Document Engineering for e-Business

Robert J. Glushko
School of Information Management and Systems
University of California, Berkeley
+1 650 533 4369
glushko@sims.berkeley.edu

Tim McGrath
School of Information Communication and Technology
University of Notre Dame Australia
Fremantle
+61 893352228
tmcgrath@portcomm.com.au

ABSTRACT

It can be said that “document exchange" is the "mother of all patterns" for business (and for e-business). Yet, by itself this view isn't sufficiently prescriptive. In this paper, we present additional perspectives or frameworks that make this abstraction more rigorous and useful. We describe an approach to artifact-driven analysis, model refinement, and implementation for document-intensive systems that unifies the “document analysis" approach from publishing and the "data analysis" approach from information systems. These traditionally contrasting approaches to understanding documents are unified in an "Analysis Spectrum" in which presentational, structural, and content components assume different weights or status. Our methodology emphasizes reuse with a "Reuse Matrix," in which both business process (or document exchange) patterns and document schema patterns are organized by different levels of abstraction and scope. Enterprise-level patterns like "supply chain" and "marketplace" can fit into this matrix along with process patterns like "RosettaNet PIP" and document patterns like the "XML Common Business Library." Taken together, these concepts form the foundation of a new discipline: “Document Engineering for e-Business.”

Categories and Subject Descriptors

I.7.5 [Document and Text Processing]: Document Capture – *document analysis;* I.7.2 [Document and Text Processing]: Document Preparation – *markup languages;* H.2.1 [Database Management]: Logical design

General Terms

Design, Standardization

Keywords

XML, Document Engineering, Document Analysis, Business Process Modeling, Patterns, Reuse, e-Business

Doc Eng ’02, November 8, 2002, McClean, Virginia, USA.

1. INTRODUCTION

It is natural to conceptualize business relationships as a chain of document exchanges. Businesses have long dealt with each other by exchanging documents like catalogs, orders, invoices, and receipts. For example, in 355 BCE, when an Aramaic farmer named Halfat paid his taxes he was issued a receipt imprinted on a clay pot. Over 2000 years later, we have a record of Halfat’s business transaction, frozen in time on a shard of pottery [4].

Figure 1 An Aramaic Tax Receipt

Even though pottery has been replaced by paper and paper has mostly been replaced by electronic documents, the idea of documents as interfaces has persevered. Using documents as interfaces enables a business to present a clean and stable relationship to its business partners despite changes to its technology or internal business processes. Today, as more and more business has become “e-business,” electronic documents hide the specific implementation details of e-business applications and web-based services.

For example, if Business A sends a **purchase order** to Business B and B can fulfill it, B might respond with an **purchase order acknowledgment**, or perhaps with an **invoice** and a **shipping notice**. As long as A and B can understand each other's documents and can produce and respond with the documents appropriate for each other's business processes, they need not reveal how they produce these documents they send nor how they process the documents that they receive. The documents - and only the documents - serve as the public interfaces to their respective business processes.

This loosely coupled architecture reduces the cost of designing and implementing new applications, which is essential as e-businesses experiment with business models. "Document exchange" is the "mother of all patterns" out of which marketplaces, supply chains, and other more complex patterns are composed. Electronic documents are thus the foundation for the compelling vision of a "plug and play" Internet economy ([5],[6] in which "virtual enterprises" or "information supply chains" [3] are created by building on and interconnecting services offered by businesses around the globe.

For example, an Internet business like Amazon.com's can be modeled using a "direct to customer" or "drop-shipment" pattern [13] that coordinates the activities of four enterprises: a retailer whose catalog offers products to customers, a supplier or warehouser who maintains product inventory, a bank that accepts the customer's payment, and a shipment service that picks up the product from the warehouse and delivers it to the customer. While this coordination may be invisible or appear seamless from the customer's perspective, it requires a complex choreography of document exchanges.

Furthermore, with careful design the same documents can be reused in different business processes. For example, the simple bi-directional exchange pattern of **purchase order <=> purchase order response** can itself be a component of an "auction" pattern that consists of iterations of multiple simultaneous exchanges with a single seller or single buyer (for reverse auctions).

But even if we can conceive of business models in terms of document exchanges, by itself this view isn't sufficiently prescriptive. We need a more complete perspective on document engineering that answers questions like these:

- How do we identify, specify, and deploy the appropriate documents?
- How do we preserve our investments in older technologies for document exchange while taking advantage of new ones?
- How do we preserve our investments in business processes and relationships while creating new ones?

These are not new questions, but the discipline of "Document Engineering for e-Business" is emerging as a more comprehensive and coherent approach to answering them. This new discipline for designing, specifying and deploying the electronic documents can apply to all business models from speeding the flow of information through supply chains, to hosted e-marketplaces and auctions to the aggregation, customization, and syndication of content. These documents allow us to automate end-to-end transactions and apply Internet technologies for dramatic economic benefits.

2. Analysis Foundations for Document Engineering

Published documents are essential to business. Documents like catalogs, brochures, and datasheets assist buyers in locating and selecting products and services. Other types of documents like assembly instructions, reference manuals, and troubleshooting guides assist buyers in using and maintaining the things they have bought. Transactional documents like orders and invoices, traditionally embodied as printed business forms but more commonly now as electronic messages, directly invoke business processes or respond from them.

2.1 Document-Centric Analysis

Because of the essential role of documents in business, it is natural that document analysis will form a major part of any document engineering methodology for e-business. Document analysis techniques developed by publishing experts emphasize the study of published documents as artifacts that are perceived as a rendition – a combination of information and format ([8],[9],[15]).

This kind of document-centric analysis has been central to the design of text processing, publishing or hypertext systems [7]. Its primary focus has been on printed publications and more recently, web page layouts. While the distinction between information and format may be blurred by WYSIWYG user interfaces, various markup languages have been developed that make it possible to separate document content from the format. This gave rise to the insight that markup could be used to facilitate functions other than formatting, such as classification, retrieval, and reuse.

Document analysis is often conducted with the goal of abstracting a logical model from heterogeneous instances and encoding it as an SGML or XML schema. The schema enables the replacement of ad hoc, inconsistent or incomplete formatting with a stylesheet that applies presentation semantics in a consistent fashion to any instance that conforms to the schema.

2.2 Data-Centric Analysis

But increasingly the documents most important in e-business are not traditional publications but data-centric electronic messages. Such documents are far more regular in their logical structures and have minimal or arbitrary presentation features. They are optimized for consumption by applications and not for people. For these kinds of documents, the traditional document analysis methods are not well suited. Instead, we have turned to data analysis methods from information systems engineering for description and design techniques that we can reapply to the document domain.

Data-centric analysis has its heritage in information systems design. Here, the focus is on computer data files and databases. Skills developed from crude hierarchical data models to more sophisticated relational models that described associations between data objects. The principles of normalization and relational theory evolved and ensured that database schemas minimized redundancy and prevented inadvertent errors or loss of

information. At about the same time the insight emerged that methods and processes could be associated with data objects to create an object-oriented view.

2.3 Unifying the Document-Centric and Data-Centric Perspectives

The document and data analysis perspectives come from different disciplines, use different tools, terminology and techniques and arguably different cultures. Both offer valuable insight into designing effective documents but until now they have had little intersection.

The challenge is to unify these two schools of thought into a common approach. We seek to describe documents with formal models comprising of data objects that have well-defined properties and associations with each other. Our methods must yield coherent and consistent definitions across the spectrum of business document types. The synthesis of these two perspectives is the essence of Document Engineering.

3. The Document Engineering Approach

In Document Engineering when we look at e-business systems and services the primary objects in our models are the documents and the components from which they are assembled. We have taken a classical analyze-design-refine modeling methodology and reshaped it to better fit the domain of documents (Figure 2).

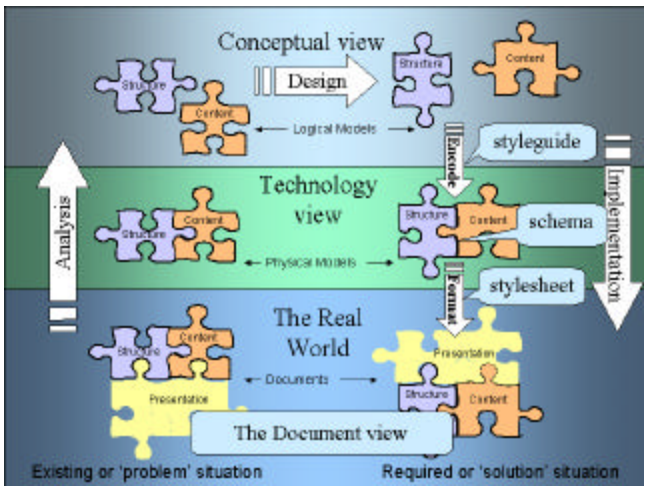

Figure 2 The Document Engineering Roadmap

3.1 Document Models

We analyze real world artifacts by creating models. These may be mental image pictures, scale replicas or architectural diagrams. We populate these models of documents by capturing meta-data, descriptions of the things that make up the model.

Because of the distributed architectures of e-business and web services we need a modeling approach that is consistent with them - loosely coupled and non-prescriptive. For businesses to "do business" their business systems must interoperate, and thus their models must also, so an efficient modeling approach actively strives to reuse and revise rather than to reinvent.

Document Engineering models of e-business systems describe the business processes that produce and consume documents. These processes convey the context in which our document structures and their components exist. For example, the details of a product on a Purchase Order sent by a supplier in a vendor managed inventory business process may contain less descriptive information than is required on a Purchase Order sent by the buyer.

Finally, it is both a theoretical and practical innovation in Document Engineering to use XML schemas as the format for encoding models for both the documents and the business processes in which they participate.

3.2 Classifying Components

When we look at a document, the "fodder" for its logical model comes from three categories of information it contains:

- *Content* components – the pieces of information in the document; the "what is it" information, or the "gray matter"
- *Structure* components – the arrangement of the content, the "where it is" information, or the "skeletal matter".
- *Presentation* components – the formatting or rendering of both structure and content components; the "what does it look like" information; much of the time it "doesn't matter" except as cues to help identify components of the other two types.

Analyzing documents in terms of these three kinds of components can be subtle. For example, a **DateOfBirth** is a content component meaning, "the date on which you were born", but is also a structural component that contains a **DayOfMonth**, **Month** and **Year** component. Finally, there may be one or more presentation components that apply to the other components. **DateOfBirth** might be formatted as 11-01-1980, but Americans understand this as MM-DD-YYYY and Europeans as DD-MM-YYYY, which illustrates that presentation rules can involve both structure and content.

The hybrid or composite skill of component analysis in Document Engineering combines the skills of the data-centric data analyst, who focuses on content components, with the skills of the document analyst, who traditionally focuses on structure and presentation. Together they cover the Document Analysis Spectrum.

3.3 The Document Analysis Spectrum

Our synthesis of document-centric and data-centric analysis implies a broad scope in the kinds of documents that can be analyzed. These can be viewed on a continuum known as the Document Analysis Spectrum.

At one end of this spectrum are idiosyncratic or one-of-a-kind documents whose presentational components are highly designed, which makes them artifacts requiring careful study. For example,

the *Oxford English Dictionary* uses a complex set of fonts, type sizes, and formatting attributes like boldface and italics to distinguish the sub-structures of each entry [10]. Likewise, the *Engineering Data Compendium*, an encyclopedia of human factors in design, is a complex and concrete merger of presentational and structural information, with a two-page facing spread for each entry and numerous layout and typographical conventions [7].

At the other end of the spectrum are data-intensive electronic documents as used in e-business – transaction documents. Here, content becomes all-important and each document follows a well-defined and regular schema. Structural components are often just pure "containers" for content components, and presentational components are less important

In between these endpoints are documents that exhibit regularity in data content but for which presentation remains important. Product catalogs or lecture slides are good examples.

The challenge for a methodology of Document Engineering is to be able to analyze documents at all points in the spectrum. .[1]

3.4 Analysis of Documents

Initially, our document models will be tightly coupled to the physical implementation as it appears in the artifacts being studied. This will be probably be influenced by the technology involved in its production. For example, if the printed document artifact we are analyzing has a three line address description, it seems natural to model this as three lines of address description. Yet this structure may have been determined by the space available on the original form.

The models that contain these technological constraints or features are the "physical" models. Physical models reflect the technology used to implement the documents or processes. This technology view shows HOW things work. As we progress to a more logical model, we remove presentational features. For example, while the content and the structural components in **DateOfBirth** matters, the formatting of the three fields is not instrinsically significant. "November 1st 1980" is the equivalent to "11-01-1980" and "the first of November 1980".

Good analysis encourages us to look beyond the physical model, to ask WHY things work. This is the conceptual view, not constrained by technical features. Here we can see beyond the three line address description constraint and recognize address descriptions as a concept. We want to look at the concepts behind the component, to find out why is it doing these things. These results represent 'logical' models. This is shown in Figure 2 as the move to a higher layer of Analysis on the left hand side of the diagram. Now we try to separate content from structures, to establish their independent meaning. For example, why is "11-01" just a different presentation for "the first of November"? Because each contains a **DayOfMonth**, **Month** and **Year**? Because each of these components has meaning outside the **DateOfBirth** and can be used for other purposes.

3.5 Document Design

Experience tells us that defining these conceptual models is where we start to understand the true nature of something. It is this understanding that leads to the possibility of improvement that is design. Indeed, at several points during our analysis a voice cries out "There must be a better way". It is when we study the logical model of the existing system that we start to formulate what that better way may be. This may mean removing redundant processes or data, standardizing on one process or rationalizing a document's structure. In Figure 2 we are now moving across the top part of diagram as part of the Design process.

Well-engineered document schemas have clear, unambiguous definitions of data, a recognition of the logical sets (or containers) in which they belong and the way these sets are related to each other. These definitions allow us to minimize redundancy, localize dependencies and ensure that information can be maintained in logical sets that reflect the constraints of the real world.

Defining the reusable data structures in documents is something that can be done intuitively. It might sound right to group **Name**, **Address** and **DateOfBirth** into a **Person** container. However, if we want to have strongly re-usable structures we need a more formal and consistent approach for grouping components.

3.5.1 Normalization

Conventional data modeling practices include formal rules for designing logical structures. In fact, much of what document analysts have done in the past, albeit informally, is establishing what data analysts call functional dependencies. In Document Engineering we apply the same rigor to document schema design that we have customarily applied to database design.

Functional dependency means that if the value of an attribute changes when another attribute value changes, then the former set is dependent on the latter. For example, suppose the price per sheet of printer paper is reduced if the pack size changes from reams to cartons. This means pricing per sheet is functionally dependent on pack size. The values for **Name**, **Address** and **DateOfBirth** date of birth are all functionally dependent on the specific **Person** in question.

In database theory, a formal technique for identifying and defining functional dependencies is known as normalization [2].

Normalization is a series of analytic steps that:

1. Ensures that all data elements in a group are discrete, i.e., can only take a single value. For example, no **Person** can have more than one **DateOfBirth**.

[1] We have found that the presentational complexity and diversity of document-centric artifacts makes them useful for teaching the skills needed to analyze and design more data-centric ones, where the presentational cues are less detectable and less intrinsic.

2. Establishes the primary identifier of each logical group. For **Person**, this would be the **Name** of the **Person**.[2]
3. Establish groups of data that are fully functionally dependent on each value of the primary identifier, i.e., for each instance of the group. For example, each time we introduce a new **Person** by adding a **Name**, we can also have a **DateOfBirth** and **Address**.
4. Ensures that all members apart from the primary identifier are functionally independent of one another. For example, the value of the **DateOfBirth** does not affect the **Address** and vice versa.

For database designers, normalization yields sets of relational tables. For Document Engineers, normalization yields the logical containers that put structure or "depth" into document schemas. The rationale is the same: "*recognizing functional dependency is an essential part of understanding the meaning or semantics of the data*" [2, pp.240-242].

Nornalization makes it easier to re-use existing patterns from other logical models. This is a formal way of stating what we do intuitively when we apply familiar patterns for structures such as **Address**. We take implicit patterns from postal labels, existing forms and maybe libraries of various business vocabularies.

3.5.2 Limits of Normalization

However, while the principles of normalization can be applied to the design of document schemas to achieve similar goals as in database design, these are not identical goals. Database models and document schemas are different in key ways [12]. Most apparent is that while most databases are built using relational structures, documents are generally hierarchical in structure. Therefore we must bear in mind that the actual implementation of normalized data structures will differ. In addition, XML document schemas may employ additional containers to preserve their historical structural integrity, that is applying the same assembly rules (e.g. page boundaries). This is often required when printed and electronic documents co-exist and you need to be able to produce paper on demand.

Many of these types of design decisions are pragmatic and based on the business rules of the required application. However, having the normalized model as a reference allows us to make these design decisions consciously and formally rather than on an ad-hoc basis. Not every database or document collection needs a data model that has been fully normalized – but it helps to know why it **isn't**.

3.6 Context

Analysis and design are two separate activities. We all suffer the temptation to build a "better way" into our models too early before we fully determine requirements. This often leads to inaccurate representations of existing systems and therefore poorer ultimate designs. For example, collapsing a three line address from a printed form into a one address field would not be an elegant design improvement because it would undoubtedly lose important sub-structure captured by the former.

But how much structure is enough? In the U.S. it may be sufficient to design **Address** to include **StreetAddress**, **City**, **ZipCode**, and **State**. However, we might also need separate structures for **RoomNumber**, **Floor Number**, **BuildingName,** **StreetNumber**, **StreetName** and less U.S.-centric structures like **PostalCode** and **StateOrProvince** if we have a wider range of addresses to encode and requirements to reuse them in other processes (such as sorting) that require this finer granularity. So we see that the requirements on the model are given to us by the context of the business process.

Recognition of context is an important factor to promote re-use of common patterns using customized refinements. Context can be applied by extending the component's name. For example, we may have a component known as **Contact** that describes a person or position that acts as a communication point in an enterprise. In the context of goods delivery, we may have a **ShippingContact** and in the context of payment we may have a **BillingContact**.

The context of a business process can be specified by a set of context categories and associated values [1] to promote consistency and completeness in the models of associated documents. For example, if a US glue manufacturer is selling to a French shoemaker, the context values might be as follows:

Contextual Category	Value
Process	Procurement
Product Classification	Glue
Region (buyer)	France
Region (seller)	U.S.
Industry (buyer)	Garment
Industry (seller)	Adhesives

Figure 3 Context Classifications Example (from [1])

3.7 Assembling Document Definitions

Having defined the components and structures we need, the next stage is to define the schemas for entire documents. Document definitions can be viewed as assemblies of structures and components based on a required business context [3].

Document assembly means creating hierarchical definitions (top-down and nested trees) as this is still the most practical way to define a document's structure.. Most documents have a strong structural hierarchy (book ? chapter ? section ? paragraph) or

[2] A person's name is not really a practical identifier since some names are duplicates (like John Smith), so we generally fabricate an identifier, such as Employee Number or SSID.

[3] "Document assembly" is sometimes used to describe the process of constructing a new document *instance* from fragments or components (see, e.g. [12]). Note that we are talking about assembling document *schemas* here.

clear divisions based on types of content (header – item details – summary) .

Assembly is carried out by creating pathways that establish the top-level structure and then navigating through the logical model based on rules provided by the context involved. For example, the context may determine that a **ShippingContact** structure is not required in a vendor managed inventory system.

For document types on the "presentational artifact" end of the Document Analysis Spectrum it is conventional to assemble the logical document in "document order" – that is, to organize the elements in the document schema so that their valid order matches the order in which they would appear in a document instance.

For example, the obvious way to assemble a document schema defining a book would be something like this, where the order of the elements follows the structure of the book in print:[4]

<!ELEMENT Book

(Forward?,Preface?,Introduction,
Chapter+,Appendix?,Bibliography?,Index?)>

But for document types toward the "data-centric" end of the document analysis spectrum there are likely to be a variety of different presentations for instances. These instances may differ in which information from the instance they present (they may be queries or views of the instance rather than a one-to-one rendering) and in the order or structure with which they present it.

For these data-centric schemas, it may be better to assemble the components of the document model in a way that facilitates authoring or the subsequent transformations rather than in an order that reflects the structure of a particular presentation of the information in the instance.

Put another way, this means that the same pool of components can be assembled with different container structures that significantly change the suitability of the document for various purposes.

This observation reminds us that rules for assembling information components into hierarchical document schemas are analogous to rules of normalization for database schemas. Normalization reduces redundancy, increases efficiency, and prevents insertion and deletion anomalies through which information is inadvertently lost or duplicated. Likewise, an appropriate container structure facilitates schema modifications (restrictions or extensions) or transformations.

3.8 Document Schema Encoding

Having established a new logical model for our documents, we have to recognize the constraints of the technology in which it will operate. We know WHAT we want, now we have to decide HOW it can be built. We are now moving back down Figure 2 into the process of implementation.

This will most likely entail encoding the models into a computer language and inheriting the constraints that this environment places on the model. Perhaps our technology platform cannot represent complex data structures or define certain business rules within its language. These factors need building into our new physical model. In an architect's plans, these would be the working drawings.

Increasingly, these models are being expressed as XML schemas. In these environments, factors such as the use of DTD or XSD may affect the new physical model.

Finally, our designs for new documents and processes are purely theoretical unless we take them and put them in the real world. If documents form the interfaces between applications, then it is these applications that provide the presentational features of the document. For example, XML programmers can build stylesheets to transform our **Address** description into three lines of text for the printed form and a different stylesheet for sorting by postal delivery. Hopefully, we have arrived at our required solution.

3.9 Practical Document Engineering

Of course, what we have presented is a panoramic view. Sometimes analysis is all we do, sometimes we repeat the cycle, or jump in at points along the way. Document Engineering is not intended to be a prescriptive methodology. Rather it presents a landscape in which to place our thinking.

4. PATTERNS

Models are valuable tools for identifying repeating or reoccurring features. We call these features, patterns. Patterns may structural, presentational or content patterns. They may be generic or context-specific.

It is by documenting these patterns that we can re-use them, either within the same model or in models for other systems.

Often these patterns are visible in the model but invisible in the concrete, real-world objects and functions that the model describes. For example, in recipes, noticing that the "beef stock" that is an incidental by-product of cooking beef may be an ingredient in other recipes or noticing that butter and margarine appear in identical or similar contexts and may be substitutes for each other.

Once patterns are identified they provide opportunities for simplifying structures and processes by replacing low-level specific descriptions with more abstract ones.

4.1 Patterns and Re-use

In addition to improving designs, patterns promote reuse. Reuse has the immediate benefit of reduced maintenance, encouraging and reinforcing consistency and standardization. Re-using 'logical' models also enables interoperability between different systems. For example, imagine if the addressing structures used by

[4] We use DTD syntax for simplicity here even though the assembly model we are defining is still a logical one whose syntactic realization is yet undefined (see Section 3.8).

Government agencies all followed the same model and that any changes to address could be propagated across agencies.

4.2 Libraries for Re-use

If patterns and other reusable artifacts of models are to be exploited they need to be easy to find. This most often implies a reuse repository or library of some kind [18]. For example, there are libraries of business processes such as RosettaNet [11] and libraries of document components such as xCBL [16] and UBL [17].

Document Engineered models can serve as the "front end" to these libraries because they provide the metadata or query structure to guide searches for appropriate patterns or models.

In reality there are often tradeoffs between using patterns and creating a model "from scratch." The effort spent studying and selecting a pattern should be exceeded by the benefits of using it, and because there may be "network effects" it can be tempting to "force fit" a problem into a pattern.

It is a designer's decision as to whether it is better to conform to a pattern or to customize a solution to achieve an exact fit.

This decision is complicated when we realize that "standards" for e-commerce such as UN/EDIFACT, RosettaNet, X12 and UBL are also libraries of patterns.

4.3 Patterns and Re-use Matrix

Different kinds of businesses vary in both their required level of abstraction and in the granularity with which they view business.

These two dimensions form a matrix we refer to as the Re-use Matrix (Figure 4).

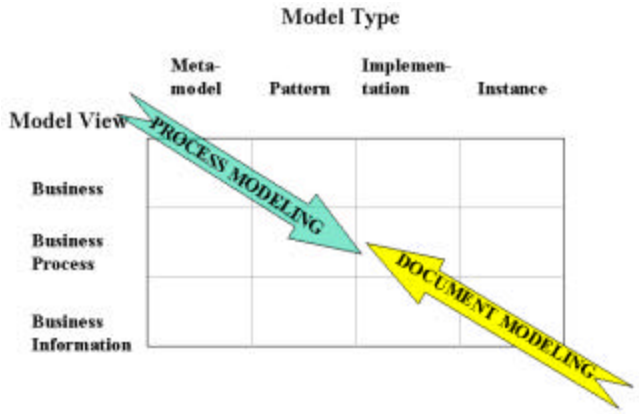

Figure 4. The Re-use Matrix

Figure 4 illustrates four levels of abstraction, varying from highly abstracted meta-models to specific instances of document themselves

Against the levels of abstraction we also have the depth of the business model itself. These range from a high-level business model, such as "vendor managed inventory" [14] or "outsource indirect procurement to a marketplace" to low level re-use of data components such as the common **Address** structure.

We observe that Document Modeling has traditionally started from the lower right (by analyzing instances of data components), while Business Process analysis has generally started from the upper left (abstract views of high level business models).

Document Engineering enables us to reach the middle of the Re-use Matrix. We want to express our business process models and business document models with enough details so that they are implementable and machine processable. By understanding the business process and business information models concurrently, we can achieve the best model for both. Effective process modeling requires an understanding of both the business information and the business process models. Business processes and business documents are complementary and should be treated with the same level of abstraction. We describe this balance as the *ying* and *yang* of e-business.

In practical terms this means developing schemas for business documents and schemas for business processes at the same time, with the same care, and to compatible levels of detail.

5. SUMMARY

We started out by describing some foundations for Document Engineering and how these melded into a view that document models can cover a spectrum of types, from human readable to electronic data. Documents can be classified according to their mix of presentational, structural and content components.

Document Engineering for e-Business focuses on the analysis and design of the content and structural components and how these are assembled into the models we know as document definitions. We saw how these document definitions must be assembled in the context of their particular business process requirements.

While they may approach their subject from differing angles, a recurring theme of Document Engineering is the value of identifying patterns in both data models and business processes. These patterns, which may be manifested as "standards," encourage interoperability through their re-use.

We started our paper by posing the questions:

- How do we identify, specify, and deploy the appropriate documents?
- How do we preserve our investments in older technologies for document exchange while taking advantage of new ones?
- How do we preserve our investments in business processes and relationships while creating new ones?

Document Engineering for e-Business is proving to be a practical methodology for solving these problems. We have been encouraged by its application in both the practical development of a new e-business vocabulary and the training of a new generation of document engineers.

6. ACKNOWLEDGMENTS

This framework for Document Engineering for E-business was developed for a course with the same title taught by the first author in Spring, 2002, at the University of California, Berkeley. We thank the brave students for their courage in taking a "first time ever" course and suggesting numerous mid-course corrections. We also thank Brian Hayes and members of the UBL initiative for engaging in many useful discussions and debates with us.

7. REFERENCES

[1] Context and reusability of core components. 10 May 2001. http://www.ebxml.org/specs/ebCNTXT.pdf

[2] Date, C.J. *An Introduction to Database Systems 3rd Edition*, Addison-Wesley, 1981.

[3] Downes, L. The information supply chain. Chapter 4 of *The Strategy Machine*, HarperBusiness, 2002.

[4] Eph'al, I. & Naveh, J. *Aramaic Ostraca*. Magnes, 1996.

[5] Glushko, R. The plug-and-play economy. *Purchasing*, (December 22, 2000, 72-74).

[6] Glushko, R., Tenenbaum, J. & Meltzer, B. An XML framework for agent-based e-commerce. *Communications of the ACM*, (March 1999, 42(3), 106-114).

[7] Glushko, R., Weaver, M., Coonan, T., & Lincoln, J. "Hypertext Engineering": Practical Methods for Creating a Compact Disc Encyclopedia. *ACM Conference on Document Processing Systems*, 1988, 11-19.

[8] Maler, E, & Andaloussi, J. *Developing SGML DTDs: From Text to Model to Markup*. Prentice Hall, 1996.

[9] Mulberry Technologies, Inc. *Data Modeling and XML Vocabulary Development.* http://www.mulberrytech.com/papers/xmlvocab.pdf

[10] Raymond, D., & Tompa, F. Hypertext and the Oxford English Dictionary. *Communications of the ACM*, (July 1988, 31(7), 871-879).

[11] RosettaNet PIP Directory http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity%5BOID%5B9A6EEA233C5CD411843C00C04F689339%5D%5D

[12] Salminen, A., & Tompa, F. Requirements for XML Document Database Systems. *ACM Symposium on Document Engineering*, 2001, 85-94.

[13] Simchi-Levi, D., Kaminsky, P., & Simchi-Levi, E. *Designing and Managing the Supply Chain*. McGraw-Hill, 2000.

[14] Supply-Chain Operations Reference-model: Overview of SCOR Version 5.0. Supply-Chain Council. 2001. http://www.supply-chain.org/slides/SCOR5.0OverviewBooklet.pdf

[15] Travis, B., & Waldt, D. *The SGML Implementation Guide*. Springer, 1995.

[16] XML Common Business Library. http://www.xcbl.org/about.html.

[17] UBL: The Next Step for Global E-Commerce. http://www.oasis-open.org/committees/ubl/msc/200112/ubl.pdf

[18] XML.ORG Registry. http://www.xml.org/xml/registry.jsp