Furthermore, a decision's consequences might introduce new requirements, so you might have to trade off a solution's benefits with the additional system burdens it entails. Trade-offs are particularly rich and complex among a system's nonfunctional attributes. For example, deciding to implement a certain security approach might impact the system's performance and usability. Because of the interaction complexities among performance, usability, and security, architects might be particularly unaware of their decisions' consequences on such nonfunctional system aspects.

The architecting process is highly intuitive. To develop an architecture, architects use their own past experiences, others' experiences, and whatever application-generic architectural knowledge is available. Using a proven and systematic approach to architecting is highly desirable—you get no style points for originality in software architecture!

During architecting, architects periodically consider one or more of the key architectural drivers—that is, the most important system-affecting requirements. They consider alternative structural approaches, decide on one or more, and repeat the process. Ideally, they should record these decisions as they happen. However, as we noted earlier, they generally document the decisions later, if at all.

Patterns play an important role in this decision-making process. For certain decision topics, architects might select one or more patterns or a single pattern's variants as alternative approaches. When they select the pattern, its usage documents an architectural decision. This has several key benefits. First, the solution has been proven to work. Second, because the literature describes patterns in detail, documentation of pattern-associated decisions already exists. Third, many architecture patterns include documentation of their consequences and system impact, including on nonfunctional requirements. Thus, architects can easily learn which further trade-offs they must consider.

As we now describe, there are several advantages and limitations to using patterns as a primary method of architectural documentation.

Benefits of patterns use

Perhaps the biggest challenge of architectural documentation is capturing the critical information surrounding the decision itself. Doing this takes time, effort, and attention; consequently, developers tend to avoid interrupting the design flow to document their work. However, postponing documentation increases the risk that they'll forget critical issues or forgo documentation all together. This is precisely where patterns shine: Their use is easily noted (without interrupting design), and at the very least, the additional information reminds architects what issues to document later. The application-generic knowledge of rationale, forces, and consequences is an important first step. Patterns address the principal difficulties of recording decisions as follows:

- *The substantial effort required to document and maintain architectural decisions seems greater than the perceived benefit.* Because patterns include a description that matches architectural decisions' required description, using the pattern is a starting point for documenting that decision. Even if developers expend no additional documentation effort, the pattern name itself refers to the generic pattern description and thus offers at least minimum documentation.
- *Architects sometimes make decisions without realizing it or without reflecting explicitly upon them, so they don't know what to document.* Applying patterns per se signifies that some of the most significant architectural decisions have been made. Furthermore, patterns explicitly state the consequences of the system quality attributes, and this helps architects recognize their decisions and implications. Patterns also contain references to related patterns, which help architects think about alternative solutions and

Architecture patterns don't relieve the architect of all responsibility for documenting decisions.

eventually select one based on a rationale.

- *Rather than disrupt the creative flow of design, architects defer decision documentation until the architecture is essentially complete; at that point, they've often forgotten many decisions and the rationale behind them.* Patterns fit well within several well-established architecture design methods. They also emerge naturally through the design process without disrupting the creative flow. Nonetheless, developers can easily document decisions related to the pattern's usage afterwards by simply reusing the pattern description information. Finally, architecture patterns fit well into the tools that support architecting methods, and we expect such tools to become more mature and more widely used.
- *Architects don't know how to document their decisions.* Patterns contain much of their own documentation. They're also compatible with emerging decision documentation formats and tools. Patterns also remind architects of issues to consider.

Limitations and further research

Architecture patterns don't relieve the architect of all responsibility for documenting decisions. First, the architect must still document application-specific decisions. Second, not all decisions have appropriate patterns. While additional architecture patterns have been and will continue to be written,[10] some architecture areas will never have patterns. So, architecture patterns will always have a limited solution space.

Similarly, you can't capture some architectural decisions in terms of patterns because they depend on the project's concrete scope and domain. Technology-related decisions (such as deciding on a specific technology vendor) or organizational decisions (such as company guidelines or project team setup) are just two examples of project-dependent decisions that have severe consequences for the resulting architecture. We need further research to integrate pattern-based architectural decision documentation with these kinds of decisions.

A fourth limitation relates to the fact that architects often use multiple patterns together. If they don't understand the various pattern interactions, they might select conflicting patterns. This problem is exacerbated by the fact that architects tend to use architecture patterns unsystematically. We're conducting ongoing research on pattern-based architecture and design approaches that account for these issues. For example, elsewhere[16] we proposed an approach to support pattern selection based on desired quality attributes, and systematic design decisions based on patterns. We propose deriving a pattern language's grammar to systematically describe the pattern relationships and annotating the grammar with effects on quality goals. In a second step, we further analyze complex design decisions using the design spaces covered by a set of related software patterns.

Finally, an important challenge with patterns is what to do if developers use the wrong pattern but don't discover this until well into the implementation phase. As with any architectural decision, backing out is difficult. However, we might draw on the rich information that patterns contain to reduce such difficulty. To our knowledge, this area has yet to be researched at all.

The power of patterns has so far seen rather limited industrial usage: architects focus on their structural solution and use their names for communication purposes. However, as we've described here, patterns have great potential for providing invaluable architectural knowledge that architects can turn into application-specific knowledge and document as an architectural asset.

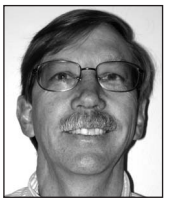
Finally, there remains the challenging question of architectural decision rationale—that is, *why* an architect made a particular decision. This key aspect of architectural knowledge must be recognized and made more explicit and systematic. Rationale becomes explicit when we study the similarities between the description formats of patterns and decisions. Patterns typically have forces that can both provide deeper insight into the problem and give information that illuminates the rationale behind the solution. Patterns can also inform us about trade-offs (such as space for time), adding to the decision's rationale. Ultimately, providing such information about a decision's rationale might be a most significant contribution that patterns make to architectural decision documentation.

References

1. P. Clements et al., *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2002.

2. P. Kruchten, "An Ontology of Architectural Design Decisions in Software Intensive Systems," *Proc. 2nd Groningen Workshop Software Variability*, Rijksuniversiteit Groningen, 2004, pp. 54–61.
3. A.G. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," *Proc. 4th Working IEEE/IFIP Conf. Software Architecture* (WICSA), IEEE CS Press, 2005, pp. 109–119.
4. J. Bosch, "Software Architecture: The Next Step," *Proc. 1st European Workshop Software Architecture*, LNCS 3047, Springer, 2004, pp. 194–199.
5. P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning about Architectural Knowledge," *Proc. 2nd Int'l Conf. Quality of Software Architecture* (QOSA 06), Springer, 2006, pp. 42–58.
6. A. Jansen et al., "Tool Support for Using Architectural Decisions," *Proc. 6th Working IEEE/IFIP Conf. Software Architecture* (WICSA), IEEE CS Press, 2007, pp. 6–9.
7. R. Capilla et al., "A Web-Based Tool for Managing Architectural Design Decisions," *Proc. Workshop Sharing and Reusing Architectural Knowledge* (SHARK), *Software Eng. Notes*, ACM SIGSOFT, vol. 31, no. 5, 2006.
8. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
9. F. Buschmann et al., *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, 1996.
10. P. Avgeriou and U. Zdun, "Architectural Patterns Revisited—A Pattern Language," *Proc. 10th European Conf. Pattern Languages of Programs* (EuroPLoP), UVK Konstanz, 2005, pp. 431–470.
11. D. Garlan, R. Allan, and J. Okerbloom, "Exploring Style in Architectural Design Environments," *Proc. ACM Symp. Foundations of Software Eng.* (SIGSOFT), ACM Press, 1994.
12. M. Shaw, "Toward Higher-Level Abstractions for Software Systems," *Proc. Tercer Simposio Int'l Conocimiento y su Ingerieria* [*Proc. 3rd Int'l Symp. Knowledge and Its Engineering*], 1988, pp. 55–61; reprinted in *Data and Knowledge Eng.*, vol. 5, 1990, pp. 19–28.
13. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice,* 2nd ed., Addison-Wesley, 2003.
14. J. Tyree and A. Ackerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, Mar./Apr. 2005, pp. 19–27.
15. P. Lago and P. Avgeriou, "First Workshop on Sharing and Reusing Architectural Knowledge," *Proc. ACM Symp. Foundations of Software Eng.* (SIGSOFT), ACM Press, 2006, pp. 32–36.
16. U. Zdun, "Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis," to be published in *Software: Practice and Experience*, R.H. Horspool and A.J. Wellings, eds., John Wiley & Sons, 2007.

About the Authors



Neil B. Harrison is an assistant professor of computer science at Utah Valley State College. His research interests include software patterns, effective organizations, and software testing. He has an MS in computer science from Purdue University. He is coauthor of *Organizational Patterns of Agile Software Development* (Prentice Hall, 2004). The Pattern Languages of Programs conference series named its Neil Harrison Shepherding award in his honor. He's a member of the ACM. Contact him at Utah Valley State College, 800 W. University Parkway, Orem, UT 84058; harrisne@uvsc.edu.

Paris Avgeriou is an assistant professor in the Department of Mathematics and Computing Science at the University of Groningen. His research interests include software engineering and software architecture, with an emphasis on architectural knowledge, modeling, evolution, and patterns. He received his PhD in software engineering from the National Technical University of Athens. He is a member of the IEEE, the European Research Initiative on Informatics and Mathematics, and Hillside Europe. Contact him at the Dept. of Mathematics and Computing Science, Univ. of Groningen, Groningen, Netherlands; paris@cs.rug.nl.

Uwe Zdun is an assistant professor at the Vienna University of Technology. His research interests include software patterns, software architecture, service-oriented architecture, distributed systems, language engineering, and object orientation. He received his doctoral degree in computer science from the University of Essen. He is a coauthor of *Remoting Patterns—Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware* (John Wiley & Sons, 2004) and *Software-Architektur: Grundlagen, Konzepte, Praxis* [*Software Architecture: Foundations, Concepts, Practice*] (Elsevier/Spektrum, 2005). Contact him at Distributed Systems Group, Information Systems Inst., Vienna Univ. of Technology, Vienna, Austria; zdun@acm.org.