INTRODUCTION
TELEPLOP – THE BEGINNING

Linda Rising

PATTERNS CONFERENCES are called PLoP™s[1] – yes, the acronym came first! It represents Pattern Languages of Programs, as defined by Ward Cunningham in preparation for the first PLoP in 1994. There are several PLoPs each year: the original at Allerton Park in Monticello, Illinois, U.S.A., EuroPLoP at Kloster Irsee near Munich, Germany, and ChiliPLoP at the Wickenburg Inn in Arizona, U.S.A. The first KoalaPLoP in Australia was held in May 2000. Originally the primary activity at these conferences was workshopping patterns, but the conferences also provide a place where pattern writers can gather to talk about ideas for patterns, pattern languages, and other pattern topics.

Dick Gabriel introduced the process of a writers' workshop to the patterns community. It's a practice used by authors to get feedback from other authors about their work. Jim Coplien (Cope) has written a pattern language for writers' workshops [Coplien99] and Jim Doble, Gerard Meszaros [Meszaros+98], and I have written about the adventure of pattern writing [Rising98]. In a nutshell, a group of writers will gather to read and comment on their work. The group sits in a circle and one author is chosen for the first session. A facilitator keeps the process on track. The author begins the session by reading a small selection from the work and then leaves the circle to become "a fly on the wall," someone who can hear the comments but make no response. The group then talks about elements of the work that shouldn't be changed – the things they liked. This is followed by suggestions for improvement. During the discussion, the author hears and takes notes but cannot respond. When all comments have been given, the author

[1]PLoP is a trademark of The Hillside Group, Inc.

returns and may ask for clarification on any comment but, again, may not respond, explain, or defend any part of the work. The reason for this is simple: If the work cannot stand on its own, the author should hear that and make any needed improvements. This is not a review in the sense of a code review where a list of defects is captured and follow-up reviews are held. The author "owns" the comments and makes adjustments as appropriate.

The PLoPs have been a success. Lots of patterns have been written and published. Lots! The troubling aspect of this success is that now the community is a bit overwhelmed by the number of patterns. I have taken a step to help with this [Rising00], but the real issue is that the focus has been on individual patterns and not on the connections between patterns or on the creation of pattern languages.

Christopher Alexander has been held up as the author of the primae facie pattern language – perhaps the **only** pattern language. For more information on Alexander and his work see http://www.math.utsa.edu/sphere/salingar/Chris.text.html#PHILOSOPHICAL. Alexander's books [Alexander77, Alexander79] describe a collection of patterns for building architecture. These patterns have strong intraconnections. They work together. Patterns depend on other patterns and lead to the use of still other patterns. It is clear when reading these patterns how they work together to build something. The software patterns community has been struggling with the way to apply this technique to our patterns. One solution led to the first ChiliPLoP in 1998. At this "different kind of PLoP" (http://www.agcs.com/patterns/chiliplop) hot topics were announced – for example, telecommunications – and experts in the hot topics attended the conference to plan for a pattern language or pattern languages in the hot topic domain.

Before the first ChiliPLoP, however, a group of telecommunications experts gathered at OOPSLA '96, where Jim Doble, Neil Harrison, and Hans Rohnert jointly proposed TelePLoP: Workshop on Patterns in Telecommunications). Their laudable goals were to:

- Enrich the still small body of publicly available telecommunications patterns.
- Help the participants learn how to write better patterns.
- Clarify costs and benefits of writing and using telecommunications patterns.
- Gain understanding regarding processes for applying patterns in the

development of architectures and designs for telecommunications systems.

- Form a group of people interested in telecommunications patterns that will transcend the workshop.

A report on that OOPSLA TelePLoP from Neil Harrison:

> At the end of the day, we talked about telecommunications patterns, and we all agreed that most of the patterns in telecomm were things that will occur in other domains sooner or later. The nature of telecommunications forced these problems on us, and we had to find solutions (the patterns) well before anybody else.
>
> As I look back, I continue to think we were right. We had to deal with high availability and reliability. Today's data networking hardware and software is only now beginning to approach the reliability that the voice networks have had for years. We had to deal with soft real-time constraints. Same thing. We have had to deal with hard memory limits (no virtual memory). James Noble and Charles Weir are just now publishing a book on patterns of limited memory systems [Noble+00], driven in part by the likes of PalmPilots. And so on.

The meeting at OOPSLA led to the hot topic at ChiliPLoP '98 and the birth of a new community. One of the "next steps" identified at ChiliPLoP was to produce a single volume of the already published telecommunications patterns. This book is that volume and can be viewed as a "fat call for papers for future volumes." If you work in the domain of telecommunications (actually this publication has been broadened to communications), please consider becoming a part of this community, participating in future TelePLoPs, and writing patterns that enrich the literature that this publication contains. Restructuring of existing patterns and pattern languages is also needed. All contributors are welcome! To subscribe to the TelePLoP mailing list, send e-mail with "subscribe" as the subject to: telecom-patterns-request@cs.uiuc.edu and then start a discussion.

Contributors to the growing collection of patterns in communications come from many different parts of the industry. Since the authors have taken the time to document their expertise, we can still benefit, even if our work is in another area. I remember when I first heard Cope talk about a pattern that the folks working on the 4ESS™ switch had documented called *Leaky Bucket Counters.* When we first heard it was a 4ESS pattern, some of us thought it was something we really wouldn't be interested in or care much about. After all, the 4ESS switch embodies old technology;

development started in the early '70s and its first service was in January 1976. As we listened to the pattern details, however, we realized that this pattern – despite the details of the 4ESS processor and the 4ESS rationale – had a much broader application. Some of us recognized that this pattern had been used on the GTD-5® and others had applied it on products at Honeywell or Motorola. This is a pattern whose context is fault-tolerant systems – bigger than even the 4ESS switch! Let our mistake in judgment help you avoid not hearing and learning from the experience of others. When you read Greg Utas's paper on call processing, for example, don't think that there won't be anything useful for you because you don't work on call processing systems or software at Nortel. Another pattern from Nortel by Gerard Meszaros, *Leaky Bucket of Credits,* used in Nortel's DMS-100, describes the same concept as *Leaky Bucket Counters* for managing distributed system architectures. This pattern is included in Gerard's chapter on reactive systems described later.

Storytelling

If Almon Strowger were alive today, he would be astounded at what has happened to the world of communications. Who is Almon Strowger? I was hoping you'd ask! It's amazing how many people working in the wondrous world of communications have never heard his name. Since I helped to write, produce, develop songs, and serve as stagehand for a production of "Strowger the Musical," at AG Communication Systems, I feel I can give you some interesting communications history. Almon Brown Strowger was an undertaker. (No, really, this is a true story.) He was born in New York in 1839. In 1886 he moved to Kansas City, Missouri, to open his own funeral parlor. In those days, when a family needed his services, they would ask the telephone operator (they were called "hello girls") to connect them, but sometimes the calls never reached Strowger. Sometimes it was because of technical difficulties – lines being busy, for example – but Strowger suspected that the operator was (ahem) conspiring with one of his competitors. His suspicions were verified when a dear friend passed away and his final needs were taken care of by the suspected competitor. This was the last straw!

Almon determined to make an "automatic" operator – something that would switch a call to the desired number without the assistance of a human interloper. Luckily his nephew, Walter, was an engineer, and together, Walter and Almon used a collar box (a round wooden box for men's collars, detachable in those days) and pins pushed into the sides. The patent for their invention was granted in 1891. In our musical, we amplified the truth a bit to say that the competitor, Henry Dahlby, had promised to marry Mabel, the operator, when his business got better. She fell for it. When the switch was invented, Mabel was downsized and Henry left her in the lurch. We had to have a happy ending, so we wrote that Walter married Mabel and Henry apologized to Almon – sappy but it made for great songs! The company founded by Almon and Walter became Automatic Electric, and eventually AG Communication Systems, today a subsidiary of Lucent Technologies.

The point of all this is that I'm sharing a good story and you're listening. That's what communication is all about – sharing good stories! Patterns are just one way of doing this. Patterns help us write down good ideas that have solved real problems. The pattern form captures enough information about this solution so that when you read it, you can use the solution to solve your problems. Your implementation of the solution might be different from mine, but the intent will be the same. When I tell you my story, you'll be able to see common threads that overlap our environments and then you can apply my ideas with your own spin. We've been doing this forever. Our brains are story based [Schank90], and a pattern is a one good way to tell a story.

Although the authors of the chapters in this publication use different pattern forms, they all share some key elements. Each pattern has a good name. This is crucial. You'll never forget "The Strowger Story." If someone comes up to you years hence and mentions the name, you'll remember how an undertaker started the communications world we live in. A good pattern with a good name does the same thing for you. The name captures the intent of the solution. Simply using the name is enough to remind the speaker and the listener of the story behind the name. If you know enough patterns and if the patterns are closely related, you can almost speak a language made up of the pattern names. This language will help you work out solutions to complicated problems and you'll be able to develop solutions faster. This works whether you're developing with just one colleague or a team of people, or even when you're talking to yourself. Powerful stuff, patterns!

In addition to the name, the problem you're trying to solve, and the solution, there should be a clear statement of the context. Patterns are proven solutions that work in the specified context. The context helps you understand when you can apply the solution to solve the problem. Changing the context – the setting or the environment – can clearly have an impact on whether the solution will work for you. In addition to the name, problem, solution, and context, the forces tell us why the problem is hard and the trade-offs we might have to make.

Other information that's also helpful includes the resulting context – what will the world be like if you apply the solution? What are possible side effects? The rationale explains why the pattern works and convinces the reader that the solution is a good one. Giving specific known uses also convinces the reader that this is not just a theoretical, pie-in-the-sky, might be a good idea solution – this is real; it's been applied and it works!

Patterns in Communications

All the chapters in this publication have previously been published. That means they're extra-credible. In addition to being workshopped, they have been reviewed and edited. The chapters describe solutions to problems in the domain of communications. Those who work outside this field might find many of these patterns also to be useful in their domains. Usually when developers read a pattern, they are happiest when they see details that mean something in their area of expertise. As a group, engineers are superb at abstraction. They go "meta" at the drop of a hat! Like the rest of the world, however, they learn best from an example, and the examples that come from their own area of interest mean the most. There are many other published patterns that are used daily in the communications area, but the patterns in this publication are clearly from that domain, use the language from that domain, and have example uses from that domain.

Gerard Meszaros is the author of Chapter 1, "Design Patterns in Telecommunications System Architecture." It appeared in a special issue of *IEEE Communications* on design patterns. This chapter does a good job of setting the stage, especially for those who might be new to patterns. Gerard provides a gentle introduction to the field of telecommunications and distributed environments. Gerard has contributed two other chapters to this publication. One describes a pattern language and the other presents a pattern that has become well known within the pattern community, even among those who do not work in this domain, Half-object + Protocol (HOPP). Gerard introduces this pattern in his first chapter, so it will seem like an old friend when you read about it later.

Large Collections

The authors of the next five chapters share their expertise by introducing collections of patterns. The idea of a pattern language, a set of patterns that work together to yield a product in a particular domain, is really the goal for pattern writers. When we start thinking about patterns and trying our hand at writing one, we think of isolated problems that we've solved successfully. When we've captured that solution, our first inclination is to say, "There! That problem is solved!" In reality, however, the result of applying one pattern is that now we face a set of new problems. These, of course, require new patterns, which, in turn, create new contexts and new problems, and require new patterns. We would like to believe that there's a stopping point, but some would say that as soon as a product is initiated, the maintenance phase begins. Problems are always with us. Related patterns that work together in a pattern language are valuable for others to study, especially newcomers to the field. Wouldn't it be wonderful if we had handbooks of pattern languages for every new person who walked in the door! These patterns would not only help the novice understand the domain but would provide the tools for communication based on the best solutions. Using the pattern names would give a meaningful introduction to ease those initial struggles.

Dennis DeBruler's Chapter 2, "A Generative Pattern Language for Distributed Processing," uses examples from cable television advertising, a video server, and telephony. His patterns address the very early stages of development. The title states that these patterns are for distributed processing, but I think you'll find that they can be useful for any kind of development. Here's what Dennis has to say about his patterns:

> In the fall of 1995, Jim Coplien gave a talk in our auditorium. The topic was a concept he called "pattern languages." The next morning I had one of my 3:00 a.m. "specials." I woke up with an idea dancing over and over in my head – a pattern language for distributed computing. I learned a long time ago, that the easiest way to get back to sleep is to get up and write the idea down. In this case, it was an e-mail to Cope.
>
> In 1996 when Cope learned about the first PLoP conference, he told me that I "had" to submit my pattern language to the conference. He said that the patterns people were "different" enough that an e-mail style would be accepted. He knew that I had not attended conferences for years because they were dominated by academics, who I feel focus on rather artificial problems. I have enough real problems to solve, that I have no need for artificial problems.
>
> Cope edited the e-mail into more traditional English. He also mentioned that

> I would not have to present the paper – that all I had to do, in fact, must do, is listen because the conference was going to try a new approach called "writers' workshops." So I decided to attend the conference, and I edited his edit of the e-mail.
>
> After the conference, I made significant revisions to what I submitted. Sometimes, I think the changes were too big.
>
> I have been very grateful that Jim talked me into attending that first PLoP. It had a nice mix of industry, consultants, and academia. The workshop discussions were also very interesting because we discussed content as well as form.

Chapter 3 is the second chapter authored by Gerard Meszaros, "Improving the Capacity of Reactive Systems." These patterns tackle problems in system capacity and throughput. Performance issues are critical in the area of communications and typically include execution time and memory constraints, but this chapter tackles just execution time. I remember when I first read the solution in *Fresh Work before Stale.* Although it makes perfect sense to me now, I can remember being disturbed to hear that service is given to newly arrived customers first, while those who have been waiting, continue to wait. It's like the long line at the supermarket when a new check stand opens and the person at the back of the line jumps over and is served right away, while those of us who have been waiting with melting frozen foods find we're still at the back of the line. The rationale for this solution makes interesting reading.

The next two chapters go well together. The first, Chapter 4 by Mike Adams, Jim Coplien, Robert Gamoke, Bob Hanmer, Fred Keeve, and Keith Nicodemus, "Fault-Tolerant Telecommunication System Patterns," describes a set of patterns that tackles a critical area of importance to communications systems. These were the first to document best practices in communication systems.

I hold these patterns dear because every time I read them I'm reminded of the first time I met Cope and Bob Hanmer. Their visit to AG Communication Systems in the fall of 1995 changed forever the way I look at patterns. In the two-day class Cope and Bob presented, we learned that several veteran developers on the 4ESS™ were due for retirement and folks at AT&T (now Lucent) were uneasy about that loss. The project managers approached Cope with the idea of extracting that expertise and capturing it as patterns. At the time of their visit, Cope and Bob had documented over 100 patterns, "mined" through a process of taping interviews with the experts and extracting patterns from the recorded conversations. A small subset of that collection was presented at the second PLoP Conference in 1995 and is reproduced here. You can read more about the topic of pattern mining in a paper by David DeLano [DeLano98] and in a chapter of a recent collection [Rising99].

Software development at AG Communication Systems falls into one of two arenas. In one is a large central office telecommunications switch, the GTD-5®, developed for GTE and independent telephone companies. It contains over 3 million lines of Pascal and continues to perform well in terms of quality, reliability, and revenue. The other, non-GTD-5 projects are smaller and written, for the most part, in C++. The focus of all the patterns activity up to the time of Cope and Bob's visit had been almost exclusively directed toward the object-oriented projects. After their visit, we expanded our patterns horizon to encompass our entire software development community. Indeed, we began to think about non-software patterns – patterns concerned with the organization of teams, for example. The world is full of patterns!

Chapter 5, a close companion of Chapter 4, was written by Bob Hanmer and Greg Stymfal, "An Input and Output Pattern Language: Lessons from Telecommunications." I asked Bob and Greg how they got together to write this paper and about the name – many patterns have memorable names – *George Washington Is Still Dead!*

From Bob Hanmer:

> During a break in a class I taught about the 4ESS Switch at AG Communication Systems, Greg Stymfal and I started talking. He commented that he had worked on a system that did I/O in the same way. One thing led to another and so we collaborated on the pattern language that we submitted to PLoP.
>
> Pattern naming is important. The name should give the reader some sense of the pattern. *George Washington Is Still Dead* does not satisfy this requirement, unless you are from Phoenix. There it seems to be a regional idiom meaning "nothing has changed, don't ask me again."

From Greg Stymfal:

> The name *George Washington Is Still Dead* actually has a story behind it. On the early "Saturday Night Live" shows, Chevy Chase would have as one of his news items: "General Francisco Franco is still dead." This was the catch phrase for old news.
>
> Because this might be offensive or misunderstood, it was changed to *George Washington Is Still Dead.* If it has now lost all meaning or is not evident from the name, the pattern could be renamed to something like: *Yesterday's News.*

It's a little unusual for people like Bob and Greg, who are from two different companies, to collaborate on a paper about patterns. Since pattern authors are sharing their experience in a domain, that experience is usually company-specific. In fact, many companies regard this information as proprietary – even when that information may be known across the industry. There's something about documenting the details of a solution used in a particular product that ruffles the feathers of those who guard a company's intellectual property. On the one hand, it's easy to sympathize with a company that wants to protect its own best interests; but on the other hand, most patterns are documenting best practices that are probably well known but not necessarily by the pattern name. Greg Utas recalls an example of this from the first TelePLoP:

> At one point someone from Lucent, fearful that he was about to give something away, whispered to a colleague, "Should we mention corrective audits?" I overheard this and interrupted with, "You mean software that recovers resources that are marked in-use but that no longer have valid owners, or software that looks for corrupted queues and fixes them?" It immediately became obvious that both Lucent and Nortel used this technique in the quest for continuous availability.

Maybe this book can help identify the commonality in our business and help us recognize what we share.

Chapter 6 is a coalescence of two papers previously published by Greg Utas. He has done some restructuring and rewriting – we could say, "refactoring" – of the two earlier papers to produce a more complete presentation of his Pattern Language of Call Processing. One of the benefits of this publication is that papers that might have been judged too long for a single publication or conference are appropriate here. This provides Greg and other authors the chance to share an entire collection with us, regardless of its length. Greg's is a very detailed pattern language. Many of the solutions are specific to the area of call processing. Call processing experts and others will benefit from reading this language, even though Greg says that things happened backwards:

> The patterns were written to document an application framework that was developed for a large reengineering project. The feature interaction patterns at the *end* of the paper were workshopped by e-mail and published *first,* since there was a suitable conference to which they could be submitted. The basic patterns were elaborated later and workshopped at the 1999 TelePLoP. A subset of these was published when *IEEE Communications* devoted an issue to patterns in communication systems, so these patterns appeared as a *prequel.*