The Failure and Success of Unstructured Data Reuse
a Pragmatic Analysis

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

Eman Monir Darwish
Dept. of Fine Arts
Helwan University
Cairo, Egypt
eman.monir@hu.edu.eg

ABSTRACT
Structured data is easy to parse and process using simple software tools with effective results. However, most valuable information is typically found in unstructured data. The problem is that unstructured data is significantly harder to process, especially at large scale. Adding some structure can greatly improve data reuse. To demonstrate our approach, we focus on one particular example, namely the design patterns, a common encapsulation mechanism, widely used in several domains to capture and reuse valuable domain experiences. Patterns are growing in numbers as an effective way of communicating knowledge among expert designers. Currently, there are numerous domains where patterns are applied, including HCI - Human Computer Interaction. HCI Patterns are typically published in books or on the Internet. The sheer number of HCI patterns, the rich and heterogeneous contents coupled with the lack of a delivery system can confuse and overwhelm a novice pattern user, even when they are meant to help novice users in the first place. We present an approach to add structure to HCI patterns, hence providing the ability to communicate and reuse them in a more scalable and effective way.

KEYWORDS
HCI patterns, information encapsulation, semi-structured data, information dissemination, information reuse

1 BACKGROUND
The first use of patterns can generally be attributed to the work of Christopher Alexander [1], [2], [3]. It is often claimed that in order to get the essential spirit of patterns, we should take a look at his work [4]. The next milestone is attributed to the renowned book of “Design Patterns” [5], widely acknowledged in the software design community. Plenty of other patterns emerged since then, all of which are generally produced by expert designers, and typically targeting the novice designers. They aim to provide solutions to common design problems in different contexts. This plethora of patterns started to overwhelm the users as to how to find and use this ever-growing number of patterns. So far, most efforts have focused on “generating” more patterns, but they stop short of addressing consequent phases like delivering patterns to their appropriate destinations, and helping in how to effectively integrate them in a new design artifact. These essential activities are left to the user, which would consequently follow ad hoc techniques to lookup as many patterns as they can, and then try to figure out how to remove redundancies, inconsistencies and conflicts between them before attempting to put them together in a new design. The problem has long been discussed and analyzed in the pattern community [6], [7] and [8]. Some tools have also been developed [9], [10], [11]. Nonetheless, this remains a typical challenge of information management that has not been adequately addressed in the pattern community before, despite the fact that elegant, well-established concepts already exist in the database domain. Besides the efforts of adapting these concepts to the specific domain of patterns, the gap between conceptual and practical aspects needs to be bridged in the form of a software system that implements these concepts by following a structured pattern dissemination process.
2 SEARCH BY SYNTAX: A RANKED GUESS WORK

In this work, we use HCI patterns as an example of large, unstructured, multi-modal, and heterogeneous data collection, which is generated with the intention of being reused to help designers, so it has a complete lifecycle of generation and consumption. Previous usability tests have shown a deficiency in how HCI patterns are reused (for example [12]). In this approach we consider three stages of its lifecycle: 1) How they are generated, 2) How they are communicated, and 3) How they are reused.

A key aspect of our work is the kind of structure that needs to be added to the pattern data to enhance its structure and render it more usable. To elaborate on this concept of what structured data is from our point of view, we define the Structured Data Continuum concept as per Figure 1.

![Structured Data Continuum](image)

**Figure 1:** Structured Data

On the far right end of scale, fully structured data can be manipulated at large scale at a smart level, but they are expensive to create, and are more suitable for homogeneous contents. They render themselves easily to be built as normalized database tables, a set of regular tables, or even a structured, itemized, or indexed text document.

On the other extreme of the scale, fully unstructured data have free-flowing form-less, rich contents, with multiple non-textual components like images, graphs, tables, audio or video. While these mixed contents have heterogeneous presentations, they most often have a homogeneous meaning and semantics, as well as common contents. This makes them easy to comprehend by humans and even fun and easy to read as the human reader looks for the semantics and meaning of the contents. On the contrary, they could be a significant task for software to process since software typically looks at and compare syntactic contents only.

Many workarounds can be done to manipulate the underlying meaning of data based on its syntax. We present two simple examples to demonstrate:

**Example 1:** We consider a simple phrase like “Mr. Fox is driving a Jaguar”. Humans can immediately identify “Fox” as a person’s family name, and “Jaguar” as a popular car brand. Computers can have hard time doing that without additional help and probably sophisticated algorithms.

**Example 2:** We consider the case of searching for a specific picture in a large picture repository. Computers can manipulate digital pictures with ultra high precisions represented as millions of pixels with millions of color variations per pixels. On the other hand, computers have hard time identifying what each picture really contains, or comparing two similar pictures if they slightly vary in unnoticeable details, zooming level or a different angle without any significant change to details. In this case, advanced imaging algorithms, pattern recognition and matching, fuzzy logic, or multi-dimensional search trees with “nearest neighbor” methods [13], [14] can be used, relying on the mathematical pattern similarities between images with limited applicability. These solutions are highly sophisticated and expensive, with limited success outside their specific applications.

One workaround to processing the actual picture contents with less complex approaches is to identify the “meaning” of the picture contents by generating textual metadata and associating it with the pictures, which could generally describe the contents of the picture in a computer-readable way. Using simple keyword search of the metadata can give acceptable results. The main assumption of this approach is that there is sufficiently high correlation between the keywords
present in the textual metadata and the actual contents of the picture. In this case the meta-data writer has to come up with sufficiently descriptive set of keywords to describe what the picture contents are. Equally important, the user (searching for particular pictures) has to come up with sufficiently descriptive set of search terms of what they’re looking for in a picture, and hope for good matches with the existing metadata terms, or at least a common overlap between the two sets. This solution fails when metadata is not descriptive enough or absent altogether resulting in low or no correlation. The failure can also result when the user is unable to find descriptive enough keywords or not sure what they are exactly looking for. We can further argue that this is a way of redirecting the core search activity into a correlated search of auxiliary contents rather than the original one. Users are basically searching textual information (the metadata), and not the original data, and the pictures we get are dependent on the quality and expressiveness of the auxiliary text as well as on the search keywords, and not on the original contents.

A newly emerged, more simplified approach to access pictures on the web while avoiding the additional cost of creating metadata is what we can consider as large-scale, low cost “pseudo-metadata” generation for Web-posted images. The idea is to scan all pictures available on the web, and to simultaneously scan all accompanying text, then use the text as a low fidelity, low cost metadata rather than generating new, high fidelity, high cost meta data for each image. The simple idea is that if the document has the word “dog” then the images in the same document probably are related to dogs as well. Further ranking criteria like the number of occurrences of the word, the location(s) in the document where the word was used (header, body, caption, etc.), can refine the returned images and rank them for higher fidelity, improving the hit rate.

We used Google Images to verify this concept. We ran a search for a simple keyword “Dog”, and got several hundred images of dogs in all contexts imaginable and unimaginable. We identified few irrelevant images. In one case of wrong hits was of a person named “Maddog”, and the pseudo-metadata algorithm wrongly identified it as dog-related picture, while the actual picture and all the textual contents had nothing to do with dogs. Another case was a mug shot of a person. As we accessed the original document, the title of the document read “Florida linebacker Antonio Morrison arrested after barking at police dog.” Source: New York Daily News, read more: http://www.nydailynews.com/sports/college/florida-lb-arrested-barking-police-dog-article-1.1404972#ixzz2eWxm5TyB

A third case was also a mug shot of a person (Figure 2) who was accused of trying to blow up a family dog http://www.koin.com/2013/08/05/deputies-man-attached-explosives-to-dog/

In all cases, the search did not produce a dog image as desired. The key issue here is that the metadata (the accompanying text in this case) did include clearly related syntax of a “dog” in all cases, and clearly related semantics of “dog”-related issues in the latter two. However, the search still failed to produce a dog image, as would a human search.

As we can see, the use of accompanying text to guess on the contents of an image can work most of the time, but it remains a mechanical workaround, and the success rate goes dramatically down with more complex or less descriptive keywords.

Similarly, as discussed earlier, while the comparison of similar but pixel-imperfect pictures can be manipulated using fuzzy logic and non-traditional database approaches with acceptable results, the problem remains as to fully describe the contents of all contents of a complex picture.
by a computer in the same intuitive way humans do.
Other approaches to manage unstructured data can be found at Google maps, where they first captured millions of pictures of earth at different heights from ultra high altitudes all the way down to street views, then hired an army of people to manually review, fix, seamlessly connect, and integrate them into one 3-dimensional continuum. In this continuum, you can move in 2-D (north-south and east-west) as well as at the 3rd dimension of zooming in and out. Google further annotated the contents with a gamut of information from street names and directions to related photos and street views. By adding all these integrated structured details, the original pixel-generated images became highly processable by a large number of software application; a perfect example of data reuse.
Amazon.com used a different approach to adding structure to their documents. With the help of millions of users, they annotated the original books contents with additional end-user and system-generated, highly structured contents (e.g. star-ranking, user feedback, related book recommendations, people who bought/looked at this book also bought/looked at those books, etc.). Further analysis shows an apparent conversion between the two approaches. While Google relied heavily and mainly on hired human force to integrate and annotate data, they later also used the million of users to upload their pictures and used them to further annotate the maps by linking the uploaded pictures to the location where they were take.
Similarly, besides Amazon’s heavy reliance on user-created contents on available books, they also included their own compilation of book contents like publisher information, contents scans and chapter examples, and integrated them with the user-created ones.

3 HOW WE SEARCH

Besides the data creator perspective (the data source), we also need to look at the other side, the data-receiving end (the data sink). From a user perspective, we can distinguish between different types of search, depending on user’s familiarity with what they are searching for. The three main categories of search can be described in the following model:

i) Search to Locate: In this category, the user has previous familiarity with an object, as to what it might look like, or might have already seen the same or a similar object before, and they are just trying to locate it. One example is if you’re looking for a particular camera; either one that you’ve seen before, or one with specific attributes in mind. The user will be looking at specific item collections and will be using specific key words. The search will probably stop as soon as the item was found. This kind of search is typically the easiest one to execute, and the easiest one to prepare data for. Same scenario applies when a researcher is looking for a particular research paper or a report that they are already familiar with, and hence the activity of search is focused on trying to locate the sought object. The initial information of search term can be exact words or phrases, or key terms. Search algorithms can help execute this kind of search even in the extreme case when the user cannot remember any exact terms, in which case multiple iterations of search and hit can help the user find closer terms and often locate the object eventually. This type of search typically has few iterations and short search duration.

ii) Search to Find: In this type of search, the user does not have a particular object in mind, but rather have a general idea of what a good object might look like. One example is when you’re looking for a good camera, but do not have any make, model, or specifications in mind. In this case, the user will probably search for different attributes, and repeat the search process several times. The user will typically browse through multiple search results, and keep changing and refining their results. Eventually, the user starts to focus their research terms as they continue, and will probably need to compare features and attributes of multiple items. For that reason, several sites offer the users’ the ability to put multiple items side by side, and compare them in one view. This feature is common in several structured and comparable items like cameras,
cars, and the likes. This search will typically not end as any particular item was found; the user will typically keep searching, comparing, and refining their search. This type of search can have several iterations and typically takes longer than the first type of search.

iii) **Search to Search:** In this type of search, the user is not sure what they are exactly searching for, and typically has no concrete starting point, but rather a vague idea of a topic. Users will probably start searching using a wide range of search terms and incoherent keywords, and then browse through the search results.

As the search progresses, the user would typically have identified more specific search terms and search topics and will use them to further refine their search and get closer to building an initial idea of a more concrete research and will have a specific set of coherent keywords. An example comparable to the previous two types is when the user is trying to search for a birthday gift, but not sure of what it might be. This is also a common case in academia when a researcher is attempting to identify general work related to the latest software design of—say—mobile applications but not sure of any specific research or design work, or a particular research group. The researcher will typically use general keywords, and as some research work is identified, the user will start educating themselves about the topic, building a coherent set of search terms and keywords. The refinement of this set results in specific keywords that are highly relevant to the now-specified area. After multiple iterations, the user will have built a concrete idea and have moved from a vague description into a more, well defined one. This kind of search is often performed as a starting point, and may eventually mature and change into a “search to find” type of search.

### 3.1 Search Lifecycle

While the three search types are stated from more specific to more generic, (categories i through iii in the previous section), we can more logically connect them in a progressive lifecycle from the most generic search (browse) to the most specific search (select) as per figure 3.

![Figure 3: Search Activities](image)

As the search progresses, the user would typically have identified more specific search terms and search topics and will use them to further refine their search and get closer to building an initial idea of a more concrete research and will have a specific set of coherent keywords. An example comparable to the previous two types is when the user is trying to search for a birthday gift, but not sure of what it might be. This is also a common case in academia when a researcher is attempting to identify general work related to the latest software design of—say—mobile applications but not sure of any specific research or design work, or a particular research group. The researcher will typically use general keywords, and as some research work is identified, the user will start educating themselves about the topic, building a coherent set of search terms and keywords. The refinement of this set results in specific keywords that are highly relevant to the now-specified area. After multiple iterations, the user will have built a concrete idea and have moved from a vague description into a more, well defined one. This kind of search is often performed as a starting point, and may eventually mature and change into a “search to find” type of search.

![Figure 4: Search Lifecycle](image)

Figure 4 depicts the progression between the three moods of search and some associated artifacts.

### 4 THE SYSTEM DESIGN

To successfully reuse pattern, we focus on an integrated approach of improving the full lifecycle of data from generation to dissemination to reuse. To validate this approach, we constructed an initial conceptual design with a detailed proof of concept to demonstrate the work. Our system is based on an integrative approach of pattern dissemination that complement the scattered efforts of writing patterns. Following a proof of concept, we built a delivery system in the form of a digital library based on a database of patterns, a transformation logic for any desired algorithms, pattern processing and transformation tools, as well as an interface to offer these functionality to the user. We selected XML as the language to
represent patterns for several reasons. XML is becoming the norm for data interchange, and is used extensively in IT systems. Besides the XML language, Several XML-based technologies are emerging to enhance XML-compatible information systems.

The system architecture is shown in Figure 5. The data flow starts at the top right corner in the form of the existing pattern collections. Looking at the data pathways depicted in the figure, we see that the input information is now bypassed from the direct path (from pattern collections directly to user preparations block, marked as A) into the system pre-storage phase, the pattern corpus, and to the rest of the system; marked as B

The User Preparations block, depicted as pathway A, represents the cognitive activity by the user to search for patterns and read through the text, analyze its contents, and figure out how to apply which patterns in the design. This is a fundamental observation in our study. We can evaluate the difference between the two processes, referred to as pathways A and B. If we considered them as representing the dissemination processes between pattern authors and pattern users, we can use the Capability Maturity Model, CMM, of the Software Engineering Institute to briefly evaluate them.

-Process A: This process does not follow a particular approach of dissemination except for relying on users preparations (looking up patterns, understanding them, and applying them in an ad hoc approach). We evaluated this to be a CMM level 1.

-Process B: As suggested by the envisioned system, there is a process in place to help users interact with patterns in a structured way. Moreover, this process relies heavily on feedback, and is constantly changing, as shown in implementing and validating the 7Cs process within the system [15], [16], a CMM level 4

The main modules of the system, as shown in Figure 5 are:

- The Pattern Corpus, a raw collection of patterns before being reformatted and saved in the XMLDB.

- The Data Models, used in the semantics of the XML Rewrites

- The Expert Preparations, used as help in rewriting pattern information to ensure the integrity of the contents and avoid redundancies and inconsistencies.

- The System Process (the 7Cs process), a structured method applied across the system

- The XML Subsystem (XML rewrites, XMLDB and XML semantics/scheme), The core constituent of the system, and the backbone of the three-tiered design.

- The Processing and the Presentation Layers are the middle- and front-tiers respectively.

One of the main aspects of our system is to be able to automatically process the contents of patterns to alleviate the user from this cognitive load. The main purpose is to have a scalable capability to effectively process large number of patterns. Representing patterns in natural language defeats this purpose. XML is much more suitable in this regard due to its machine-readability. It is based on the fundamental concept of automatic processing. Goldfarb [17], the inventor of SGML (the parent, and superset of XML) explains that the vast majority of XML documents will be created by computer programs, and processed by other computer programs, then destroyed. Humans will never see them. The first step is to rewrite patterns in an XML format. A simple XML syntax rewrite (like PLML, Pattern Language Markup Language, [18] can be a small step in this direction, but -by itself- it will not do much good as will be discussed later. To achieve global interoperability, we needed to design the semantics and the behavior modeling behind the XML syntax, according to concrete data models.

Database is a core constituent of the system. An obvious choice for data store is an XMLDB. We implemented part of the system using it. The steps are as follows:
Phase 1, **No-Database System**: By temporarily including the data inside the system (hard-coding the data). We have two major prototypes: In the UPADE project. We developed a systematic approach to glue patterns together, support the integration of patterns at the high design level and automate pattern composition. UPADE [19] generated program elements from an extensible collection of "template" patterns.

In the second prototype, [20], [15], we experimented with prototypes of the interface aspects, and we also hard-coded patterns inside the prototypes. At this stage we were able to test and refine the design and functionality of the system.

**Figure 5**: System Overview

Phase 2, **Flat-File-Database**: To start building an independent database module, external to the system, and to connect it to the rest of the system. We used “XMLSpy” to prototype the XMLDB concept. The XMLSpy allowed us to build an actual, partially functional XMLDB based on flat files as the internal storage medium. The major advantage of this approach is the extreme simplicity of the system, which allowed us to further refine the fundamental concepts of the system without being carried away by the implementation details. The main drawback was its limited scalability and unacceptable performance, as will be explained in step 4.

Phase 3, **Web-based System**: To implement the functionality of the system as built around external XMLDB on a Web-based environment. We implemented several aspects of the system using the database we have from phase 2. Using 3-tier system architecture (data-, logic-, and presentation tiers), DHTML and Web technologies (CGI), we built a web-based distributed system to test our XMLDB. We implemented the system to interact with pattern users by offering a search interface at the presentation tier.

As the user queries the system for some patterns, the system middle tier (at the web server) processes the request by accessing the data tier at the back end, and returning results by dynamically generating a web page with search results.

The 7Cs Process

One of the main aspects of process-oriented approaches is the dependence on a predetermined process. Gaffar [21] describes the “7Cs Process” as a systematic approach to adding structure to rich but ultra heterogeneous data. An established design process instigates quality design by allowing designers to follow structured methods in their activities. In our approach, we emphasized the need for both dissemination and assimilation processes. In this paper, we briefly summarize the 7Cs dissemination process as completely decoupled from any specific assimilation process [6], [9]. 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. The 7Cs is “a structured process to
replace the huge cognitive load of manipulating HCI patterns with a dissemination system of smart patterns” [21]. The 7Cs 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 [22], [23]. As part of the pattern reuse problem is associated with missing roles in the dissemination activities (all left to the user), the 7Cs process addresses both how these activities need to be done, and who should be doing each of them. Briefly, the 7Cs process moves gradually from current unplanned generation and use of patterns into building an automated pattern collection. The process comprises seven steps [21]:

Step 1) Collect: Place Different Research Work on Patterns in One Central Data Repository

Step 2) Clearout: Change from Different Formats/Presentations into One Uniform Style

Step 3) Certify: Define Domain Boundaries and Clear Terminology

Step 4) Contribute: Receive Input from Pattern Community

Step 5) Connect: Establish Semantic Relationships between Patterns Using a Formal Relationship Model

Step 6) Categorize: Define Clear Categories for Patterns that Map them into Assimilation Processes

Step 7) Control – Build MachineReadable Format for Software Tools

The ultimate goal of the 7Cs process is to allow designers to use the leverage of tools to process large number of patterns efficiently, and to be able to assimilate them effectively into a new design. Patterns that are both human- and machine-readable are the main outcome of the process of pattern dissemination and the first step towards assimilating them.

DATA MANIPULATION TECHNIQUES

The XML technology is evolving in an rapid pace. Just few years ago, as we started the project, the XMLDB was not supported by Oracle, an industrial norm in databases; or by Apache, an industrial norm in Web servers. The current and future trends are now determined by the new development, as explained in phase 4 of the system development.

-Phase 4 is to use a more advanced, fully-fledged XMLDB to overcome difficulties we discovered in phase 3. XMLSpy internally implemented the XMLDB as flat files. While invisible to the user, this solution is not scalable. It relies on the file structure of the underlying operating system, and is affected by its performance. Besides, inserting to- or deleting from the DB incurs the penalty of rewriting the whole file. The other major obstacle was that some XML technologies we used are still in their early specification stages, and are immature for full implementation. For example, XQuery is the query language supported by XMLDB. We used it to query our XMLDB. While easy to use, it still does not support insertion or deletion, so we had to implement them manually in phase 3.

In phase 4, we are rebuilding the database to be less dependent on immature XML technologies while still offering all XML functionality upfront. The idea is to use newly emerging commercial- or open source systems that are robust enough to support the full functionality of our XMLDB pattern system, while delegating the internal database implementation to the system, be it native- or non-native XML database. Native XMLDB offers an XML interface to the database, and store the database internally in pure XML format. Non-native XMLDB offers an XML interface to the database, similar to the native one, but transforms it internally to relational database, or to a proprietary database format before storing it physically on the hard disk.
Among the hundreds of commercially available XMLDB today, we compared three common alternatives:

1- ORACLE RDBMS with support to XMLDB. An authoritative commercial application that offers a fully functional XMLDB interface, while translating it internally to an RDBMS (a non-native XMLDB)

2- ZOPE, an open source industrial application that implements XMLDB using internal proprietary RDBMS, but provides fully functional XMLDB interface (also a non-native XMLDB).

3- Apache-Xindice XMLDB, a powerful open source native XMLDB technology, supported by Apache (one of the best known and used web servers).

CONCLUSION

Patterns are useful data encapsulation mechanisms that can help UI designers improve the quality of their work by reusing well-known practices of domain experts. For our purpose of building a proof of concept for programmable data contents, they serve as a good example due to their nature of highly unstructured presentation and highly heterogeneous multi-modal contents. They also rely heavily on reuse —at least in principle—allowing us to focus on the amount of reuse as an indicator of success of dissemination. In the pattern community, knowledge dissemination activities focus mostly on generating patterns, with little or on plan on how to deliver them or how to reuse them in new designs. The proliferation of HCI –Human Computer Interaction- patterns is associated with redundancies and inconsistencies that often confuse and overwhelm the user. More traditional approaches like database domain have provided methods to deal with these problems by allowing data to be organized and reused efficiently using normalized database tables and structured queries. We have proposed and built a proof of concept to complement the process of generating patterns by reducing inconsistencies and conflicts, and by providing patterns in a reusable format. We have designed a process that can help in the assimilation and dissemination of patterns. We have implemented a system that —in applying the process— will help novice users receive patterns in an orderly fashion and be able to effectively use them in their design artifacts. The system relies on an underlying XML database and the process will help feed the patterns into the database and rewrite them in a semantically reusable format. The database and the system allow us to add tools to the interface to automatically process patterns in a user-friendly design environment and to connect them to other XML- or UML-based tools.

7 REFERENCES

5. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, Boston, Massachusetts, USA. (1995).