A Pattern Language for Pattern Writing

Gerard Meszaros
ClearStream Consulting

Jim Doble
Tavve Software Company

Abstract

As the patterns community has accumulated experience in writing and reviewing patterns and
pattern languages, we have begun to develop insight into pattern-writing techniques and
approaches that have been observed to be particularly effective at addressing certain recurring
problems. This pattern language attempts to capture some of these "best practices" of pattern
writing, both by describing them in pattern form, and by demonstrating them in action. As such,
this pattern language is its own *Running Example*.

1.0 Introduction

The use of patterns to communicate wisdom and insight in computer/software systems design is a
relatively new idea. As such, techniques and approaches for writing patterns and pattern languages
are continually being improved, as creative individuals try new ways to organize and communicate
their thoughts. Although there is no single right way to write patterns, this pattern language
describes and demonstrates a collection of writing practices that have been observed to be
particularly effective. The language is targeted at both novice and experienced pattern writers:
novices may choose to treat these patterns as suggestions to be tried and to be adopted where they
help, experts can use these patterns as a form of checklist, helping them keep in mind some of the
issues and forces in effective pattern writing.

Unlike a prescriptive pattern language, which describes the steps or recipes for solving some
problem, this pattern language describes the desired result. The pattern author is free to employ
different techniques to achieve these results. This approach should allow this pattern language to
be employed in whole or in part as the reader sees fit.

History of This Pattern Language

Most of the patterns in this language started out as observations about "things that worked well" in
a particular pattern or language being reviewed in a PLoP-95 writers' workshop. As PLoP-95
progressed, these observations led to hypotheses that certain of these techniques and approaches
would be particularly effective at addressing recurring pattern-writing problems. These hypotheses
were tested in subsequent pattern reviews; if the hypotheses were shown to be true more often than
not, we started referring to them by evocative names. (In the beginning, the names were all we had
to communicate them, so they had to be evocative to be effective.) Comparing notes based on our
respective review groups' experiences, we began to observe that the kinds of practices that were
being applauded were very much the same despite the independence of our review groups. It was
quickly recognized that these techniques and approaches could themselves be expressed in the

Figure 1: Pattern Language Structure
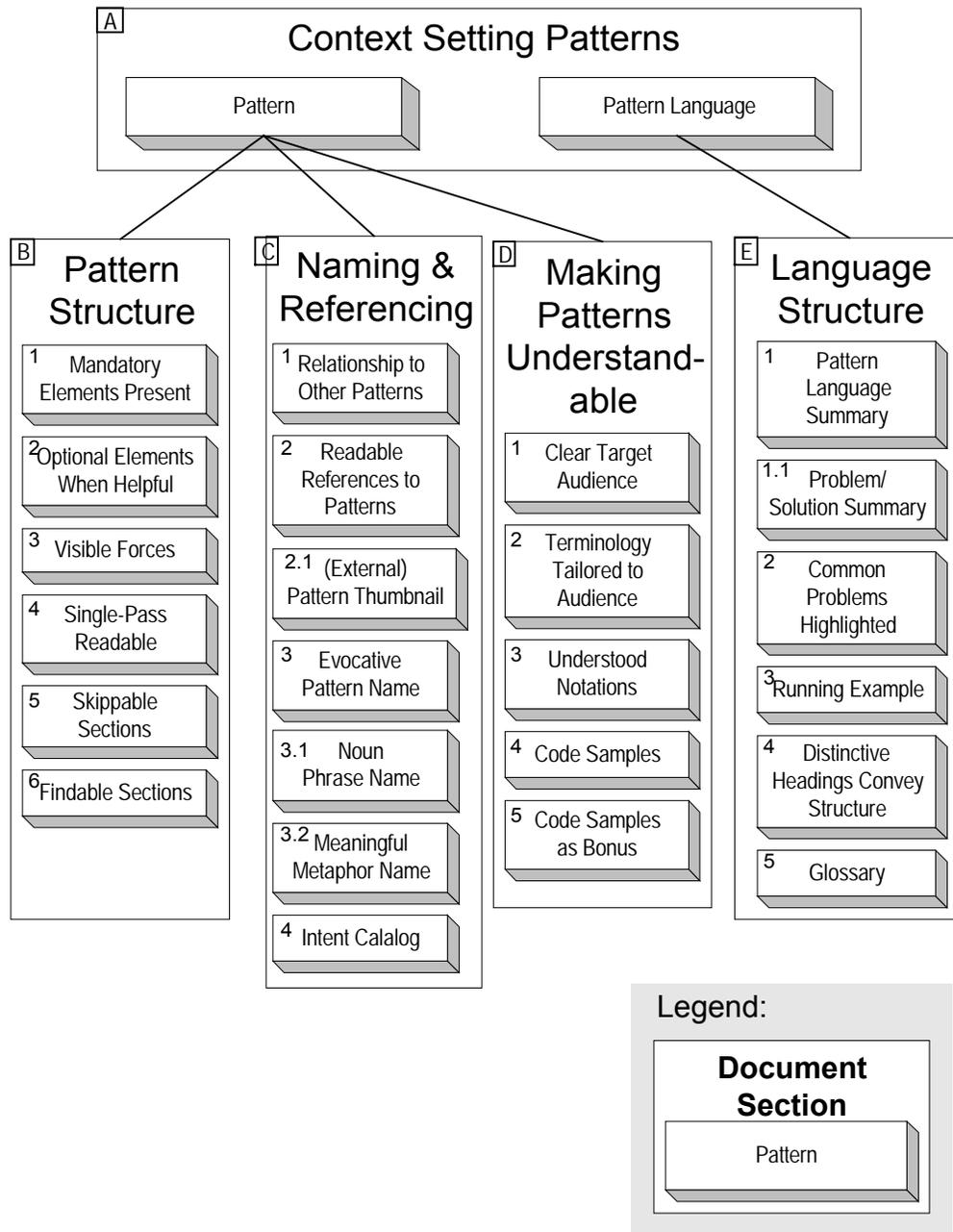
form of a pattern language[1].

While we did not keep detailed statistics, we only present patterns here that emerged from this broad consensus. In the months that followed PLoP'95, many of these patterns were made available to practitioners conducting pattern writing courses, and the feedback to the authors indicates that they have been very helpful. In addition, these patterns were reviewed extensively at

---

[1] Clearly the result of over-exposure to pattern languages. After two intensive days of reviewing pattern languages, just about any collection of scatter-brained ideas begins to look like a candidate for a pattern language.

PLoP-96 and became part of the working vocabulary of several review groups.

1.1 Pattern Language Summary

The patterns in this language are grouped into five sections lettered A through E.

- Section A, **Context-Setting Patterns,** introduces the concept of a pattern (a Solution to a Problem in a Context) and a pattern language (collections of patterns that are related to each other by virtue of solving the same problems or parts of a solution to a larger, partitioned problem) so that they may be used throughout this pattern language.

- Section B, **Pattern Structuring Patterns,** contains patterns describing the desired content and structure of individual patterns, whether free-standing or part of a larger pattern language.

- Section C, **Pattern Naming and Referencing Patterns**, contains patterns that describe techniques for naming your pattern(s) and for including references to other patterns within your pattern(s).

- Section D, **Patterns for making Patterns Understandable** contain patterns that capture techniques for making your patterns and pattern languages easier to read, understand and apply.

- Section E, **Pattern Language Structuring Patterns,** contains patterns that describe the desired content and structure of pattern languages.

Each section starts with a brief summary that introduces the patterns described in the section. The patterns in the entire collection are depicted graphically in Figure 1 and summarized at the end of this paper in the *Problem/Solution Summary* section in tables 1-4.

1.2   How to Use These Patterns

A reader searching for a solution to a particular pattern writing problem should refer to the Problem/Solution Summary tables to see if any of the problems resemble the one they are trying to solve. Since each pattern has been written with *Skippable Sections*; a reader could flip through the language looking at the **Name, Problem** and **Solution** sections. Once they have determined that a pattern is of interest, they can look at the **Context** and **Forces** section for guidance on determining whether it is applicable to their situation. Finally, they can look at the **Rationale**, **Resulting Context**, **Related Patterns** and **Examples** sections to get further appreciation of the nuances of the pattern.

1.3   Conventions

Throughout this pattern language, pattern names are indicated in *italics* and pattern terms are indicated in **bold**.

A: Context-Setting Patterns

It is not the primary purpose of this pattern language to define the concept of a pattern or a pattern language. However, since the patterns in this language are applied within the context of writing patterns or pattern languages, we must include some sort of working definition. In keeping with the spirit of the patterns movement, we do this in the pattern form.

A.1 Pattern: Pattern

Context:

You are an experienced practitioner in your field. You have noticed that you keep using a certain **solution** to a commonly occurring **problem**. You would like to share your experience with others.

Problem:

How do you share a recurring **solution** to a **problem** with others so that it may be reused?

Forces:

- n   Keeping the **solution** to yourself doesn't require any effort

- n   Sharing the **solution** verbally helps a few others but won't make a big impact in your field.

- n   Writing down your understanding of the **solution** is hard work and requires much reflection on how you solve the **problem**.

- n   Transforming your specific **solution** into a more widely applicable **solution** is difficult.

- n   People are unlikely to use a **solution** if you don't explain the reasons for using it.

- n   Writing down the **solution** may compromise your competitive advantage (either personal or corporate.)

Solution:

Write down the **solution** using the pattern form. Capture both the **problem** and the **solution**, as well as the reasons why the **solution** is applicable. Apply *Mandatory Elements Present* to ensure that the necessary information is communicated clearly. Include *Optional Elements When Helpful* to capture any additional useful information. Distribute the resulting pattern to the largest audience you feel it could help that does not compromise your competitive advantage. Often, this means publishing your patterns exclusively within your company via Intranets or company journals.

A.2 Pattern: Pattern Language

Context:

You are trying to use the "pattern form" to describe a procedure with many steps or a complex **solution** to a complex **problem**. Some of the steps may only apply in particular circumstances. There may be alternate **solutions** to parts of the **problem** depending on the circumstances. A single pattern is insufficient to deal with the complexity at hand.

Problem:

How do you describe the **solution** such that it is easy to digest and easy to use parts of the **solution** in different circumstances?

Forces:

- n   A single large **solution** may be too specific to the circumstance and impossible to reuse in other circumstances.

- A complex **solution** may be hard to describe in a single pattern. A "divide and conquer" approach may be necessary to make the **solution** tractable.

- Factoring the **solution** into a set of reusable steps can be very difficult. Once factored, the resulting pieces may depend on one another to make any sense.

- Other pattern languages may want to refer to parts of the **solution**; they require some sort of "handle" for each of the parts to be referenced.

Solution:

Factor the overall **problem** and its complex **solution** or procedure into a number of related **problems** with their respective **solutions**. Capture each **problem**/**solution** pair as a pattern within a larger pattern language. Each pattern should solve a specific **problem** within the shared context of the language. Strive to ensure that each pattern could conceivably be used alone or with a limited number of patterns from the language.

To give the pattern language an identity of it's own, give it an *Evocative Name* by which it can be known and referenced. Describe the overall **problem** and how the patterns work together to solve it in a *Pattern Language Summary*. Relate the patterns to each other *using Readable References to Patterns* within the pattern description, especially in the **Context** and **Related Patterns** elements.

Example:

This pattern language is itself an example of tackling the complex problem of writing patterns and pattern languages. It presents the **solution** as a number of patterns each of which describe the **solution** to a specific smaller **problem**.

B.   Pattern Structure Patterns

A pattern is just "a description of a **solution** to a **problem** found to occur in a specific **context**." But many other types of writing would claim to satisfy this phrase. What sets patterns apart is their ability to explain the **rationale** for using the solution (the "why") in addition describing the **solution** (the "how"). A key contributor to this characteristic is the structure of the pattern form.

The patterns in this section describe the structure of an individual pattern*,* whether free-standing or part of a larger pattern language. Patterns are easier to understand and apply when all *Mandatory Elements (are) Present* regardless of the pattern style chosen. *Optional Elements When Helpful* gives pattern writers considerable flexibility in what additional information they present and how they structure it to maximize the readers' understanding. While structuring your pattern, strive to make it *Single-Pass Readable* to minimize the frustration of the pattern reader. Some techniques for achieving this include having *Visible Forces* so they can be easily picked out, and *Skippable Sections* that can be bypassed to speed up the first reading.

B.1   Pattern: Mandatory Elements Present

Aliases: All Elements Present

Problem:

How do you make sure that all necessary information is covered in a pattern?

Context:

You are writing a pattern, either standalone or as part of a pattern language.

Forces:

- n   All patterns do not require the same kinds of information to be effectively communicated. Capturing all elements regardless of need only clutters many patterns.

- n   For a pattern to be truly useful, it must have a minimum set of essential information. These information elements are required to allow patterns to be found when required and to be applied when applicable.

- n   If the necessary elements are missing, it becomes much harder to determine whether the pattern solves the reader's problem in an acceptable way.

- n   There is no single correct style or template for patterns; trying to impose one could stifle creativity and get in the way of effective communication.

Readers expect certain information to be present in a pattern. This is what differentiates a pattern from a mere problem/solution description.

Solution:

Include the following Mandatory Elements in the pattern. The exact names of these elements vary from one pattern style to another and the exact order in which they appear in is not as crucial as ensuring that they are all present. They are presented here in an order chosen to facilitate understanding of their relationships. The nature of the relationships between the elements is illustrated in Figure 2.

Figure 2: Relationships Between Pattern Elements
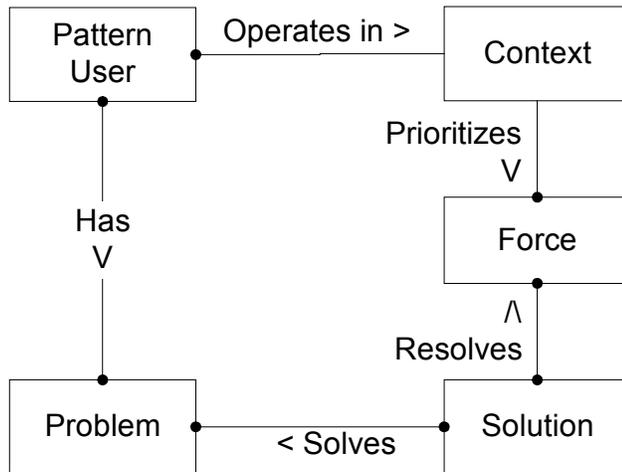
Pattern Name: A name by which this problem/solution pairing can be referenced.

Context: The circumstance in which the problem is being solved imposes constraints on the solution. The **context** is often described via a "situation" rather than stated explicitly. Sometimes, the **context** is described in terms of the patterns that have already been applied. The relative importance of the **forces** (those that need to be optimized at the expense of others) is determined by the **context**.

Problem: The specific **problem** that needs to be solved. Use *Context-Free Problem* [2] to ensure that the **problem** is kept separate from the constraints on the **solution**.

Forces: The often contradictory considerations that must be taken into account when choosing a **solution** to a **problem**. The relative importance of the **forces** (those that need to be optimized at the expense of others) is implied by the **context**.

Solution: The proposed **solution** to the **problem**. Note that many **problems** may have more than one **solution**, and the "goodness" of a **solution** to a **problem** is affected by the **context** in which the **problem** occurs. Each **solution** takes certain **forces** into account. It resolves some **forces** at the expense of others. It may even totally ignore some **forces.** The most appropriate **solution** to a **problem** in a is the one that best resolves the highest priority **forces** as determined by the particular **context**. Use *Solution Clearly Related to Forces*[3] to ensure that the reader understands why this **solution** was chosen.

Rationale

A pattern goes beyond a mere description of the **solution** by providing a window on the thought processes behind choosing the **solution**. The mandatory pattern elements described here are essential to communication of this information. In the many patterns that have been written since **The Timeless Way of Building** [Alexander79] and **A Pattern Language** [Alexander77]

---

[2]*Context-Free Problem* [yet to be written]: The problem statement should be independent of the Context. This allows other patterns to be written that solve the same problem in different circumstances. In [Roberts96], the same problem appears twice, with two different contexts and two different solutions.

[3] *Solution Clearly Related to Forces* [yet to be written]: The solution section of a pattern should clearly describe how the **solution** optimizes the **forces** that the **context** implies should be given the highest priority. This ensures that the reader understands how the **forces** influenced the selection of this **solution**.

were first published, these mandatory elements have been found to be the minimum information required to effectively communicate a pattern.

Examples

All the patterns in this language have *Mandatory Elements Present*. This ensures that potential users of these patterns understand why and when to apply them. The elements are highlighted through the use of headings. Most of these patterns start with the **Problem** statement followed by the **Context**, while others start with the **Context**. This was done to illustrate both styles. [Berczuk96] consistently places the **Context** before the **Problem** section.

In some pattern styles, the pattern elements have different names or are organized differently. See [Copelien96] for a more complete list of pattern styles and their features.

Christopher Alexander and his associates used this basic structure in **A Pattern Language**. The mandatory elements are separated typographically, with the **solution** paragraph(s) being introduced with a "therefore".

In **Design Patterns** [GHJV94], the **Problem** and **Context** sections are replaced with an **Intent** and an **Applicability** section, respectively which are augmented by a more concrete example of the **problem** in the **Motivation** section. The **Solution** section is replaced by 4 sections: **Structure**, **Participants**, **Collaborations** and **Implementation**.

B.2   Pattern: Optional Elements When Helpful

Problem:

How do you communicate essential information that does not fit well into the mandatory elements?

Context:

You are writing a pattern and have applied *Mandatory Elements Present*.

Forces:

- All patterns do not require the same kinds of information to be effectively communicated.
- Capturing all elements regardless of need only clutters many patterns.

Solution:

The following sections may be included if they make the pattern easier to understand or provide better linkage between the pattern in question and related patterns:

Indications: The symptoms that might indicate that the problem exists.

Resulting Context: The context that we find ourselves in after the pattern has been applied. It can include one or more new problems to solve. This sets us up for applying more patterns, possibly the next pattern(s) in a language.

Related Patterns: Other patterns that may be of interest to the reader. The kinds of patterns include:

- Other solutions to the same problem,
- More general or (possibly domain) specific variations of the pattern,
- Patterns that solve some of the problems in the resulting context (set up by this pattern)

Examples: Concrete examples that illustrate the application of the pattern.

Code Samples: Sample code showing how to implement the pattern.

Rationale: An explanation of why this solution is most appropriate for the stated problem within this context.

Aliases: Other names by which this pattern might be known.

Acknowledgments: You should acknowledge anyone who contributed significantly to the development of the pattern (or language) or the techniques described in it. If your pattern has been through a "shepherding process" or "writer's workshop", significant contributors (such as the shepherd!) are candidates for being acknowledged.

Examples:

In **Design Patterns**, the **Resulting Context** is known as **Consequences** and **Code Samples** is called **Sample Code**. **Examples** is called **Known Uses** and is augmented by a more concrete representation of the problem in the **Motivation** section.

[Cockburn96] introduced the idea of an **Indications** element, there called **Symptoms**.

B.3   Pattern: Visible Forces

Problem:

A pattern presents a **solution** to a **problem** within a **context**. How do you ensure that the reader understands the choice of **solution**?

Context:

You are writing a pattern or pattern language that is intended to convey one of potentially several **solutions** to a **problem**. You have applied *Mandatory Elements Present*; you are now writing the **Forces** section.

Forces:

- n   There are many different styles of patterns, some more structured than others.

- n   People like to have convenient handles for concepts such as the **forces** that affect the choice of **solution**.

- n   Prose pattern descriptions can be very pleasing to read but may be hard to use as a reference because the **forces** are buried in the prose.

- n   Having a separate **Forces** heading makes the **forces** very easy to find but may make the pattern less pleasing to read.

- n   Too much structure can impinge on the literary quality of a pattern.

Solution:

Regardless of the style chosen for the pattern description, ensure that the **forces** are highly visible. This can be done by defining a meaningful "name" for each **force** and visually setting if off from text by making them minor headings, or by highlighting them using fonts, underlining, or other typographic techniques.

Example

This pattern language uses a "bullet list" within a **Forces** heading to make the **forces** visible.

[Foote96] highlights the **forces** within the prose of the pattern description.

B.4   Pattern: Single-Pass Readable

Problem:

A person in search of a solution may need to look at many potential solutions. How do you help the reader understand your pattern in the least amount of time, in order to facilitate this search?

Context:

You are writing a pattern with *Mandatory Elements Present*.

Forces:

- **n** People sometimes only have a limited time to read a pattern.

- **n** People get frustrated and give up when the effort is too high.

- **n** A pattern that must be read several times before being understood is more likely to be misunderstood.

- **n** A simple message is more likely to be understood correctly.

Solution:

Single-Pass Readable is easier said than done, and probably merits a pattern language on its own. However, here are a variety of techniques that can be helpful to achieve single-pass readability:

- **n** Use *Evocative Pattern Names* or *Pattern Thumbnails*[4] in cases where some understanding of a forward referenced pattern is necessary for the reader to keep reading.

- **n** Help the reader locate key information by using *Findable Sections* and *Visible Forces* to highlight the tradeoffs involved.

- **n** Use *Skippable Sections* (such as *Code Samples as Bonus*) to highlight information that can be skipped on first reading.

- **n** In a pattern language, provide a clear, concise *Pattern Language Summary* outlining the structure of the pattern language, then remind the readers where they are within the structure as they go along, using *(Distinctive) Headings Convey Structure*.

- **n** If you need to introduce and/or define a number of concepts or terms in the introductory sections of your pattern, try to pare down your list by selecting only the most essential concepts and terms, and write your pattern using this reduced list. Remember that normal adults are able to keep seven (plus or minus two) items of information within their short-term memory [Miller56]. If you introduce more information than your reader can remember, he/she will need to keep referring back to the beginning, which defeats single-pass readability.

- **n** Provide a *Glossary* so that readers don't have to search around for definitions of terms that they can't remember. This isn't *Single-Pass Readable* per se, but it is preferable to the alternative.

Rationale:

In order to help the reader get what they need from your patterns in the minimum amount of time, you need to help them to read only the sections they need, only once. *Skippable Sections* and *Findable Sections* will help readers to find the sections they need. The techniques described above help the reader avoid going back (or forward) to read sections more than once.

---

[4] A brief footnote summarizing the main idea of a Pattern. Just like this one.

Related Patterns:

- *Pattern Language Summary* introduces the larger **problem** being solved and how the **solution** has been factored into a number of patterns.

- *(Distinctive) Headings Convey Structure* helps the reader understand the structure of the pattern language by reminding them where they are in the pattern language.

- *Skippable Sections* and *Findable Sections* help the user quickly find only the sections they need to understand the essence of the pattern solution.

- *Evocative Pattern Names* and *Pattern Thumbnails* reduce the need to follow pattern references before proceeding.

- *Glossary* defines the terminology in one place so the reader doesn't have to scan the pattern (language) looking for it.

B.5   Pattern: Skippable Sections

Context:

You are writing a pattern that is part of a collection intended to be used as a reference. You have applied *Mandatory Elements Present* and *Optional Elements When Helpful. You* are striving to make the pattern easily understood and *Single-Pass Readable*.

Problem:

How do you make it easy for the reader to get the essence of a pattern while still providing enough information to apply it?

Forces:

- The information required to determine whether a pattern is applicable may be a small subset of the information required to actually apply the pattern.

- People require different amounts of information to understand and apply a pattern.

- People sometimes only have a limited time to invest in reading a pattern.

- A long-winded pattern description may cause a reader to skip the pattern entirely because the expected return does not justify the investment.

- Separating information into sections may make a pattern more bulky.

Solution:

Clearly identify the **Problem**, **Context** and **Solution** parts so that the reader can quickly determine whether this pattern applies to them. Put more detailed information (such as **forces** or **code examples**) in clearly identified sections that may be skipped if a person doesn't want all the detail.

Rationale:

When the reader is trying to become familiar with a set of patterns, they often want to "cut to the chase" quickly. Too much information gets in the way. Much of the information in a pattern is only required once you have narrowed down the list, or have decided to use the pattern. This pattern makes it easier to get the essence of a pattern without being bogged down in detail, thus allowing the reader to assimilate more patterns in a shorter period of time.

Resulting Context

Unless the *Skippable Sections* are at the end, the reader may need to scan for the beginning of the next section of interest. Use *Findable Sections* to make this easier.

Related Patterns

The pattern *Code Examples as Bonus* describes a special case of a *Skippable Section*. *Optional*

Elements When Helpful describes when to include a section while *Skippable Sections* focuses on helping the reader read a pattern efficiently

Examples:

The "Alexandrian" pattern style uses fonts and *** delimited paragraphs to allow the reader to pick out the **problem** and **solution** sections. More structured styles (such as used in this pattern language) use headings to separate the different sections. The introduction of this pattern language tells the reader which pattern elements they should concentrate on for a "quick read".

In **Design Patterns**, the **Applicability** section allows the rest of the pattern to be skipped when searching for a pattern to solve a particular **problem**.

B.6   Pattern: Findable Sections

Context:

You are writing a pattern that is part of a collection intended to be used as a reference. You have applied *Mandatory Elements Present, Optional Elements When Helpful* and *Skippable Sections*. You are striving to make the pattern easily understood and *Single-Pass Readable*, and usable as reference material.

Problem:

How do you make it easy to find key elements of the pattern, most notably the **Problem**, **Context**, **Forces** and **Solutions** elements?

Forces:

- The information required to determine whether a pattern is applicable may be a small subset of the information required to actually apply the pattern.

- People require different amounts of information to understand and apply a pattern.

- People sometimes only have a limited time to invest in reading a pattern.

- A long-winded pattern description may cause a reader to skip the pattern entirely because the expected return does not justify the investment.

- If a section is skipped, it may be hard to determine where to restart reading.

Solution:

For your pattern style, determine which sections a reader may be specifically looking for when using the material as reference. Clearly identify the beginnings of each of these sections so that the reader may find them easily. This can be done typographically (using fonts, underlining, etc), using headings, or graphically (using diagrams, *'s, etc. between sections).

The start of a pattern is a special case of a *Findable Section*. Techniques to help find the reader find the start of patterns include starting all patterns on a new page, shaded headers, and evocative illustrations at the start of patterns.

Rationale:

A section is only truly skippable if the next section of interest is easily found without reading or skimming the section to be skipped. The more visible the section demarcations, the easier it is to skip directly to them.

Related Patterns

Skippable Sections focuses on helping the reader read a pattern efficiently while this pattern improves the usability of the pattern as reference material**.**

Examples:

The "Alexandrian" pattern style uses fonts and *** delimited paragraphs to allow the reader to

pick out the problem and solution sections. More structured styles (such as used in this pattern language) use headings to separate the different sections. In some pattern languages, it is often possible to key elements by merely lifting ones finger and flipping the page. This is because all patterns start on a new page and each pattern element starts at roughly the same point on the page.

[Foote96] makes effective use of a combination of shaded headers and evocative illustrations to make the start of individual patterns easy to find.

C. Pattern Naming and Referencing Patterns

Few patterns live in isolation. Typically, they introduce new, hopefully smaller and more tractable problems that will lead you to other patterns. Or, there may be other patterns that solve the same problem. The patterns in this section all deal with how to describe these *Relationships to Other Patterns* in a pleasing and efficient manner. Many elements of a pattern may need to refer to other patterns. Including *Readable References to* patterns within the prose makes these references informative without being intrusive. Naming your patterns with *Evocative Pattern Names* makes it easier to refer to them and reduces the need for the reader to follow the references every time you have included a pattern name. Two especially useful patterns for creating *Evocative Pattern Names* are *Noun Phrase Name* that names the pattern after the resulting solution, and *Meaningful Metaphor Name*, which names the pattern after some commonly understood metaphor. When the reader may need more information about the pattern than just the name, include a *Pattern Thumbnail*. A pattern's *Intent Catalog* is a good source for *Pattern Thumbnails*.

C.1   Pattern: Relationship to Other Patterns

Context:

You are writing the **Related Patterns** element of a pattern with *Mandatory Elements Present*.

Problem:

How do you make a pattern part of a larger group of patterns?

Forces:

- Few patterns are truly isolated; they usually lead to other patterns or they solve problems set up by other patterns.

- Patterns are more useful if their relationships to other patterns are documented.

- Determining and describing the related patterns can be hard work.

- Too many references to patterns may distract the reader from the solution you are trying to describe.

- It may be hard to find all the related patterns.

Solution:

One of the key advantages of a pattern language over a standalone pattern is its ability to guide the reader to the solution of a complex problem by leading them from one pattern to another. Stand-alone patterns have to work harder to establish their relationships.

These relationships can take many forms. The pattern being written may:

- *Lead to* other patterns, often within a pattern language, by creating a **problem** that a subsequent pattern solves. The **problem** introduced (and possibly the pattern to solve it) should be described in the **Resulting Context** or **Solution** section.

- The pattern may be *set up* by other patterns that introduce the **problem** this pattern solves. Capture the preceding patterns in the **Context** section.

- The pattern may *Specialize* a more general version of a pattern to make it more easily applied within a specific problem domain, or

- Or it may *Generalize* one or more domain specific patterns to allow more general applicability. Capture these in the Related Pattern sections.

The pattern may have *alternative* patterns that solve the same **problem** in different ways. Capture these in the **Related Pattern** sections.

The pattern description should point out similarities or differences from other patterns that might

seem, on the surface, to be the same. Capture these in the **Related Pattern** sections.

A pattern addressing a **problem** in one domain may be complemented by a pattern in an orthogonal domain.

While thinking about or writing a pattern, read other pattern languages and identify relationships to other patterns. Refer to related patterns as appropriate throughout the pattern, most notably in the **Context**, **Solution**, **Resulting Context** and **Related Patterns** sections. Use *Readable References to Patterns* to cite the related patterns. Where necessary, include a *Pattern Thumbnail* so that your reader doesn't have to look up the pattern to understand how it is related.

To make your pattern(s) easier to refer to from other patterns, give your patterns *Evocative Pattern Names* and summarize their intent(s) in an *Intent Catalog* or *Problem/Solution Summary*.

Rationale:

Understanding the relationships between a pattern and other patterns makes a pattern more understandable and useful since alternative **solutions** can be assessed and follow-on patterns can be found and applied.

Examples

[Berczuk96] does a particularly good job of relating technical patterns to organization patterns.

Context:

You are writing a pattern structured according to *Mandatory Elements Present*. You need to refer to other patterns in one or more of the elements.

Problem:

How do you refer to other patterns within the description of your pattern?

Forces:

- A pattern may be described much more concisely if it can delegate much of its **solution** to other patterns.

- Citations to other patterns could disrupt the reader's train of thought or cause them to lose their **context**.

- The reader may want to read the pattern being referred to and will require more information than just the name.

Solution:

When referring to patterns within the body of your pattern, weave the pattern names into the narrative. Augment the pattern name with a pattern reference that can be used to look it up. Set off the pattern name from the surrounding text by highlighting it typographically.

Where the patterns have *Noun Phrase Names*, you should be able to use the pattern name directly in the sentence (as we have done in this sentence.) In most cases*, Meaningful Metaphor Names* can be treated in the same way. *Verb Phrase Names*[5] are somewhat more difficult to weave into the narrative because they are typically "imperative" or "prescriptive" in nature; it may be harder to use them to describe the result.

To make it easier for the reader to find the description of the pattern, it is desirable to include a reference to the pattern description. This could take the form of a traditional literary reference of the form pattern-name[reference-name], or you can use an *External Pattern Thumbnail*.

Rationale:

Weaving the pattern names into the text makes the pattern easier and more pleasurable to read while the reference satisfies those readers who want to find the original.

Related Patterns:

External Pattern Thumbnails can be used to refer to patterns that must be understood to get the essence of the current pattern.

Example:

In this pattern language, we have used a special character style for pattern names to distinguish

---

[5]*Verb Phrase Name* [not included here]: name the Pattern after the process used to create the solution. Example: *Verb Phrase Name* is itself a Noun Phrase Name. To recast this as a *Verb Phrase Name*, you could try: *Name the Pattern After The Process Used to Create the Solution.*

them from the surrounding text. Because most of the pattern referenced have *Noun Phrase Names*, we have been able to weave the pattern names into the text. The names have been augmented with a footnote or an internal pattern reference number. We have included an *External Pattern Thumbnail* for any patterns that are not included within the language.

C.2.1   Pattern: (External) Pattern Thumbnail

Context:

You are writing a pattern that makes reference to related patterns that appear later or are not included within the pattern or pattern language you are writing. A basic understanding of these patterns may be necessary for the reader to fully understand your pattern.

Problem:

How do you refer to other patterns in a concise but meaningful manner with minimum interruption of the reader's "flow", so that the understanding of your pattern is maximized?

Forces:

n   Referencing external patterns is an effective way to build upon, or relate your work to existing patterns and pattern languages.

n   Including the complete description of related external patterns within your pattern will make it too large, and will distract the reader from what you are trying to communicate.

n   A basic understanding of related external patterns may be necessary for the reader to fully understand your pattern.

n   Some readers will be familiar with the referenced external patterns, while others will not.

n   The most concise way to refer to an existing pattern is to provide sufficient information for the reader to obtain the complete description of the referenced pattern, typically using an author/year tag and a References section.

n   A reader may not have the time (or energy) to obtain and read the complete description of the referenced pattern, prior to finishing reading and understanding your pattern. Even if the reader does this they will be significantly distracted from your pattern.

Solution:

Include the *Evocative Name* of the external pattern within the text. The first time the external pattern is referenced, provide both an author/year tag and a footnote with a brief (one or two sentences) "thumbnail" description of the essence of the external pattern. The thumbnail should provide just enough information about the external pattern to maximize understandability of your pattern.

As an alternative to using footnotes, you can include the thumbnail and reference in the body of your text (in parentheses).

Rationale:

Readers who are familiar with the external referenced pattern should not be distracted by the thumbnail footnote. Other readers will be able to continue reading and understanding your pattern after they have read the thumbnail. Readers who want to understand the referenced pattern in detail should be able to obtain its complete description using the information in the References section.

Related Patterns:

n   *Evocative Pattern Name* helps reduce the need for *Pattern Thumbnails*.

Examples:

Several of the patterns in this language include *External Pattern Thumbnails*. For example, *Evocative Pattern Name* contains a thumbnail reference to *Buffalo Mountain*[7].

Episodes [Cunningham96] includes a table of *External Pattern Thumbnails* as an appendix to help the reader understand the essential aspects of yet unpublished patterns.

C.3   Pattern: Evocative[6] Pattern Name

Aliases: *Understandable Pattern Name, Solution Revealing Pattern Name*

Context:

You are writing a pattern (or pattern language) that may need to be referred to by other patterns or pattern languages.

Problem:

How do you name a pattern so that it is easy to remember and refer to?

Forces:

- Patterns may vary based on differences in **problem**, **context**, **forces**, **solutions**, etc. or any combination of these. Each combination may require a distinct pattern name.

- People should be able to use patterns as a vocabulary, i.e., the identity of the pattern becomes a "word" in a person's design vocabulary.

- A name short enough to use as a noun in a sentence may not convey enough meaning to be understandable out of context.

- The most memorable patterns are those who names conjure up a clear image of the solution.

- Cute but obtuse pattern names may be meaningful to the writer but few readers will remember what they mean later.

Solution:

Choose a pattern name that is likely to conjure up images that convey the essence of the pattern solution to the target audience. Imagine using the name in conversations or referring to it from other patterns. Test the name by having people unfamiliar with the pattern description guess at what the pattern might be about based only on the pattern name.

Rationale:

As patterns are used to construct systems or to express how they are related to one another, the name is used without the accompanying description. A name chosen using this pattern is more likely to be understood and that makes it more likely to become part of the vocabulary of the readers.

Related Patterns

Intent Revealing Method Selector in [Beck96] describes the solution to the problem of naming methods in Smalltalk programming. By capturing the intent in the method name, the reader of the program should not have to refer to the method description every time they see the name.

Two patterns for creating *Evocative Pattern Names* are: *Noun Phrase Name* and *Meaningful Metaphor Name*. *Buffalo Mountain*[7] is an example of a cute but obtuse name for a pattern that many people remember but few can recall what it describes.

---

[6] Evocative: adj. Tending or having the power to evoke. Evoke: n. To call to mind or memory. (American Heritage Dictionary) Not the same as *provocative*.

[7] *Buffalo Mountain* in [Cope94] describes how to optimize communication within an organization. "The Pattern's name comes from the similarity between the visual graph (when the communications between roles in a project are plotted in a scatter gram) and the characteristic shape of a mountain in Colorado, and from analogies that can be made about the **forces** contributing to each."

This pattern can also be applied to pattern languages since they, too, require names that are memorable and easy to refer to.

Example

All the patterns in this language have *Evocative Pattern Names*.

Many of the patterns in well known pattern books, such as **Design Patterns**, use names designed to invoke images of the solution: *Bridge*, *Adapter*, *Proxy*, *Decorator* to name a few.

C.3.1 Pattern: Noun Phrase Name

Aliases: *Solution Phrase Name*

Context:

You are writing a pattern (or pattern language) that may need to be referred to by other patterns or pattern language*s*. You have created a pattern that you are attempting to name by applying *Evocative Pattern Name*.

Problem:

How do you name a pattern so that it is easy to remember and refer to?

Forces:

- n   Names that describe the **problem** are not unique since there may be several **solutions** to the **problem**. Supposed you were directed to "Apply the *Implementing State Machine* pattern." If there were several patterns that solved this **problem**, which **solution** does this name refer to?

- n   Names containing verbs or prepositions are difficult to use in conversation. Note the tension in this conversation: "What's that?" "Oh, it's an Object from a State."

- n   Names that describe the process of creating the **solution** are hard to use in a sentence describing the **solution**. It forces you to use phrase like: "In this design, we have an example of Create Objects for States."

- n   Describing the result of applying a pattern helps the reader visualize the result but it does not help convey the **problem** being solved. "What's that? Oh, that's a State Object."

- n   Since an important purpose of patterns is to foster communication by creating a shared vocabulary, pattern names should be easy to say.

Solution

Name the pattern after the result it creates. This allows the name to be used easily in conversation. In a pattern language, use a *Problem/Solution Summary* to help the reader find the right **solution**.

When referring to a pattern with a *Noun Phrase Name* where an understanding of the **problem** is important, include the **problem** in the referring phrase as in: "In this design, we used a *Proxy* to allow an object to be referred to remotely.."

Rationale

One of the most compelling aspects of a pattern is the way it transforms a situation, resolving some **forces** and giving rise to others. This is what makes patterns more than just "design rules" or a "style guide".

Focusing on the thing created by a pattern for naming leads to noun phrases. Naming the pattern above after the object it creates results in the name "State Object".

Example

The name of this pattern*, Noun Phrase Name,* is itself an example of a *Noun Phrase Name*.

Related Patterns

This pattern is a way to create an *Evocative Pattern Name*.

Another way to create an *Evocative Pattern Name* is to use *Meaningful Metaphor Name.* A *Meaningful Metaphor Name* may itself be a *Noun Phrase Name*.

C.3.2   Pattern: Meaningful Metaphor Name

Context:

You are writing a pattern (or pattern language) that may need to be referred to by other patterns or pattern languages. You have created a pattern that you are attempting to name by applying *Evocative Pattern Name*.

Problem:

How do you give your pattern a useful and memorable name?

Forces

n   Metaphors are a good source of short *Noun Phrase Names*.

n   People often find it easier to understand new concepts if they can be related to other concepts with which they are already familiar.

n   If you try to explain new concepts in terms of unfamiliar concepts, the reader will be baffled. Rocket Science metaphors are typically understood only by Rocket Scientists. Star Trek metaphors ("Darmok and Jalad at Tenagra[8]") are best understood by Trekkers. Hockey metaphors are understood best by Canadians.

n   If the link between the metaphor and your pattern is clear, readers will be able to transfer their knowledge of the metaphor into the context of your pattern, helping to clarify and facilitate their understanding of your pattern.

n   If the link between the metaphor and your pattern is unclear, the reader will be baffled. "I understand rocket science, but what does that have to do with your pattern?"

Solution

Find a meaningful metaphor for the pattern, and name the pattern accordingly. Some people are better with metaphors than others, so if a good metaphor doesn't jump out at you, go back to Noun Phrase Name. Ideally a metaphor will be familiar and easily understood by the average reader. If you have to explain the metaphor, it is not familiar enough. Clearly identify how the problem and solution relate to the metaphor, so that the reader is able to link his/her understanding of the metaphor concepts with your pattern.

Rationale

A metaphor effectively creates an association between your pattern and a set of parallel concepts with which the reader is (hopefully) familiar. Naming your pattern according to the metaphor you use to explain it helps the reader remember both your pattern and the metaphor. Clearly linking metaphor concepts with pattern concepts will help readers transfer their knowledge of the metaphor into the context of your pattern, helping to clarify and facilitate their understanding of your pattern

Example

---

[8] In the episode, "Darmok", the Enterprise crew encounters the Tamarians, whose language is based entirely on metaphors such as "Darmok and Jalad at Tenagra", taken from their own mythology [Okuda94]. Picard and the crew are initially frustrated in their attempts to understand the Tamarians because they are unfamiliar with Tamarian mythology. Note that this is an example within an example. Quite appropriate for a paper on metapatterns, n'est ce pas?

The *Visitor* pattern from **Design Patterns**, is an example of a Meaningful Metaphor Name.

The *Shopper* pattern in [Doble96] is another example of a *Meaningful Metaphor Name*. This design pattern describes how a Shopper object visits a number of objects to fill its Shopping Bag with items specified in a Shopping list. The name evokes an image of a person wandering from store to store trying to gather all the items on their shopping list.

Related Patterns

This pattern is a specialization of *Evocative Pattern Name*.

Noun Phrase Name is an alternative way to create an *Evocative Pattern Name* though many *Meaningful Metaphor Name*s are also *Noun Phrase Names*.

Buffalo Mountain is an example of a metaphor name that is not clearly explained as part of the pattern description, thus it has meaning only for the author and those with whom it has been shared verbally.

This pattern is itself an example of *Duplicate Problem Description*[9] since it has the same problem statement as *Noun Phrase Name*.

C.4   Pattern: Intent Catalog

Context

You are writing a pattern to which you would like other pattern writers to refer.

Problem

How do you make it easy to cross-reference patterns in a meaningful way?

Forces

- n   Coming up with concise summaries of problems and/or solutions is hard work; not everyone can do a good job of it.
- n   People will not provide references if it is too much effort.
- n   People who don't understand your pattern completely may make up inappropriate summaries of it.
- n   Patterns get too big if everything must be included.
- n   Patterns are hard to understand if relevant information is not included or referenced.

Solution

Provide a catalog of **pattern intents** that can be used as *Pattern Thumbnails* when other patterns need to refer to this pattern. The **intents** in the catalog should provide a 1-2 sentence "thumbnail" of what this pattern does. Where the pattern can be used to achieve more than one **intent**, each **intent** should be in the catalog.

Resulting Context

You may have more text to maintain as you evolve your pattern (language.)

Rationale

A good way to encourage something to happen is to make it the path of least resistance. Providing the intent catalog makes cross referencing less work than duplicating the information.

---

[9] Described briefly in the Related Patterns section of E.2 Pattern: Common Problems Highlighted

Related Patterns

The result of this pattern may be used as an *External Pattern Thumbnail* in a pattern that needs to refer to this pattern. In a pattern language, the *Intent Catalog* may be incorporated into a single Problem/Solution Summary. The main difference between the two concepts is that an *Intent Catalog* provides a list of possible uses for a single pattern while a *Problem/Solution Summary* lists the problem solved by each pattern in a language.

Examples:

In this pattern language, the *Intent Catalogs* of all the patterns have been collected into a *Problem/Solution Summary* table as part of the *Pattern Language Summary*.

D. Patterns For Making Patterns Understandable

A pattern is only as useful as it is perceived by its users. Pattern writers can put a lot of effort into describing their patterns but all this effort is for naught if the reader cannot understand it or gives up out of frustration. A key contributor is the quality of the writing, a factor into which we will not delve here. There are other factors some more specific to patterns, others somewhat general. The patterns described here are included because the reviewers felt they made a significant difference in how easily a pattern or pattern language was understood.

The patterns in this section apply to all elements of a pattern or pattern language. They all strive to help the pattern writer communicate their thoughts to the pattern reader in the most effective manner possible. A key step is the identification of a *Clear Target Audience*. This helps the writer chose *Terminology Tailored to Audience* as well as commonly *Understood Notations* for diagrams and illustrations. If the audience includes programmers, it is appropriate to include *Code Samples*, while *Code Samples as Bonus* ensures that the reader isn't obligated to read them to understand the pattern.

D.1   Pattern: Clear Target Audience

Aliases: *Target Audience, Identified Audience*

Problem:

Many people may read a particular pattern. How do you ensure that a pattern is easily understood by its intended audience?

Context:

You are writing a pattern or pattern language.

Forces:

- A pattern can be different things to different people.
- You can't satisfy all the people all of the time.
- Different people use different terminology.
- People with different backgrounds require different amounts of detail.

Solution:

Clearly identify a primary target audience with whom you would like to communicate the solution. Keep this audience in mind while writing the pattern. "Test" the pattern with (representative) members of the target audience.

It may even be useful to explicitly describe the target audience in the pattern (language) introduction. This helps set the expectations of the reader by telling them "up front" that they are (not) the intended audience. It also helps people determine which meaning of an ambiguous term you had intended.

Rationale:

A *ClearTarget Audience* focuses the pattern by providing criteria for including some information in the pattern and omitting other information.

Related Patterns:

Once you have identified the target audience, choose *Terminology Tailored to Audience* to maximize the bandwidth of communication to them.

Example:

This pattern language identifies its target audience in the first paragraph of the Introduction.

D.2   Pattern: Terminology Tailored to Audience

Problem:

How do you maximize the likelihood of the intended reader understanding your pattern?

Context:

You are writing a pattern or pattern language and have identified a *Clear Target Audience*.

Forces:

- n Concepts can be described using a variety of language styles and terminology.

- n Translating abstract concepts into concepts within a specific domain may be difficult for some people. The more concrete the terminology, the more likely it is to be understood by people familiar with the terminology.

- n The goal of a pattern is to be useful to the reader. If the reader doesn't understand the terminology, the pattern will not be as useful.

- n Expanding all acronyms and technical terms makes a pattern description more wordy.

- n Using terms without defining them can lead to misunderstandings (i.e. false agreement).

- n Using too much audience-specific jargon may limit the potential audience.

Solution:

Use terminology that is tailored to the audience. Use only those terms with which the typical member of the audience could reasonably be expected to be comfortable. Test the terminology with representative members of the target audience. As part of the introduction be sure to inform the reader of the "default" terminology source.

To ensure that you do not limit the audience unnecessarily, use the simplest language that effectively communicates the concepts. Include a Glossary of terms that may be unfamiliar. Introduce new terms in footnotes as they are encountered (or refer the reader to the Glossary.)

Resulting Context

The pattern or pattern language may not be as understandable to those readers outside the Target Audience if the terminology is too specialized.

Rationale:

A pattern that can be understood by the target audience is more likely to be useful.

Example:

This pattern language uses terminology specific to the patterns community. It does not explain terms such as **Forces** and **Context** because the *Target Audience* is already familiar with them and would be put off by detailed explanations.

D.3   Pattern: Understood Notations

Context:

You are writing a pattern and are trying to communicate concepts that are most appropriately communicated using diagrams or illustrations. You have identified a *Clear Target Audience*.

Problem:

How do you ensure that the diagrams are easily understood by your entire target audience?

Forces:

- **n** Diagrams and illustrations are often more effective than prose when it comes to communicating concepts, especially those related to software design. "A picture is worth a thousand words."

- **n** For any given concept (e.g. object model relationships) there may be a variety of diagramming notations and styles that can be used (e.g. Booch, OMT, etc.).

- **n** Readers are not necessarily familiar with all such notations and styles. If the readers are not familiar with the notation you have used, they may be unable to understand your pattern. "A picture you can't understand is worth a thousand words you can't understand."

- **n** Readers are diverse. If you leave room for interpretation, different readers may interpret your diagram in different ways.

- **n** Providing a detailed description of diagramming notations to your pattern will make it too large, and will distract the reader from what you are trying to communicate.

- **n** An expressive but obscure notation is less effective at communicating with most audiences than a less expressive but better-known notation.

Solution:

Use diagramming notations that are likely to be familiar to the target audience. Such notations should be widely used and easily understood (e.g. message sequence charts). If you are using a standard notation, always provide a reference to the standard. If not, or if there is any likelihood that potential readers are not familiar with the notation you are using, provide a clear, concise explanation of the notation when you first use it or refer the reader to a more detailed explanation in an appendix.

Rationale:

The more widely used the notation you use, the more likely that readers will be able to understand your diagrams without the need for a bulky and distracting explanation. Brief explanations of less common notations will help readers who are unfamiliar with the notation understand your diagrams, hopefully without creating too much of a distraction from the essence of your pattern. References for standard notations provide a means for curious readers to learn more.

Related Patterns:

If the explanation is included within the pattern or pattern language, ensure that it is a *Skippable Section*.

D.4   Pattern: Code Samples

Aliases: *Code Examples*

Context:

You are describing a solution to a software architecture or design problem. You have identified a *Clear Target Audience* that includes significant numbers of software designers and programmers.

Problem:

How can you make a software pattern sufficiently clear and unambiguous to facilitate straightforward implementation?

Forces:

- **n** Software-related concepts are often complex and difficult to explain.

- **n** Informal descriptive text is often unclear, and ambiguous.

- **n** Programming languages are designed to convey software concepts in a formal, precise, and unambiguous manner.

- **n** Many software workers are experienced and adept at reverse engineering concepts from software samples, and in fact prefer to learn ideas by looking at code.

- **n** Many software patterns can be implemented in many different ways.

- **n** Too much code interrupts the pattern's flow and may make it unmanageably large.

Solution:

Provide one or more implementation code samples, written in a prevalent programming language, to illustrate the pattern concepts. Use a programming language likely to be understood by the *Target Audience*. Choose an implementation approach that clearly demonstrates the essence of the pattern in a straightforward manner while minimizing unnecessary or distracting detail. Ensure that the code samples are well-commented and that all assumptions and design decisions are stated. Differentiate between aspects of the example that are essential to the pattern vs aspects that are arbitrary. Ensure that your code samples are "ready to run" (i.e. they are free from syntax errors and are complete). Syntax errors in code samples can be as distracting to people as they are to compilers.

Rationale:

Well-commented example code is formal, precise, and unambiguous, and can be readily understood by many experienced software workers. Code examples provide concept reinforcement, providing a means for the reader to verify that they have understood the essential concepts of the pattern.

Example:

All of the patterns in **Design Patterns** [GHJV95] include *Code Samples*.

Related Patterns:

Code Samples as Bonus ensures that the pattern can be understood without the *Code Samples* and can help reduce the disruption of flow.

D.4.1　Pattern: Code Samples as Bonus

Aliases: *Code Examples as Bonus*

Problem:

How can you ensure that the essence of your software pattern can be understood by your entire target audience, regardless of their familiarity with specific programming languages?

Context:

You are writing a software architecture or design pattern and are including *Code Samples*.

Forces:

- **n** Well-commented example code is formal, precise, and unambiguous, and can be readily understood by many experienced software workers.

- **n** There is no universally understood programming language. *Code Samples* will be understood only by those readers who are familiar with the language you use.

Solution:

Ensure that the pattern can stand on its own, able to communicate its essential concepts even if the code examples were deleted. *Code Samples* should be treated as an optional bonus, providing concept reinforcement and implementation guidance for those readers who are familiar with the language of the examples. Ensure that *Code Samples* embedded within the text can be easily skipped, or that they are in a separate *Skippable Section*.

Descriptions of essential algorithms and key object relationships and interactions should be provided using notations other than implementation code. Suitable notations include: pseudo-code, flowcharts, object modeling notations, event traces and object interaction diagrams. Whatever notations you choose, ensure they are *Understood Notations*; don't invent your own, or use little known notations, unless absolutely necessary.

Rationale:

The purpose of a pattern is to communicate to as wide an audience as possible. If the pattern cannot be fully understood without reading the code examples, then readers who are not familiar with the example language will not be able to understand the pattern.

Code Example:

Reader >> understandsSmalltalk

"Answers True if the reader understands Smalltalk, otherwise signals an exception"

self understands: #Smalltalk

ifTrue: [^True]

ifFalse: [self doesNotUnderstand].

E. Pattern Language Structuring Patterns

This section contains patterns that solve problems unique to pattern languages. They deal primarily with how to assemble a number of related patterns into a cohesive pattern language that is more than the sum of its parts. The language should be introduced using a *Pattern Language Summary* that introduces the overall problem and the patterns that will be used to solve it. The *Problem/Solution Summary* is a key part of this introduction because it allows individual patterns in the pattern language to be picked out when the document is used as a reference manual. Larger pattern languages often have a non-trivial structure that can be better communicated using *Distinctive Headings (that) Convey Structure*. They also often contain alternative, possibly mutually exclusive solutions to the same problem; these can be pointed out by ensuring that *Common Problems (are) Highlighted*.

A good way to tie together the patterns in a pattern language is through the use of a *Running Example* that illustrates the application of the patterns to an example of the larger problem. To improve understanding, any non-standard terminology should be expanded in a *Glossary*.

E.1 Pattern: Pattern Language Summary

Problem:

How do you give the reader an overview of a set of patterns?

Context:

You are writing a pattern language describing the solution for a complex problem.

Forces:

- n A pattern language should be more than just the sum of its parts.

- n The connections between patterns (how they relate with one another) are not always obvious.

- n Inter-pattern relationships are sometimes difficult to understand solely from the perspective of the patterns involved in the relationships.

- n Describing the relationships between many patterns in one place takes extra effort and increases the bulk of the language.

Solution:

Identify the set of patterns as a pattern language and write a summary that introduces the larger problem and the patterns that contribute to solving it. This summary explains why the patterns belong together, the common threads found in more than one pattern, and how the patterns can be used together to do something useful. It can also be used to introduce the *Running Example*. By describing the overall context, it may significantly reduce the need to provide duplicate, detailed contexts within each pattern, although this could make the individual patterns less usable outside the context of the language.

In larger pattern language*s*, it is useful to provide a *Problem/Solution Summary* to help the reader find the pattern(s) that solve their specific problems.

Rationale:

A *Pattern Language Summary* provides the "big picture" while the related patterns section of each pattern provide the detailed linkages. The *Pattern Language Summary* may be the only place one can talk about the pattern language as a whole.

In the PLoP'96 review sessions in which the authors participated, the reviewers consistently preferred languages that introduced the patterns in a *Pattern Language Summary* over those that launched right into describing the patterns.

E.1.1 Pattern: Problem/Solution Summary

Context:

You are writing the *Pattern Language Summary* of a pattern language that includes patterns that may be useful individually as well as within the flow of the language. Many of the patterns have *Noun Phrase Names* based on the **solution**.

Problem:

How do you make it easy for a reader to pick out useful patterns that solve their **problem**?

Forces:

- ⊓ The **problems** and **solutions** in a pattern language may be spread across many pages of text. It could be very time consuming to read the whole language in search of a particular (yet to be identified) pattern.

- ⊓ A person using a pattern language may not need to use all the patterns in the language (and certainly not all at once).

- ⊓ Summarizing patterns in the introduction takes extra effort and increases the size of the pattern language.

Solution:

Provide a table in the *Pattern Language Summary* that summarizes all of the patterns, including a brief description of each pattern's **problem** and the corresponding **solution**. You can do this by collecting the *Intent Catalogs* for all the patterns in the language into a single convenient table as part of the *Pattern Language Summary* or in an Appendix or References section.

Rationale:

This additional information helps the reader of the pattern language quickly zero in on the pattern(s) that may solve their specific **problem**.

Examples:

This pattern language use a *Problem/Solution Summary* to give the reader an early indication of the structure of the language.

E.2 Pattern: Common Problems Highlighted

Context:

You are writing a pattern language that provides several patterns that solve the same **problem**.

Problem:

How do you make readers aware that they should choose one of the alternative solutions?

Forces:

- ⊓ A pattern is normally considered to be a **problem-solution** pair. Most pattern forms currently in use do not lend themselves to sharing a **problem** section amongst several competing patterns without taking some liberties with the form.

- ⊓ Repeating the **problem** in each pattern that provides a **solution** may confuse readers by giving them a sense of "deja vu" without explaining the cause for it. They may not realize that there are several **solutions** to choose from and may expend considerable energy trying to figure out how to apply all the **solutions** simultaneously!

- ⊓ Having the **problem** repeated in each pattern that provides a **solution** increase the effort required to maintain each pattern.

n   The **problem** section is not the only part of the pattern that would have to be duplicated. All patterns that solve the same **problem** should include the same set of **forces**, while the **context** determines their relative priority.

Solution:

When several patterns solve the same **problem**, make this obvious by pointing out to the reader that there are several **solutions** to this **problem**. You can capture the common **problem** and **forces** in one place using *Separate Problem Description*[10] or *Referenced Problem Description*[11]. If you choose to repeat the **problem** description as described in *Duplicate Problem Description*[12] you should notify the reader that you have done so.

Related Patterns:

Separate Problem Description solves the **problem** by factoring out the common **solution** into a separate pattern.

Duplicate Problem Description solves the **problem** using the brute force method of cloning the **problem** description into each pattern that provides a **solution**.

Referenced Problem Description solves the **problem** by including the **problem** description in one pattern and referring to it from each other pattern that provides an alternate **solution**.

Rationale:

It is very easy for readers to become confused if several patterns have similar or identical **problem** descriptions unless it is pointed out to them that these patterns provide alternative **solutions**.

Example:

Common Problems Highlighted is itself an example of *Separate Problem Description* since it exists primarily to point the reader to the alternative **solutions**. This is easily recognized by the fact that the **solution** section merely refers the reader to a number of other patterns; it acts like a "traffic cop."

E.3   Pattern: Running Example

Problem:

How can you make it easier for the reader to put a pattern language into practice?

Context:

You are writing a pattern language that provides step by step instruction on how to do or implement something.

Forces:

n   The pattern should be clear and complete so that reader can use it with minimum effort or chance of mistake.

n   The pattern should be as concise as possible without being too terse for most people to understand.

n   Many people find abstract descriptions very hard to understand.

n   Examples are very useful but must not take a lot of effort or prior knowledge to understand.

---

[10] Yet to be written.

[11] Yet to be written.

[12] Yet to be written. *Meaningful Metaphor Name* and *Noun Phrase Name* solve the same problem and share a *Duplicate Problem Description*.

- **n** Any one example may not be ideal for explaining a specific pattern.

- **n** When a language contains a significant number of patterns, each pattern must necessarily be more concise than a free-standing pattern if only for reasons of overall pattern language size.

Solution:

Try to use a single example in all patterns in the language. Explain it once, possibly in the language introduction. Use it to illustrate how each pattern in the language contributes to the solution. Use additional examples where the *Running Example* does not illustrate the pattern effectively.

Rationale:

A single *Running Example* gives the reader more insight into applying the whole pattern language than a bunch of individual examples. In effect, it is a case study. The reader does not need to invest time and effort into understanding the example for each pattern; they pay this cost only once, when the example is introduced.

While this rationale has been expressed in terms of pattern languages, the same arguments are applicable to the use of a running example within stand-alone patterns.

Example:

This pattern language attempts to be a *Running Example* of all the patterns it contains. The authors have also tried to identify one or two examples of each pattern from published pattern works to augment the *Running Example*.

E.4 Pattern: (Distinctive) Headings Convey Structure

Problem:

How do you help the reader understand how the individual patterns he/she is reading fits within the overall structure of the language?

Context

You are writing a pattern language that has a non-trivial structure. You are applying *Visible Language Structure*[13] because you recognize that it is important for the reader to be able to understand how the individual patterns he/she is reading fits within the overall structure of the language. You are attempting to make the resulting language *Single-Pass Readable*.

Forces:

- **n** In pattern languages with complex structure, readers may find it easier to appreciate the individual patterns if they understand how they fit within the structure of the language.

- **n** When reading through a complex pattern language for the first time, it is easy to lose track of where you are within the structure of the language.

- **n** A lengthy "you are here" section for each pattern is repetitive, adds unnecessary bulk to the language, and may distract the reader from the patterns themselves.

- **n** An introductory section, at the beginning of the pattern language, can be an effective means to communicate the overall structure of the language.

- **n** A language takes longer to read (and is not *Single-Pass Readable*) if the readers need to constantly refer back to the introductory section to figure out where they are.

Solution:

---

[13] Yet to be written. *Visible Langauge Structure*: Ensure that the structure of your language is visible so that the reader can keep track of where they are in the language.

Make individual pattern headings visibly different from all other document section headings. Prefix pattern headings with hierarchical section numbers, where the section numbering hierarchy parallels the language structure.

Rationale:

The reader can easily recognize pattern sections and can tell at a glance how the given pattern fits within the language structure. Section numbers are concise, and do not distract the reader from the patterns themselves. Major changes in section numbers can signal the reader that he/she has come to a new section of the pattern language.

Example:

In this pattern language, we have organized the patterns into 5 major categories lettered A thru E. Within a category any patterns that are clearly subservient to another pattern have been numbered by adding a .1 to the pattern number of the higher level pattern. *Code Samples* (D.2) is a pattern within the *Maximizing Understanding* category (Section D.) *Code Samples as Bonus* (D.2.1) is an extension of Code Samples hence the subordinate numbering. Similarly, *Evocative Pattern Name* (C.3) is supported by *Noun Phrase Name* (C.3.1) and *Meaningful Metaphor Name* C.3.2); all fit within the *Pattern Naming and Referencing* category (Section C.)

In **Episodes** [Cunningham96], the patterns are divided into three sections, Product, Development, and Programming. The patterns in each section are numbered accordingly.

Related Patterns:

Section-Name Running Footers/Headers is another pattern that can be used to achieve *Visible Language Structure*.

Page Numbered Pattern References make it easier for a reader to flip to the page describing a specific, referenced pattern.

E.5 Pattern: Glossary

Problem:

How do you clarify unfamiliar terminology in a pattern language without interrupting the flow of the pattern?

Context:

You are writing a pattern language that involves terminology that may not be familiar to the *Target Audience*.

Forces:

- Patterns may need to use terminology that is unfamiliar to readers.

- Patterns should be concise. Defining all terms within the pattern description may make it hard to follow for those familiar with the terminology..

- Expanding the terminology elsewhere may require the reader to flip pages often.

- Putting all the definitions in one place makes it easier to find them.

Solution:

Provide a glossary of terms as part of the pattern language. The glossary gathers terms that are used in multiple patterns within the language with definitions of the terms. If you feel that it is essential to have the definition handy, you may include a short definition of the term in a footnote as well.

Rationale:

A glossary collects terminology from multiple patterns in one place, thereby making the patterns more concise. The definitions make the patterns understandable by people unfamiliar with the terminology. Glossaries are a proven technique used in many written publications to achieve the same purpose.

3.0 Back Matter

Concluding Remarks

This pattern language is by no means complete. As long as the art of pattern writing continues to evolve and mature, this language will need to evolve with it. There are many areas that this language has not even attempted to cover. It does not prescribe a process for the creating a pattern or pattern language. A number of such patterns come to mind, patterns such as *Record the Solution*, *Determine the Problem*, *Find the Forces* and *Separate the Problem from the Context*.

Except for the section on pattern Naming and Referencing, this language has deliberately tried to avoid any questions of style. Style is a very personal thing and it is too early in the life cycle of the pattern to prescribe a specific style. Each style of pattern writing probably warrants its own pattern language.

The authors hope that you find this language useful in your pattern and pattern language writing endeavors and that you will share your favorite pattern writing patterns with the patterns community. Please forward any comments to the authors via e-mail.

Acknowledgements

4.0 Appendices

4.1 Problem/Solution Summaries

The following tables summarize the patterns in this pattern language for reference purposes.

Table 1: Pattern Structure Patterns

Problem	Solution	Pattern Name
How do you make sure that all necessary information is covered in a pattern	Include the following elements: **Pattern Name**, **Problem**, **Context**, **Forces**, and **Solution**	*Mandatory Elements Present*
How do you communicate essential information that does not fit well into the Mandatory Elements?	Include the following sections when they help convey the information: **Resulting Context, Related Patterns**, **Examples**, **Code Samples**, **Rationale**, and **Aliases**.	*Optional Elements When Helpful*
How do you ensure that the reader understands the choice of **solution**?	Regardless of the style chosen for the pattern description, ensure that the **forces** are highly visible	*Visible Forces*
How do you make it easy to get the essence of a pattern **solution** quickly?	Write the pattern so that it is not necessary to read the later parts in order to understand the earlier parts.	*Single-Pass Readable*
How do you minimize the amount of reading required to get the essence of a pattern?	Clearly identify the **Problem**, **Context** and **Solution** parts so that the reader can quickly determine whether this pattern applies to them.	*Skippable Sections*

Table 2: Pattern Naming and Referencing Patterns

Problem	Solution	Pattern Name
How do you refer to other patterns within the description of your pattern?	When referring to patterns within the body of your pattern, weave the pattern names into the narrative.	*Readable References to Patterns*
How do you name a pattern so that it is easily remembered and referred to?	Choose a pattern name that conjures up images that convey the essence of the pattern **solution**.	*Evocative Pattern Name*
How do you give a pattern a useful and memorable name?	Name the pattern after the result it creates.	*Noun Phrase Name*
How do you give a pattern a memorable name?	Find a meaningful metaphor and name the pattern after it.	*Meaningful Metaphor Name*
How do you make a pattern part of a larger group of patterns?	Read other pattern languages and describe the relationships to other patterns.	*Relationship to Other Patterns*
How do you refer to external patterns in a concise but meaningful manner, so that the understanding of your pattern is not compromised?	Include the *Evocative Pattern Name,* an author/year tag and a footnote with a brief summary of the pattern.	*External Pattern Thumbnails*

Table 3: patterns for making patterns Understandable:

Problem	Solution	pattern Name
How can you make a software pattern sufficiently clear and unambiguous to facilitate straightforward implementation?	Provide one or more implementation code examples, written in a prevalent programming language, to illustrate the pattern concepts.	*Code Samples*
How can you ensure that the essence of your software pattern can be understood by your entire target audience, regardless of their familiarity with specific programming languages.	Ensure that the pattern can stand on its own, able to communicate its essential concepts even if the code examples were deleted.	*Code Samples as Bonus*
How do you ensure that diagrams are easily understood by your entire target audience?	Use diagramming notations that are likely to be familiar to the target audience.	*Understood Notations*
How do you ensure that a pattern is easily understood by its intended audience?	Identify a clear target audience and keep this audience in mind while writing the pattern.	*Clear Target Audience*
How do you maximize the likelihood of the intended reader understanding your pattern?	Use only those terms with which the typical member of the audience could reasonably be expected to be comfortable	*Terminology Tailored to Audience*

Table 4: Language Structure Patterns

Problem	Solution	Pattern Name
How do you give the reader an overview of a set of patterns?	Summarize the pattern language in the Introduction.	*Pattern Language Summary*
How do you make it easy for a reader to pick out useful patterns that solve their **problem**?	Provide a table that summarizes all of the patterns, including a brief description of each pattern's **problem** and the corresponding **solution**.	*Problem/Solution Summary*
How do you make readers aware that they should choose one of the alternative **solutions**?	When several patterns solve the same **problem**, make this obvious by pointing out to the reader that there are several **solutions** to this **problem**.	*Common Problems Highlighted*
How do you help the reader understand how the individual patterns he/she is reading fits within the overall structure of the language?	Prefix pattern headings with hierarchical section numbers, where the section numbering hierarchy parallels the language structure.	*Headings Convey Structure*
How can you make it easier for the reader to put a pattern language into practice?	Try to use a single example in all patterns in the language.	*Running Example*,
How do you clarify unfamiliar terminology in a pattern language without interrupting the flow of the pattern?	Provide a Glossary of all terms that may be unfamiliar to the audience.	*Glossary*

4.2 References

[Alexander77] Christopher Alexander et al., A Pattern Language, Oxford University Press, New York, 1977.

[Alexander79] Christopher Alexander, The Timeless Way of Building, Oxford University Press, New York, 1979.

[Beck96] Kent Beck, "Smalltalk Best Practice Patterns", Prentice-Hall, New Jersey 1996

[Bercz96] Stephen P. Berczuk, "A Pattern Language for Ground Processing of Science Satellite Telemetry" in [PLoP95].

[Cockburn96] Alistair Cockburn, "A Medical Catalog of Project Management Patterns", PLoP'96 Proceedings.

[Copelien96] Jim Coplien, "A White Paper on Patterns", SIGS books, 1996

[Cunningham96] Ward Cunningham, "Episodes: A Pattern Language of Competitive Development", in [PLoP95]

[Foote96] Brian Foote and Joseph Yoder, "Attracting Reuse", PLoP'96 Proceedings.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software."* Addison-Wesley. ISBN 0-201-63361-2.

[Miller56] George A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information", Psychology Review, 63, 81-97.

[Okuda94] Michael Okuda, Denise Okuda and Debbie Mirek, The Star Trek Encyclopedia, Pocket Books, 1994. Copyright © 1994 by Paramount Pictures.

[PLoP94] Proceedings of PLoP-94 - "Pattern Languages of Program Design" published by Addison-Wesley in 1995.

[PLoP95] Proceedings of PLoP-95 - "Pattern Languages of Program Design" published by Addison-Wesley in 1996.

[Roberts96] Don Roberts, Ralph Johnson, "Evolve Frameworks into Domain-Specific Languages", PLoP'96 Proceedings.