Smart Data: A System for Generating Application-Agnostic Interactive Data

Ashraf Gaffar
Dept. of Engineering & Computing Systems
Arizona State University
Mesa, Arizona, USA
agaffar@asu.edu

Abdessamad Tridane
Dept. of Mathematical Sciences
UAE University
Al-Ain, UAE
a-tridane@uaeu.ac.ae

Kris Pribadi
Renaissance Sciences Corporation
Chandler
Arizona, USA
kpribadi@rscusa.com

ABSTRACT

Interaction design focuses on the complex interaction between human and software. The cognitive part of interaction relies on the application features offered through the user interface and implemented by the underlying software. Data associated with this paradigm is application-dependent and has little value without it, a “stand-alone-application data”. On the other hand, “stand-alone data” could be built as database table or other formal structures, allowing for “smart and scalable” communication using queries. However, the majority of valuable data is unstructured, heterogeneous documents of mixed contents (text, links, tables, graphics, audio, video, etc.). While great for human consumption, this approach is not scalable, resulting in information overload and anxiety; humans can comprehend limited amount of data. Tools focus solely on the syntax without comprehension and end up with many false hits, a “guesswork” at best. We propose a new Smart Data approach to create and disseminate data with both syntax and semantic contents built-in, allowing for smart and scalable interaction between humans and data regardless of the application used. We use HCI-patterns as a case study of adding smart layer to heterogeneous, rich data.

KEYWORDS

Smart Data, Interactive data, semi-structured data, information overload, XML.

1 INTRODUCTION

Christopher Alexander is considered one of the pioneers to use the concept of design patterns [1]. His goal was to improve the quality of work in architecture and building construction by recording a well-defined set of his previous design rules which he referred to as patterns. “Although the quality of a well-designed building is hard to put into words, the patterns themselves that make up that building are remarkably simple and easy to understand” [2]. The software engineering community has popularized patterns throughout the entire design spectrum from requirements all the way to testing, deployment and reengineering. In the field of Human Computer Interaction (HCI), patterns are commonly used as a step towards building more usable user interfaces. For instance we have patterns for tasks, dialogs, user types, interface layout, and platform specifics [3]. HCI patterns records information about frequently encountered problems and explain how they can be solved. By capturing the indications and solutions of common problems, patterns can enhance communication between software practitioners and increase design reuse. HCI pattern Languages have also been proposed as a way to accumulate and group a large body of HCI knowledge. When similar or complementary patterns are used simultaneously, they represent a “design language” that resembles a process for a systematic approach of solving software development challenges. Pattern languages are not intended to be similar to high-level programming languages. They rather resemble a set of interrelated-patterns. Nonetheless, they still provide design building blocks for describing a particular design problem along with a solution.

Patterns are meant to help in developing concrete design artifacts with valuable human experience in them. However, apart from scarce code snippets, the predominant presentation method is a highly heterogeneous set of text, graphics, and sometimes video. While this can generate great, “cool” documents, they are only human-readable and –
processable. Apart from limited text and metadata search, they are not suitable for bulk software processing, greatly limiting their use in large numbers. Different interviews and studies [4], [5], [6], revealed that designers found patterns highly beneficial, yet faced some difficulties when they tried to find, understand and apply them correctly in their context. When asked to implement the same pattern on different platforms, they found it to be more difficult than they expected. Once they were trained, transforming a pattern into a program code was a manageable task.

Several researchers identified that the lack of tool support, coupled with unclear description of the process of disseminating them, compromises these benefits [7]. The text-based formats currently popular for documenting patterns cause them to have problems similar to those of design guidelines and recommendations – they tend to be abstract and ambiguous, and hence difficult to reuse [8], [9].

2 BACK GROUND AND RELATED WORK

Writers of HCI patterns tend to emphasize user experience and human aspects of the user interface (UI) design. They use elaborate narrative formats to convey theories and concepts of interaction design and human factors. On the other hand, application UI developers need concrete and pragmatic details through their design activities. They often find it hard to comprehend and translate textual pattern knowledge into concrete design. Text descriptions of patterns are not easy to integrate into design (CASE) tools, an indispensable help to developers today. With the plethora of patterns available, they also get inundated with huge number of pattern literature and links in many books and on the Internet [4], [5], [6]. They have to manually read and sift through piles of texts looking for some concrete patterns to apply. In this paper we introduce a framework to represent patterns data as smart components by identifying and rewriting their semantics as a model for designers and design tools. This model transforms textual patterns into programmable pattern objects with well-defined interfaces that reflect the knowledge underlying patterns data. This makes them accessible in any object oriented programming language as well as in XML for tool interoperability.

2.1 Early Attempts to Increase Pattern Comprehension

Pattern authors have long realized the problem of the limited ability of humans to comprehend large volume of text, and the need to increase access to the large number of patterns available. Attempts to address this problem can generally be put under three categories:

- **Internal Pattern Structure**, where pattern author use an existing, well-known format (e.g. [10], or define a new template that they strictly apply throughout their collection (e.g. [2]. This can help deliver important and popular pattern ideas like problem, context, forces and solution to users. However, other authors simply published their patterns in a free format without using any specific template [11].

- **Intra-collection Structure**, where pattern authors organize patterns within each collection (using boxes and arrows, or using pointers and references within each pattern) to show how they can interact with each other in a comprehensive way.

- **Inter-collection Structure**, which refer to how patterns from different collections can be organized together. While this is an important aspect for the user -due to the ever growing number of pattern collections- it has generally received little or no attention by most pattern authors. An effort was made by [12] who suggested a pattern language for pattern writing. They introduced a format that can be used to format all patterns and pattern languages. The main disadvantage of this attempt was trying to impose a fixed format that only few authors were willing to use, defeating the main purpose of this approach. Pattern authors have their individual points of view in using different formats. Other
attempts followed suit [13] but –again– fixed formats were imposed, garnering only limited support.

Besides the different formatting techniques, there are other issues associated with patterns documents. Logistical concerns are often tied to the storage, publishing and delivery of patterns to their users. Furthermore, reuse challenges represent the decisive factor in their success or failure. [14] discussed the potential benefits of using patterns as a vehicle for design components reuse. Due to the innovative nature of the design activity, the reuse of designs represents only a small part of the total design effort, the integration of multiple artifacts together is even less.

Patterns are a popular and common approach to convey human knowledge. Despite their creative nature, designers usually need to apply a structured approach to help manage all design activities and keep them within the planned resources. Partial automation of this process, combined with sound experience and good common sense can significantly facilitate the analysis and design phase of software development [15]. If patterns were presented in machine-readable format, tools could help combine them together at high design levels similar to what we do with machine-readable code idioms and high-level programming languages. The Smalltalk Refactoring Browser for example; a common tool for reusing Smalltalk coding patterns at the source code level, provide developers with help on how to use patterns in three ways as per [16]:

- "Generate program elements (e.g. classes, hierarchies) for a new instance of a pattern, taken from an extensible collection of "template" patterns.
- Integrate pattern occurrences with the rest of the program by binding program elements to a role in a pattern (e.g. indicating that an existing class plays a particular role in a pattern instance)
- Check whether occurrences of patterns still meet the invariants governing the patterns and repairing the program in case of problems"

Our work aligns with these efforts and proposes a new approach to write patterns as smart components that offer their semantic contents and behavior to users and tools through human user interface (UI) as well as programmable interfaces (API). In this paradigm, we do not suggest a fixed template to be followed by pattern authors, but rather an abstract “generic model” that covers a wide range of formats and promotes interoperability between them both at human- and machine levels. The generic model allows for instantiating concrete, programmable pattern objects, or “complex types” in the object oriented paradigm. The idea allows pattern authors to use their creativity and their own formats in writing patterns and then add an additional layer to transform them into software components with the same semantics, and without loosing any knowledge contained in them. Once transformed, patterns become smart objects runnable in any runtime environment and can be manipulated by tools. The concept equally encourages new patterns to be written originally as objects instead of text artifacts.

3 MODEL-BASED APPROACH AND SCALABILITY

In document-based approach, authors directly create text documents that are instances of human experience, only suitable for human reader or keyword lookup. In model-based approach, an additional “modeling layer” is inserted between the data and the reader (human or machine). To create smart data using model-based approach, we first design documents as abstract models, and specify the desired structure, syntax and semantics using these models. We formally describe the desired behavior of each model and the interaction (or interoperability) between different models as well as the integration of these models into different design processes and artifacts. During the modeling phase, we use real samples or simulated data contents as well as algorithms and software tools to experiment and validate the models. This modeling approach has many advantages. At different design phases, it allows designers to manipulate abstract models with just the right amount of information and integrate them into
their design artifacts without cluttering them with too many details. At the code level, it enhances interaction and integration with the application source code once data models have well known interfaces that accurately represent their semantics. Moreover, scalability is enhanced by allowing tools to interact with these models in an automated way, and in much greater numbers than the human processing capacity with the narrative format. Looking up information using different queries and search criteria will be greatly improved.

A Smart Data Environment, SDE, is the system that helps users get access to a large number of patterns as software components, and offers the functionality needed to efficiently manipulate them. To explain the idea, we abstract the concept of Figure 1 into the complete lifecycle of smart data as shown in Figure 2. The key points of this lifecycle are the dissemination process, the assimilation process, and the Smart Data Environment, SDE, which connects these two processes together.

3.1 The Dissemination Process

Dissemination refers to “the activities associated with delivering knowledge and experience from pattern authors to pattern users –or designers”. For efficient dissemination, we need to reduce the time spent by users in looking up patterns, and the ability to locate all patterns that can be useful according to some search criteria that a user can apply. So far, these activities have been left up to the user. Pattern authors simply publish there patterns, generally in books or on the Internet and the dissemination process stops there. Some pattern authors recognized the problem and added links within their collections to other collections. This, however, adds to the confusion of the users as they see similarities between collections. They get distracted or lost in the available maze of patterns. In this context, we define the visibility problem that new patterns suffer as “new patterns become diluted in huge pattern offerings and hence get no significant chance of making their way to users”. This has been confirmed by the results of an empirical study [4], [5]. Consequently, many designers limit their pattern
repository to few patterns that they already know, and rarely look for new patterns.

3.2 The Assimilation Process

Assimilation refers to the design decisions made by pattern users as to how selected patterns can be applied during different stages of design, and how they can be combined together within the artifact being designed. Pattern authors have connected several patterns in their collections and presented them in graphs showing patterns as boxes and relationships as arrows. They often suggested how each pattern could interact with other patterns in an “approved” way. However, it is left up to the pattern user to figure out how to incorporate patterns from different collections and in different design phases as well as which patterns correctly belong to each phase. Tool support is essential in this stage to help users manipulate and integrate patterns within a well-defined pattern-oriented dissemination process. Pattern components work as objects that offer their semantics in XML as well as java / C# classes with well-defined interfaces to communicate with the design environment, the SDE.

4 USER CENTERED DESIGN, UCD

[17], [18] discuss the two paradigms of product-oriented and process-oriented approaches in software engineering. The problem with the product-oriented approach is that the product developed is disconnected from the users work environment. In other words, no information about user work environment are collected or used in the design. This is exactly what we are trying to avoid by following process-oriented UCD approach. Process oriented approaches are designed to complement the product-oriented view by connecting program development process to the work environment (context of use) as well as the users themselves. UCD process ensures relevant functionality is properly identified from the users’ perspective, and usable design is hence created to match users’ view and expectations.

4.1 The Conceptual View of Smart Data

As we can see, the space we call “Structured Data Continuum” is rich with many methods and techniques to try to manipulate the ever growing amount of unstructured data we produce everyday. In the following work, we introduce a new concept to add structure to fully unstructured data to greatly increase human’s ability to manipulate it at large scale while still focusing on the semantics of this manipulation rather than just the syntax. Starting at an early phase of proof of concept, we use a minimal amount of details. We present an abstract view of the document first. [19] emphasize that abstract representation of a system helped designers focus on the contents, while too many details distracted them into discussions about unimportant issues. We first present the main components of our actual system as shown in Figure 3.

![Figure 3: Smart Patterns](image)

The system is intended to combine existing textual patterns with formal models to produce programmable pattern objects, or smart patterns. Expert help is needed to manually collect and rewrite existing patterns using the models. This manual help is performed at the initial construction of the dissemination system and for adding more patterns to it. In this functional view we ignored some roles. Pattern users are not an input to our workflow. They are seen as receivers of the final pattern objects. Similarly, pattern authors are strategically replaced by expert help performed on available patterns. The idea is that we cannot force pattern authors to write their patterns directly into a dissemination system using any predefined format. They usually prefer to not be bound to a specific layout. Therefore, the system allows patterns in textual formats to be rewritten using the given models without loosing
any information. They can be retrieved and restored to their original form or manipulated as software objects. Finally, system designers and system builders are not shown here as their roles are non-functional to the dissemination of pattern knowledge.

4.2 The 7Cs Process

As discussed earlier, the central aspect of process oriented approach is its dependence on a predefined process. A systematic design process instigates quality design by providing designers with a structured method to use in their design activities. In our approach, we emphasized the need for both dissemination and assimilation processes. In this paper, we present the dissemination process completely decoupled from any specific assimilation process. This allows it to offer patterns that can be integrated simultaneously in several assimilation processes. “Free patterns” that do not belong to any process at all are hard to integrate in design. Similarly, “proprietary patterns” that are specifically tailored to manually fit one design process using few specific examples defeat the main purpose of pattern generality and abstraction. We see that a pattern can be integrated in several assimilation processes by properly encapsulating its knowledge and presenting its behavior through a well-defined interface. Any assimilation process can then lookup pattern objects and select the appropriate ones using different search criteria. The selected pattern components can then be integrated in new designs or used to generate code fragments. We define the 7C’s as “a structured process to replace the huge cognitive load of manipulating HCI patterns with a dissemination system of smart patterns”. The 7C’s process identifies both logical and physical aspects of the system. A logical process focuses on what actions and activities need to be done. A physical process complements the logical process by specifying the roles associated with the process, and details who is going to do what [20], [21]. As part of the pattern reuse problem is associated with missing roles in the dissemination activities (all left to the user), the 7C’s process addresses both how these activities need to be done, and who should be doing each of them. Briefly, the 7C’s process moves gradually from current unplanned generation and use of patterns into building an automated pattern collection. The process comprises seven steps:

Collect: Place Different Research Work on Patterns in One Central Data Repository
To unify the diverse collection of pattern information, this step is the starting point, involving the creation of a central storage of patterns.

Clearout: Change from Different Formats/Presentations into One Style
Going through step 1, we identified several patterns that are dealing with the same problem from different perspectives, so in this step, we focus on consolidating existing patterns into one integrated, unified view.

Certify: Define a Domain and Clear Terminology
Answer Garden [22] route new pattern proposals to a set of distributed experts. Those experts provide feedback on the relevance of patterns, and whether they belong to the domain or not. They further determine where they exactly belong within the defined domain. We follow the same approach.

Contribute: Receive Input from Pattern Community
Experts would come up with a partial or a complete set of patterns only after they’ve spent years doing relevant work [1], [23] or have time to update an existing collection [2], [8]. By building our repository of patterns, we can help unify pattern defined by different individuals in the future. We can further identify areas where there is a shortage of patterns and declare this shortage to the associated community.

Connect: Establishing Semantic Relationships between Patterns in a Relationship Model
It has been long determined in the pattern community that a considerable portion of information associated with patterns lies in the relationships between them [24], [25], [26] rather
than in the patterns themselves. Therefore, an essential step in the process is to identify and document this relationship information.

**Categorize: Define Clear Categories for patterns that Map them into Assimilation Processes**

Based on user perspectives and their mental model as they build certain expectation from the patterns they are looking for, rather than what pattern authors might think [27], [28], [29], we build a classifications system to make patterns more manageable by the user. The main objective is to reduce the complexity of searching for patterns, or in understanding the relationship between them.

**Control – Machine Readable Format for Future Tools**

Defining formal pattern models which represent the semantics of a pattern system [30] will eventually allow for automated tool development that can process the meaning and the content of patterns rather than just text matching. The methodology, when centered around the given set of models can allow for integrating disparate tools into one integrated framework [31], [32]

5 THE SMART DATA ENVIRONMENT

While it can be seen as another “tool”, we can more accurately explain the notion of SDE as a “unifying concept” that offers interoperable models, a notation to implement these models using XML or any object oriented language, and an underlying multi-tier information system to store the essential data and to offer interaction with it.

5.1 The Role of XML in Pattern Components

XML is a standard that has been widely accepted in software industry. Every text can be rewritten in XML syntax by simply adding markup tags to explain what the subsequent piece of text is. The mere fact that a document is written in XML allows for some automatic processing to be applied to it using programming languages, while keeping it readable by humans. Simply stated, the contents of the document can be “automatically manipulated”. This is often referred to as document-oriented XML.

However, XML can offer much more than that. [33] explains that true interoperability requires not just interoperable syntax, but interoperable semantics. If the structure of the XML document was built according to a valid schema following predefined models that allow for interoperability, the capability and the power of processing the data becomes much greater as it becomes fully structured; or formal. This is referred to as data-oriented XML and is intended mainly for automatic processing, not for humans to read.

Not only is it easier to write tools to manipulate the contents of a document, it also opens the gate to a large number of available XML technologies to be directly applied to the document for automatic processing. To achieve universal data manipulation, the input to XML space is generalized into abstract data source and the output to abstract data sink. Accordingly, any data format can be seen as an “information source” that feeds into the XML space by initially transforming it into XML (Figure 4). After the transformation phase, the document can be rendered into the format of any “information sink” at the output, or redirected back into the XML space for further transformation. This allows several XML technologies to be pipelined in a modular way to generate arbitrarily complex processing and rendering schemes. The enabling key into this XML space is to represent data in a properly expressive semantic format that displays the desired structure and behavior in it, regardless of any specific application. This can be referred to as
“smart data” contrary to smart applications. In smart applications, data is created through- and belongs to- a specific application (e.g. Excel spreadsheets, MS Word or an Adobe Acrobat document). Data is represented internally in different proprietary formats and are accessible only by running the “owner” application. Transferring data from one application to another requires knowledge of all details of proprietary formats of both applications. For m applications, this could mean m*(m-1) transformations, a tedious and unpractical approach. Smart data, on the other hand, present all semantics within data itself in the common, non-proprietary format of XML. Any application or tool can be seen as an information source that exports its proprietary data once to XML space. Similarly, applications can be seen as information sinks, and XML data can be imported by them from the XML space. To cover all possible transformation, this needs to be done only twice for each application; in and out of XML space. For m applications to fully communicate, the total number of transformation is reduced to 2*m. For a large m, this is a significant saving. Following this concept, we use XML as the central representation of pattern components, and we use tools to automatically transform them back and forth between object-oriented classes, text as well as other XML based languages.

6 CONCLUSION AND FUTURE WORK

The current process of pattern reuse is a simple process of publishing numerous patterns using different media and leaving it up to the users to do their best in figuring out how to find and use them. We suggested an addition to this concept to help define and standardize the process of pattern dissemination and assimilation, which can lead to an effective reuse of the knowledge contents within patterns. We proposed to represent patterns as software objects that encapsulate pattern semantics and allow interaction with them through their interfaces. We have developed several prototypes under two broad categories, namely as an online digital library with universal contributions, and as a personal digital library on personal computers for small scale pattern collections. In both categories we prototyped several options at the database level (persistent layer), the processing level (tools), and the interface. The models and the associated notation can be adopted by other users and tool developers to enhance the system. We are conducting empirical study to test and validate the system we developed and we are gradually evolving it into a mature system.

7 REFERENCES


