Semantic University

Semantic Web Design Patterns

The free book “Linked Data Patterns” is available online (http://patterns.dataincubator.org/), as as PDF (http://patterns.dataincubator.org/book/linked-data-patterns.pdf), and an Ebook (http://patterns.dataincubator.org/book/linked-data-patterns.epub). The articles will pick out some of the more useful patterns from the book that address commonly encountered problems.

Today’s Lesson

Like a question on Stack Overflow, a design pattern is often posed as a question: how do we solve some design problem? However a design problem is, by its nature, nonspecific, and rarely has a single straight-forward answer. There might be several ways to solve the same problem, some better than others depending on the specific situation and the specific context of the problem.

So a design pattern also captures more context: when might the problem arise? What alternative solutions are there? And what trade-offs are part of a specific solution? A design pattern is intended to share not just solutions but a better understanding of both the problem and how it might be solved.

Why Design Patterns Are Useful

Let’s look in a little more detail at design patterns and how they are different to a simple tutorial or an entry on a Q&A site. What are the benefits of a design pattern approach?

In my experience, design patterns have a number of different benefits.

Firstly, patterns have a well-defined structure (more on that in a moment). This consistent layout makes it easy to browse through a collection of patterns to find relevant help and then dive further into the material. The structure encourages the author of the pattern to think carefully about the knowledge they're sharing, whilst making the material more consistently accessible to a reader.

Secondly, unlike a tutorial or recipe which typically guides you through a single approach to solving a problem, patterns encourage discussion of related and complementary approaches. Design decisions are rarely clear cut, so it can be useful to understand the context in which a decision is made and the resulting trade-offs. Communicating these nuances is how we share knowledge rather than just fixes for a problem.

Finally, patterns usually have simple clear names. Whilst this makes them more memorable, the real benefit is that patterns can start to form a shared vocabulary that can help streamline discussion and communication between developers within a team or in an industry. It's often possible to clearly describe a technical architecture or piece of code by reference to a set of patterns, avoiding the need to get into the implementation details.

A knowledge of key design patterns has become an essential part of enterprise software development as it allows teams to quickly communicate their designs to one another. In particular software developers have embraced design patterns help document and share good practices in designing object-oriented systems. But the technique isn't limited to use in that context. Design patterns have been used to describe refactoring techniques, user interface design, architecture, and in many other different areas. Design patterns are useful wherever we encounter recurring problems with common solutions. And in particular, wherever there is a need to share good practices within a community.

Semantic Web technologies can be complex to learn and can be applied in many different ways. There is a growing body of experience in the wider community about how to create good semantic web applications and how to structure data using RDF that makes it easier to share and extend. A design pattern approach is useful for sharing this hard-won experience.

Anatomy of a Design Pattern

As I mentioned earlier, design patterns have a consistent structure. While there are a number of variations, a design pattern is typically organized as follows:

- **Problem Statement**—a description of the problem or issue that the pattern attempts to solve. This is typically expressed as a simple question
- **Context**—describes the circumstances when the problem might occur to help you identify it
- **Solution**—a description of a recommended solution to the problem
- **Example**—a worked example that helps to illustrate the solution
- **Discussion**—a deeper discussion of the solution, typically highlighting advantages and disadvantages and alternate options
- **Related Patterns**—references to other relevant patterns that offer alternate approaches to or build on this pattern

A collection of patterns is referred to as a pattern catalog. A catalog can also include anti-patterns: examples of poor design and things to avoid.

Linked Data Patterns

The Linked Data Patterns (http://patterns.dataincubator.org/) book is a pattern catalog that gathers a range of design patterns relevant to Semantic Web applications. Rather than focusing on individual technologies the patterns cover a number of broad topics. These are outlined in the following sections. Later articles in this series will focus on each of these topics in more detail.

Identifier Patterns

Identifiers are fundamental to RDF and Linked Data. Successful use of the technology requires having good, stable URIs that can be used to publish and share data. Patterns for defining identifiers can help answer questions such as:

- How can we create URIs from existing identifiers, such as database keys?
- How can we create stable URIs that are free from implementation details?
- How can be make URIs “hackable” to make it easier for people to link together different datasets?

Modeling Patterns

Modeling is one of the richest source of design patterns. There are often many different ways to structure some data, each with its own impacts on flexibility and ease of evolution of the resulting data model. RDF modeling is, in essence, entity-relationship modeling and will already be familiar to many. But even so there are some graph data patterns and general RDF best practices that are useful to share. RDF modeling patterns can help answer questions ranging from the simple to the complex:

- How can we communicate a preferred label for a resource?
- How do we model complex relationships between resources?
- How do we structure data to get the most from a graph model?

Publishing Patterns

Linked Data is intended to be published and shared: both privately within an enterprise and more openly on the web. There are a number of publishing patterns that can make data easier to discover, easier to consume, and more usefully inter-linked. Unlike modeling patterns—which emphasize the flexibility of an individual data model or data set—publishing patterns encourage thinking about data in a wider context, helping answer questions such as:

- How can we discover data associated with a web page?
- How can we integrate different datasets?
- How can we remove a dataset, or move it to a new location?

Data Management Patterns

When working with RDF data its often necessary to partition a dataset into smaller chunks known as named graphs. A named graph is just a set of RDF triples that has its own identifier. Named graphs have been available as a feature in many RDF databases and as part of the SPARQL query language for many years. They are a very powerful feature that, when applied, can make an RDF dataset much easier to maintain.

However, there are several different ways that a dataset could be broken down into graphs to make it more manageable, and the right approach depends on the needs of the individual application. Data management patterns help document these alternate approaches to using named graphs, addressing

concerns such as:

- How do we track the source of some collection of RDF triples?
- How do we organize a triple store to make it easier to manage individual resources?
- How can we get a full description of a resource, regardless of how the data is organized into graphs?

Application Patterns

There are plenty of great design patterns that cover software design and architecture in general. However, the flexibility of RDF and SPARQL—coupled with their integration with web technologies—offers a lot of extra opportunity for creating dynamic data-driven applications. These applications have some fundamentally different requirements from traditional software applications, and require new, different design patterns.

RDF application patterns cover a range of different topics including how to apply SPARQL to more than just querying of data, as well as how to take advantage of RDF extensibility and merging capabilities to parallelize applications and data retrieval. Application patterns can help answer questions such as:

- How can we validate or transform some RDF data using SPARQL?
- How can we improve performance of data loading or retrieval?
- How can we write applications to take advantage of new data, whilst being tolerant of missing data?

Conclusion

In this tutorial we've introduced the concept of a design pattern: a solution to a modeling or software design problem that is used to share knowledge and experience among developers. Design patterns complement tutorials and reference documentation by helping to share alternative approaches whilst also developing a shared vocabulary among practitioners. Design patterns have been successfully used for many years in many different areas.

Increasingly semantic web developers are starting to recognize the benefits of design patterns and how they can help share solutions that cover a wide range of different areas. These areas include creating good identifiers, data modeling, publishing data for re-use, managing large datasets, and building flexible semantic web software.

In future tutorials in this series we'll dig into each of these areas in more detail, exploring some of the key design patterns that can help you make the most of semantic web technology.

‹ PREVIOUS LESSON (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/LEARN-SPARQL/SPARQL-VS-SQL/)

NEXT LESSON › (HTTPS://CAMBRIDGESEMANTICS.COM/BLOG/SEMANTIC-UNIVERSITY/SEMANTIC-WEB-DESIGN-PATTERS/SEMANTIC-SEARCH-SEMANTIC-WEB/)

1 Beacon St | 15th Floor | Boston, MA 02108