Editor: **Cesare Pautasso**
University of Lugano
c.pautasso@ieee.org

Editor: **Olaf Zimmerman**
University of Applied Sciences
of Eastern Switzerland, Rapperswil
ozimmerm@hsr.ch

# A Decade of *Enterprise Integration Patterns*

## A Conversation with the Authors

**Olaf Zimmermann, Cesare Pautasso, Gregor Hohpe, and Bobby Woolf**

**IN AN INDUSTRY** that thrives on constant change, few books can survive the test of time. *Enterprise Integration Patterns* (*EIP*)[1]—with its highly influential collection of messaging patterns—is definitely one of those few. So, we interviewed the authors Gregor Hohpe and Bobby Woolf; here, we have the pleasure of sharing their reflections with you. You can discover the inside story of their book project as well as their views on pattern language design and on integration technology's evolution. We also thank them for their precious advice for the next generation of pattern authors and integration solution designers.

> Each pattern represents a decision, so the language walks the reader through the decisions that need to be made.

### A General Retrospective

**Olaf Zimmermann:** How did your book come to be? How did you get together, and how did you find your contributors and reviewers?

**Bobby Woolf:** Martin Fowler was the matchmaker. When he wrote *Patterns of Enterprise Application Architecture*,[2] Kyle Brown pointed out that his pattern language was not addressing asynchronous messaging. Martin felt that he already had plenty of patterns to write, which motivated Kyle and me to submit a collection of 27 patterns to the 2002 Pattern Language of Programs [PLoP] conference under the title, "Patterns of System Integration with Enterprise Messaging."[3]

**Gregor Hohpe:** I was based some 3,000 miles away, using enterprise application integration [EAI] tools, such as TIBCO and Vitria, in my consulting job. I had a nagging feeling that these tools share underlying concepts, which are obfuscated by different terminology. Martin encouraged me to document my findings in the form of patterns, also to be submitted to PLoP 2002,[4] where I first met Bobby and Kyle.

**Bobby:** So Kyle brought me into the effort he'd started with Martin, then Martin brought Gregor in. While Martin and Kyle contributed a lot of material and guidance, they eventually lessened their involvement, leaving Gregor and me to write and complete the book. Gregor and I hadn't known each other before, so it was a crash getting-to-know-you opportunity.

With encouragement from Martin and Kyle, we decided to combine our papers with the goal to turn them into a book. While there was some merging to be done, the two papers complemented each other well. We only had a handful of patterns that overlapped: Kyle's and my paper described message patterns ("message construction" in the book) and message client patterns (later "messaging endpoints").

**Gregor:** Coming from the EAI perspective, my 17 patterns focused on what was "between" the endpoints: message routing, message transformation, and message management. It also contained an early version of the pattern icons.

0740-7459/16/$33.00 © 2016 IEEE

**Cesare Pautasso:** In retrospect, what do you still like best about your book?

**Bobby:** The fact that the book's content is still relevant after a dozen years is quite a rarity for computer books and a testament to the power of using patterns to document expertise. Patterns capture common behavior across products. The forces that influence the solution, packaged into a consistent format with an expressive name, do not depend on a specific technology. As a result, the book has continued to stay relevant when applied to new ESB-style products and even to cloud integration.[5] [ESB stands for enterprise service bus.]

> Pattern icons bring the quality of Alexander's pattern sketches to the software world.

**Gregor:** The proudest moment was easily when Grady Booch listed our book as one of the great software pattern books at the 2005 Object-Oriented Programming, Systems, Languages, and Applications conference, right next to *Design Patterns*.[6] The visual language is one of the key features of the book and has inspired other software pattern authors to use icons for their patterns.[7]

**Bobby:** I agree; the diagrams with an icon for each pattern are one of the best features of the book. Gregor clearly had a very good vision (literally!) for how to show these solutions using what John Crupi, the coauthor of *Core J2EE Patterns*,[8] eventually dubbed "Gregorgrams." The Pipes and Filters architectural style that underlies the messaging pattern language allows the icons to be easily composed into larger solutions. This style of diagramming integration solutions became so popular that readers even developed stencils for Visio and Omnigraffle for them.

**Gregor:** The icons bring the quality of Christopher Alexander's pattern sketches[9] to the software world. Most other software pattern books use class or sequence diagrams, which resemble blueprints. Alexander's hand-drawn sketches highlight the essence of the pattern and remind the reader that a pattern is not a copy-and-paste solution. For example, Alexander's Bed Alcove pattern does not specify that the alcove has to be 6′8″ long and 3′ wide. Rather, it depicts the essence in a rough sketch.

**Olaf:** And what would you do differently now?

**Gregor:** I would make an icon for the Idempotent Receiver pattern, which describes a receiver that can process the same message multiple times without any harm. Somehow we seem to have missed that one.

**Bobby:** Nah, we didn't miss that one. We tried to create an icon but couldn't think of how to draw idempotency. If you couldn't draw it, I certainly couldn't!

**Gregor:** Regarding the icons, I would have liked to reflect that an endpoint can combine multiple patterns; for example, an idempotent consumer can also be transactional and polling. The composition of endpoint patterns is different from Pipes and Filters, so it might have benefitted from a different visualization.

**Bobby:** I wish we had better separated transport concepts from integration to cover messaging with unqueued transports. A lot of the techniques in the book do apply to both queued and unqueued messaging, as we see with HTTP-style messaging or representational state transfer [REST]. For example, the root pattern of our language is Messaging. I'd like to refine that into Messaging, Queued Messaging, and Unqueued Messaging. Most of the patterns in the language apply to both queued and unqueued messaging. With queued messaging, you get some additional qualities and techniques.

**Gregor:** We also didn't include much on error handling, except Dead Letter Channel. That's where all the bad messages go—but then what happens? Describing error-handling strategies requires a broader vocabulary that includes state,[10] which would have expanded the scope of the book significantly. We felt that 700 pages is plenty!

## Pattern-Language Design

**Cesare:** How are your patterns different from others from an organizational viewpoint?

**Gregor:** We documented a pattern language, not a pattern catalog,

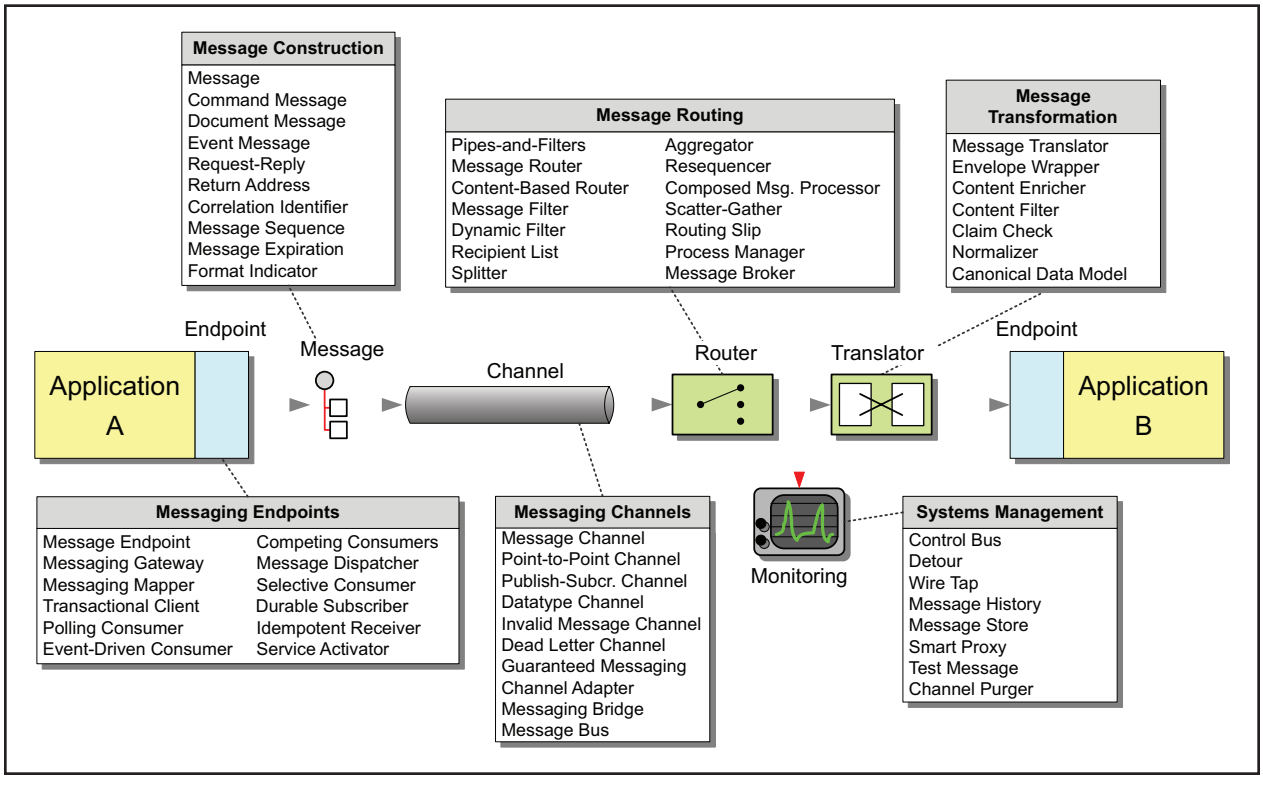


**FIGURE 1.** The messaging pattern language follows a message's flow, presenting root patterns for each major component of an integration solution. The root patterns guide more detailed design decisions toward selecting from alternative patterns for concrete problems. (Source: www.enterpriseintegrationpatterns.com/patterns/messaging; used with permission.)

meaning the patterns build on each other and form an ecosystem. Our pattern language is structured into six major sections, which are essentially arranged chronologically along the flow of a message. First, a message is created, then placed on a channel, then routed and transformed, and finally consumed. The illustration on the inside back cover reflects this nicely (see Figure 1). It is different from Alexander, who used a progression from the macroscopic view to the microscopic view: from large-city planning down to small details like the placement of ceiling lights.

**Bobby:** Actually, the root patterns in Chapter 3 [of *EIP*] provide a macroscopic big-picture view, which is detailed by the subsequent patterns. We explained this in the diagram "Relationship of Root Patterns and Chapters" in the book's introduction. The pattern language guides readers in decomposing the problem: to first make major decisions, then minor ones.

**Olaf:** Talking about stepwise refinement and architectural decisions, how do you suggest choosing or combining patterns? Do some of them exclude one another?

**Bobby:** When designing an integration solution, you are likely to combine patterns that derive from different root patterns. You'll want to create messages, place them on a channel, route and transform them, and so on. For a specific problem, you are likely to choose between alternatives—for example, by using the decision tree in the "Message Routing" chapter. It guides you to the right pattern on the basis of whether the router handles one message at a time or multiple messages, whether it publishes as many messages as it consumed, and so on. The introduction to each pattern chapter, as well as the related-patterns section for each pattern, helps guide the reader on which patterns can be used in combination and which are alternative choices.

**Gregor:** Indeed, some chapters present clear alternatives. For example,

you will need to decide between Messaging and File Transfer to solve a specific problem. Such patterns share the same pattern context. Other chapters are designed to combine multiple patterns. For example, an Idempotent Receiver may also be a Polling Consumer. These relationships are described in the chapter introductions. In hindsight, we could have formalized this a bit.

**Cesare:** Did you consider including messaging antipatterns? Why or why not?

**Gregor:** Despite the accessible format, patterns have an elaborate structure where dissecting the problem reveals forces that are resolved in the solution. The resulting context alerts you to implementation pitfalls and guides you to subsequent patterns. I find it difficult to incorporate this type of tension and resolution into antipatterns. Therefore, they often end up being less rich.

**Bobby:** We avoided antipatterns because they tell the reader what not to do. We wanted to focus on what to do. For instance, I think ambiguously structured data, where the sender and receiver can use any message format they'd like, is an antipattern. It's better to structure message data such that its schema can be determined and ensure that all senders and receivers follow that structure. But an antipattern that says "Don't use message formats with ambiguously structured data" doesn't provide good guidance on what to do.

Another antipattern we continue to see is to not consider messaging. Remote, synchronous connections between distributed components are brittle and make the interaction less reliable. Yet developers use them extensively. I hear "messaging is slow," when actually it optimizes throughput. Simpler messaging products like RabbitMQ and MQ Light [MQ stands for message queueing] may increase adoption, but only if developers overcome the synchronous mindset.

**Gregor:** Another worry of mine is that some people want to solve every problem with messaging and our patterns. That was not the intent. If you need low-latency, synchronous interaction, by all means use a synchronous protocol and no message queues. You may still benefit from many of the patterns, as Bobby outlined earlier.

> When a pattern is implemented as a feature or as a standard, it's still a pattern.

## Message-Oriented Middleware and Technology Evolution

**Cesare:** Did your patterns get implemented faithfully, or have any of your patterns been misused or misinterpreted? Can your patterns serve as a benchmark to compare competing messaging offerings?

**Bobby:** Most open source ESBs have embraced our pattern language. It has given these products a much-needed reference point and has certainly benefited our book as well. It's a symbiotic relationship.

Our patterns document best practices, which products should implement and support. It's not surprising and in fact beneficial for products to be similar to our product-neutral documentation.

Early products influenced us, and our documentation influenced later products—it's a circle of life. Later users benefit from products that better fit the best practices for using them and may not even recognize how easy they've got it or whom to thank. Some early feedback on our book drafts claimed, "That's not a pattern; that's a feature." But Sean Neville stated that when a pattern is implemented as a feature or as a standard, it's still a pattern, just one that's become much easier to apply.[1]

**Gregor:** I never considered our patterns to be a feature checklist—patterns are implementation advice, not guidance on product selection. It certainly helps integration developers that many of our patterns are implemented in the ESB platforms, but this is not required for all pattern languages. For example, in the early days of the "Gang of Four" (GoF) book,[6] some companies tried to implement those patterns inside the integrated development environment ("Make me an Observer"), but that did not work well. Being soft around their edges, patterns are not copy-and-paste code snippets.

**Olaf:** Let's look at the state of the practice in integration and messaging. What works well, and what's still missing?

**Gregor:** A wire-level protocol like AMQP [Advanced Message Queueing Protocol] has long been missing from the messaging world. MQTT expands messaging to the Internet of Things (IoT). While messaging has many elegant properties, of course it is not meant for everything—streaming and synchronous protocols have their place, too. Documenting streaming patterns would be a great exercise to understand commonalities and differences between streaming and messaging patterns. My guess would be that we would see some overlap.

**Bobby:** I agree. As much of computing has evolved—first to service-oriented architecture (SOA) and more recently to cloud, mobile, the IoT, microservices, and the API economy—integrating the parts has become as important an aspect of application design and development as automating business logic. Queued messaging products have been fairly stable, whereas unqueued protocols are constantly evolving, from Corba and Enterprise JavaBeans (EJB) to SOAP and RESTful HTTP.

**Cesare:** Along those lines, how are trends such as cloud computing, digitalization, and the IoT affecting messaging and your patterns? Can they benefit from your patterns?

**Bobby:** As I pointed out already, the more things change, the more they stay the same, and *EIP* remains relevant. SOA drove the adoption of ESBs, which incorporate our patterns. Now microservices require integrating the process components, so messaging solutions will be needed for those as well. Hybrid cloud applications with components deployed across multiple clouds will need messaging. Hybrid IT applications with systems of engagement hosted in the cloud connecting to systems of record hosted in private datacenters will need messaging. Mobile devices tenuously connected to their back-end applications can benefit from messaging. The IoT is a massive set of components dynamically joining and leaving spontaneous networks communicating via low-bandwidth, unreliable connections; that's the full-employment act for messaging solutions. Even when the messaging is synchronous, it helps decouple the components. Queued messaging and asynchronous invocation decouple components even more.

Queued messaging products have been fairly stable, whereas unqueued protocols are constantly evolving.

**Gregor:** I agree. I found many of our patterns in the recently released Google Cloud Pub/Sub service.[11] As systems become more distributed and more interconnected, integration is not only remaining relevant but also becoming more important.

## Looking Ahead

**Cesare:** What are you working on these days? Do you have any plans for a second edition, or do you think about other pattern languages or book projects?

**Bobby:** These days, I'm focused on cloud computing, especially for enterprises with existing IT systems running in datacenters. Applications running in the cloud need to integrate with those running in private datacenters, and the parts of hybrid applications running in separate clouds need to integrate with each other. Same dance, different tune.

**Gregor:** I rebooted my career a few times. When writing *EIP*, I was in consulting, which provided me with valuable input for the book. Becoming tired of travel, I switched to Internet-scale software development at Google. Now I am the chief architect at a large insurance company to bring that Internet-scale development and many of the topics Bobby mentioned into large-company IT. Essentially, I am now Bobby's customer.

**Bobby:** Lucky you!

**Gregor:** I was considering updating *EIP* with contemporary examples—those are the only parts of the book that aged. Sean may also want to rewrite the "futures" section after 12 years. On the other hand, readers still find the book useful as it is—it has become a classic. So we might not mess with it too much and (humbly) follow the footsteps of the GoF, who also never published a second edition.

At the same time, integration is much more than just messaging. For example, on top of messaging, systems engage in stateful conversations, execute workflows, or publish events in an event-driven architecture. There are a lot more patterns to be mined in

this space, which would also do the full scope of the title "Enterprise Integration Patterns" justice. I started collecting patterns on stateful conversations,[12] which one day may become *EIP* volume 2. I am also thinking about documenting IT transformation patterns that help traditional IT shops enable digital business. You can tell I like the patterns format for capturing and disseminating expertise.

**Olaf:** Do you have any advice for aspiring technical writers and prospective pattern authors?

**Bobby:** Pick your audience. A eureka moment early on was when we discovered that we're not documenting how to implement a messaging system, but how to use one.

**Gregor:** Writing well is hard, but worth the effort. Complex technical topics need the reader's complete attention, so the text must be free from noise and clutter. The curse of writing is that text is linear but most concepts aren't. This is one reason our book includes many navigation aids: the front and back inside cover plus the decision trees and tables in the chapter introductions. These allow readers to navigate the book in the order of their problem, not the order of the pages.

**Bobby:** Indeed. With a rich pattern language, each reader may take a different path through the book to best address the particular situation he or she is trying to solve. The same reader may take different paths developing different solutions.

**Gregor:** It's a myth that technical books are written by someone hiding in a corner, typing away. During the review phase alone, our material grew from 400 to 700 pages, based on feedback. It made the book much stronger and well rounded.

**Bobby:** Our material was extensively reviewed by experienced pattern authors, such as Ralph Johnson and Martin Fowler, and integration experts, such as Mike Rettig and Sean Neville, both of whom ended up contributing to the book. Such feedback not only helps improve the content but also helps the authors know when the material is becoming solid enough to be published. After 18 months of poring over your own words, you really need that external reference point.

**Gregor:** As you would have guessed, writing a book is a lot of work, so you need perseverance and strong commitment. Martin was a great mentor who kept us going in those tough moments when we assumed we were done, but suddenly realized the real work still lay ahead of us. Also, having a coauthor really helps. We certainly did not agree on everything, but whenever one person lost steam, the other person kept going. When you write alone, there is bigger risk your effort will stall.

**Bobby:** Having a coauthor doesn't mean your work drops by half, though! The workload is still 80 to 120 percent of writing the book by yourself, because you're constantly reviewing, merging, and adding to each other's work. The difference is not a lower workload but a higher-quality result. Two heads are simply better than one. And with a lot of reviewers and constant feedback, a hundred heads are even better.

Come to think of it, we wrote the book in a rather agile manner, with frequent releases where our customer gave us feedback we incorporated into subsequent releases. When our customer indicated the material was "good enough," we deployed to "production"—that is, the printer.

**Olaf and Cesare:** Thank you very much for your insights, Bobby and Gregor. Is there anything else you would like to share with us—for example, a summary of your *EIP* experience?

**Gregor:** Having a big toolbox is impressive. Knowing when to use what tool and why separates the true expert from the show-off.

**Bobby:** Patterns capture expertise that is timeless. Skills learned through patterns remain applicable even as the products and technologies evolve.

## References


1. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2003.
2. M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.
3. B. Woolf and K. Brown, "Patterns of System Integration with Enterprise Messaging," 2002; www.hillside.net/plop/plop2002/final/woolfbrown2.pdf.
4. G. Holpe, "Enterprise Integration Patterns," 2002; www.hillside.net/plop/plop2002/final/Enterprise%20Integration%20Patterns%20-%20PLoP%20Final%20Draft%203.pdf.
5. A.C. Oliver, "Long Live SOA in the Cloud Era," *Infoworld*, 21 June 2012; www.infoworld.com/article/2615838/service-oriented-architecture/long-live-soa-in-the-cloud-era.html.
6. E. Gamma et al., *Design Patterns*, Addison-Wesley, 1995.

7. C. Fehling et al., *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*, Springer, 2014.
8. D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*, 2nd ed., Prentice Hall, 2003.
9. C. Alexander et al., *A Pattern Language: Towns, Buildings, Construction*, Oxford Univ. Press, 1977.
10. G. Hohpe, "Your Coffee Shop Doesn't Use Two-Phase Commit," *IEEE Software*, vol. 22, no. 2, 2005, pp. 64–66.
11. G. Hohpe, "Google Cloud Pub/Sub," blog, 8 Apr. 2015; www.enterpriseintegrationpatterns.com/ramblings/82_googlepubsub.html.
12. G. Hohpe, "Conversation Patterns: Overview," 2015; www.enterpriseintegrationpatterns.com/patterns/conversation.

**OLAF ZIMMERMANN** is a professor and institute partner at the Institute for Software at the University of Applied Sciences of Eastern Switzerland, Rapperswil. Contact him at ozimmerm@hsr.ch.

**CESARE PAUTASSO** is an associate professor at the Faculty of Informatics at the University of Lugano. Contact him at c.pautasso@ieee.org.

**GREGOR HOHPE** is the chief IT architect at Allianz. Contact him at info@enterpriseintegrationpatterns.com.

**BOBBY WOOLF** is an IT specialist in cloud platform services at IBM. Contact him at woolf@acm.org.