Abstract

In the race to transform the World Wide Web from a medium for information presentation to a medium for information exchange, the development of practices for ensuring the security, auditability, and non-repudiation of transactions that are well established in the paper-based world has not kept pace in the digital world. Existing Internet technology provides no easy way to create a valid 'digital receipt' that meets the requirements of both complex distributed networks and the business community. In addition, an improved articulation of digital signatures is needed. Extensible Forms Description Language (XFDL), developed by UWI.Com® and Tim Bray, is an application of XML that allows organizations to move their paper-based forms systems to the Internet while maintaining the necessary attributes of paper-based transaction records. XFDL was designed for implementation in business-to-business electronic commerce and intra-organizational information transactions. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: XML; XFDL; Electronic commerce; Electronic records; Records management

1. Introduction

This paper discusses the issues surrounding the creation of legally-binding electronic transaction records on the Internet and outlines an XML-based solution called Extensible Forms Description Language (XFDL). This discussion begins with an analysis of the current function of paper forms within organizations, and then moves to an analysis of the business need for viable transaction records in electronic commerce. From there, digital signatures and existing Web-based technology are studied, and finally, XFDL is explained in detail.

2. Understanding the need for forms

Forms can be described as the primary user interface for business [13]. Industry analysts have estimated that organizations worldwide produce 90 billion pre-printed pages per day — a significant portion of which are paper forms [4]. In order to understand why forms are so pervasive, it is necessary to analyze the functions that paper forms currently serve within organizations.

2.1. Paper forms are data structures

Paper forms allow specific sets of data to be easily organized, classified, and understood. Without a structured way to represent data, high-volume business processes would be very difficult to implement and organize. For example, purchasing departments typically depend on the structured data
of forms to gather and process critical information.

2.2. Paper forms are user interfaces

Paper forms are a simple and elegant method of providing structured data to information systems. As with any user interface, there are design elements that provide inherent logic to meet the demands of the human user on the front end and the process on the back end. Paper forms are a unique user interface, in that they provide both the templates for data collection, and the record or archive of the data collected.

2.3. Paper forms are transaction records

Perhaps the most important function of paper forms is their ability to create transaction records. Most paper forms are contractual in nature, either explicitly or implicitly. In practice, users of a form indicate their participation in this contract by filling in the necessary information and adding their signature. Should there be any disagreement about the contract represented by the form, the form can be used as proof of the transaction. The term ‘transaction’ is used generically and can refer to the exchange of goods, services, money, information, and so on.

Paper forms capture a fixed record of the state of the data at the time the form was signed — in other words, they create a valid ‘source document’. Indelible ink prevents tampering, and the signature provides signer authentication and authorization.

While this is an important feature of paper forms, it is also creates a dependence on physical documents that severely hampers the automation of forms processing systems. Many organizations have strict rules governing the maintenance of transaction records that may also restrict automation. Until recently, these issues have been largely ignored in the creation of Web-based forms.

Clearly, paper forms are a vital part of the business process. However, it is also clear that dependence on paper-based systems is inefficient and costly in a digital world. Analysts estimate that it costs organizations in the USA over US$ 120B annually to process paper forms [11].

3. Understanding the business need

The emergence of electronic commerce as a driving force for technological development is widely recognized. Various mechanisms are now in place to enable effective transactions between businesses and consumers. However, another area of growth that may ultimately have a greater impact on the development of Web-based technology for exchanging structured information is that of business-to-business electronic commerce. Business-to-business electronic commerce may also be a key driving force for the acceptance of XML, as it cannot be accomplished within the limitations of HTML alone [6].

It is estimated that the value of business-to-business electronic commerce will grow to US$ 1.3 trillion by 2003 [16]. Unlike consumer-based electronic commerce, business-to-business electronic commerce requires both buying and selling organizations to seek methods for integrating their business processes.

XML [10] has been heralded as a key enabling technology for this type of integration [20]. Since transactions of this type typically involve the communication of some form of structured data, having a standard syntax for creating and exchanging data structures is obviously an important step. However, XML is not a solution in itself, but simply a framework for describing the necessary protocols.

XML offers a method for integrating business processes by providing an open, extensible structure for data exchange. Reliance on unique or proprietary technology for these tasks is costly, inefficient, and risky, often resulting in ‘black box legacy systems’ that may be part of a strategy to lock organizations into a particular vendor’s products [19]. EDI was an attempt to enforce standard data formats for electronic commerce, but the cost of implementation has limited its application. XML can provide many of the benefits of EDI without the prohibitive infrastructure costs.

3.1. What makes a valid transaction record?

When the importance and pervasiveness of paper forms in business is considered in conjunction with the growth of electronic commerce, it is clear
that a method for moving paper forms to the Internet is needed. More specifically, the features that paper forms offer (data structures, user interfaces, and transaction records), must be integrated into electronic commerce systems. XFDL was designed specifically for this purpose. In order to understand the design of XFDL, it is necessary to look at the architecture of transaction records in more detail.

The legal requirements for paper records have been clearly established over many years. In addition, the regulations governing the general admissibility of electronic records have been dealt with in many jurisdictions [15]. There are three key concepts to be considered in the management of transaction records: security, non-repudiation, and auditability. These concepts are based on a foundation of authorization, signer authentication, document authentication, and context preservation. Fig. 1 illustrates the relationships between these concepts in the ‘hierarchy of transaction record validity’.

3.1.1. Security

Guidelines for transaction record security can be found in the Digital Signature Guidelines of the American Bar Association [3]. Security is achieved through authorization, document authentication, and signer authentication.

The ceremony of signing a record denotes the signer’s authorization, which means that the terms of the transaction are understood and the signing parties agree to proceed with the transaction. Document authentication refers to the fact that records must provide protection from tampering after they have been completed. Paper forms provide this, at least to a degree that is acceptable to the courts, due to the difficulty of modifying the form’s appearance without destroying the paper. Signer authentication refers to the ability to identify who signed a document, message, or record. The inherent uniqueness of handwritten signatures on paper forms authenticates each signer by providing proof of identity.

The prevailing electronic method for achieving transaction record security is the digital signature. In an environment where digital signatures are used, each person is given a mathematically related pair of ‘keys’. The ‘private key’ uniquely identifies the signer; consequently, access must be restricted to the key’s owner. The ‘public key’ is used by all persons to authenticate signatures created with the private key. The typical formulation for the digital signature algorithm [14] states that when a signer ‘performs the ceremony’ of creating a signature (implying authorization), a signature $S$ for a given message $M$ is created by the function $Encrypt(hash(M), PrivateKey)$, and that verification of signature $S$ is accomplished by testing the equality of $hash(M)$ and $Decrypt(S, PublicKey)$. The $hash()$ function is a mathematically sound method of detecting a change in $M$, thereby achieving document authentication. Encryption protects the hash value from tampering, and decryption will only be successful with signer’s public key. Because the public key is uniquely related to the private key used to create the signature, signer authentication is achieved.

3.1.2. Non-repudiation

A record that offers transaction non-repudiation ensures that the agreement represented by the record cannot be disputed. Paper forms provide all the content necessary to prove the nature of an agreement. A paper form provides a logical structure that is fixed and immutable. A form’s fields, check boxes, and other elements have corresponding labels (associated by position) that provide the ‘questions’ of the contract, and the data entered with permanent ink provide the ‘answers’ to those questions. Paper forms accomplish non-repudiation in such a transparent manner that it takes some effort to realize that special precautions must be taken to achieve the same results by digital means.

In the electronic world, it is usually assumed that transaction non-repudiation is accomplished through the use of digital signatures. Actually, it was quickly recognized that the digital signature algorithm did not completely solve the problem. The efficacy of a digital signature is directly proportional to the
security with which the signer’s public key can be delivered to the verification phase. An entire industry devoted to the creation and maintenance of Public Key Infrastructures (PKI) has been developed to address this issue.

There is a similar problem on the opposite side of the digital signature process — during signature creation. Consider the reason why a signer creates a digital signature. A digital signature is only needed if message $M$ will be given to another party — the verifier. Since $M$ has a source and a destination, it is essentially a transaction record. A digital signature can achieve transaction record non-repudiation, but the above stated goal of transaction non-repudiation is to “ensure that the agreement represented by the record cannot be disputed”. Hence, the efficacy of a digital signature in achieving transaction non-repudiation is directly proportional to the extent that the transaction record $M$ completely represents the transaction.

The Association for Information and Image Management’s (AIIM)\textit{ Performance Guideline for the Legal Acceptance of Records Produced by Information Technology Systems} lays out the following guidelines:

The following shall appear with sufficient clarity so that each can be recognized:

- Individual letters, numbers and symbols;
- Combinations of letters, number, and symbols forming words or sentences;
- Graphics, such as signatures, logos, pictures, etc;
- Sound;
- Other features of records such as color, shape, texture, etc. that relate to the content of the information [2].

These guidelines further illustrate the need for transaction records to contain not only the content of an agreement, but the context and structure as well.

3.1.3. Auditability

A record that provides non-repudiation not only protects the parties in the agreement, but also can consequently be part of a reliable audit trail for internal (business audits) or external (legal, or chain-of-custody) purposes. In order for a transaction record to be part of a viable audit trail, it must contain at least the following information:

- who was involved;
- when the transaction occurred;
- the nature of the transaction; and
- the results of the transaction [13].

The paper forms involved in each step of a supply chain, for instance, are all part of the audit trail for a given transaction. As the ‘hierarchy of transaction record validity’ in Fig. 1 illustrates, auditability is predicated on effective transaction non-repudiation, so automated processes will only be more effective than paper when they meet or exceed the ability of paper to achieve non-repudiation.

3.2. Existing internet technology

Although there are many approaches to Web-based forms and transactions, it is useful to look at HTML forms as a baseline in order to illustrate key points about transaction records on the Web. HTML forms are often used as a replacement for paper forms in business-to-consumer electronic commerce. Although HTML forms do provide a user interface, they do not duplicate the other two important features of paper forms — namely, the provision of a data structure, and the creation of a viable electronic record.

HTML is a language that was designed to separate markup from content, a concept inherited from SGML. While this philosophy has been a cornerstone in the development of the Web, it also makes HTML unsuitable as a replacement for paper forms in high-value business-to-consumer and business-to-business electronic commerce.

Most existing Internet protocols cannot address this issue because of the philosophy behind their basic architecture. Traditionally, engineers have divided an application’s data, presentation, and logic into separate, clearly-defined architectural layers. This approach to systems architecture has a great deal of merit — stratification provides a tremendous degree of flexibility. In a stratified information system, interfaces can be changed and extended without modifying the underlying data, new data sets can be used while maintaining a consistent look-and-feel for the users, and modifications can be made to the application’s logic without affecting the existing interfaces, or the underlying data. Unfortunately, this elegant flexibility is exactly what cannot exist in
transaction records. The ability to easily change the context (interface) while keeping the content (entered data) static diminishes a transaction record's legal effectiveness.

In practice, when a user fills and submits an HTML form, only the input data (content) is sent to the server, where it usually becomes fields in a database. The HTML form (context and structure) itself is maintained as a separate HTML file. Because the act of submitting an HTML form separates the content from the context and structure, a transaction record that offers non-repudiation and auditability is not created.

Consider this example: A person buys a US$250K life insurance policy by filling out an HTML form with the desired policy parameters and digitally signing the submission. Five months later the person is diagnosed with a terminal illness, then dies after another five months. When the bereaved spouse requests the insurance payment, the claim is denied due to a caveat that the policy does not cover death due to terminal illness diagnosed within six months of the start of the policy — a caveat that appeared in the fine print. Litigation is the most likely result.

The spouse can easily repudiate the transaction because the insurance company only has a digital signature on the tag=value pairs that represent the policy parameters collected by the HTML form. There is no way to prove that the fine print was even there. Further, even if the HTML form mentions a six month caveat, the wording is very important: was it diagnosis within six months, or death within six months? Now suppose that the static text of an HTML document was included in the signature. The spouse can still repudiate the transaction by claiming that the fine print had the same color as the background, that it was positioned in negative pixel space, that the font was too small, and so on. Indeed, the proper formulation of the transaction record must include the questions, the user's answers, and a representation of how this was presented on the computer screen [2]. Unlike a paper form, which captures a ‘snapshot’ of the agreement at the time of signing, HTML forms offer no similar guarantee.

Java systems usually have the same problem — they store and transmit the data files separately from their presentation (the Java classes). In addition, EDI systems were designed primarily to reduce input errors caused by manual key entry and to bypass slow physical delivery systems, and as such, very few, if any, EDI systems have a built-in facility for capturing the content and context of electronic transactions [13].

3.3. Liability and technical shortcomings of existing implementations

Although electronic commerce is growing very quickly, most organizations do not have the ability to create viable electronic transaction records. In the absence of these records, the parties involved in a transaction simply have to accept the risk of liability. Most business-to-consumer electronic commerce transactions are currently being conducted on a ‘faith’ basis. If the transaction falls into dispute, it is understood that one party (usually the larger one) will simply accept the loss. Larger partners sometimes enter into ‘blanket’ agreements with financial institutions that define specific resolution mechanisms should disputes arise.

Other organizations approach this problem using paper/digital hybrids. These companies use an electronic forms package to design, distribute, and fill their forms. However, to create viable transaction records, these forms must be printed and physically signed. The paper record is then filed manually or scanned into a digital imaging system for archiving. The disadvantages of this ‘fill and print’ approach are clear. A great deal of money could be saved if the form remained in a digital format throughout the entire process.

HTML and Java-based solutions may be appropriate for low-value business-to-consumer electronic commerce, where blanket dispute resolution agreements exist. In these situations, the requirement to retain detailed information may not be critical for regulatory compliance, or for discovery and litigation [13]. However, these solutions are not adequate for high-value business-to-business electronic commerce and critical intra-organizational information transactions. A better system — one that more closely approximates existing paper business practices — is required. XFDL was designed for this type of implementation, where improperly-kept transaction records have the potential to inflict serious liability.
4. Extensible Forms Description Language (XFDL)

XFDL is a highly-structured XML protocol designed specifically to solve the body of problems associated with digitally representing paper forms on the Internet. The features of the language include support for a high-precision interface, fine-grained computations and integrated input validation, multiple overlapping digital signatures, and legally-binding transaction records [8].

4.1. XFDL is a document-centric markup language

XFDL takes a ‘document-centric’ approach to digital forms by storing each necessary element of a viable transaction record in one securable file. Document-centrism is a controversial design tenet for an XML derivative, but it is the key feature in creating non-repudiable transaction records — the ‘message’ must accurately record the full transaction context in order to achieve the essential purpose of digital signatures. Just like a paper form, the content, context, and structure of an XFDL form are all stored together. In the event of a dispute, an XFDL document can be recalled and used to prove the exact nature of a transaction. An organization can therefore create a ‘forms repository’ of XFDL documents that provide legally-admissible transaction records for dispute reconciliation, business audits, chain-of-custody processes, and so on.

4.2. XFDL is a computational language

Unlike most XML languages, XFDL is also a programming language. XFDL is smart enough to make decisions, handle arithmetic, and respond to user input. The fine-grained computational power of XFDL allows it to automatically perform the math and logic functions that are performed manually in paper forms. This allows an XFDL document to direct a user through the interface, performing calculations and error corrections ‘on-the-fly’.

This computational power, or logic, is embedded into the XFDL document and becomes part of the transaction record that is digitally signed. Embedded logic also gives XFDL documents offline functionality, as an external connection to a script or some other source of programmatic intelligence is not required. This also makes XFDL documents useful for nomadic, or disconnected, data collection using portable computers and hand-held appliances.

Infix notation was initially chosen to represent computations within XFDL. Allowance was made in the XFDL specification for supporting the prefix semantics of MathML [18] in future versions. The primary reason for this decision was readability. Understanding simple calculations in an infix notation is reasonably intuitive, whereas doing so in a prefix notation is not. This problem is aggravated when complex decision logic is represented, where a one-line infix statement can require ten to twenty lines of prefix code to accomplish the same task.

4.3. XFDL is assertion-based

XFDL’s computational logic is expressed using assertions. Unlike most traditional programming languages that function on a procedural mode, there is no thread of execution to be managed in XFDL. Creating computations in XFDL is much like using a spreadsheet. For example, to make Field1 the sum of Field2 and Field3, Field1 is ‘told’ that this is the case. As the user interacts with the form, changing the values of Field2 and Field3, the XFDL language makes sure that the assertion is always true.

XFDL was engineered as an assertion-based language for two reasons. The first is readability. The type of logic that is normally used in complex forms is very easy to describe using assertions. Capturing events and running procedural code is overly complicated for operations like “make Field1 the sum of Field2 and Field3”.

The second, more compelling reason for an assertion-based design is that it is very easy to freeze the exact state of a document when the user decides to save it, sign it, or send it to a server. Capturing the exact state of a program written in a procedural language would require the acquisition of a great deal of information, such as the heap, the call stack, the data and code segments, and so on. Even if this information could be captured easily, it would be difficult to interpret — the transaction would not be human readable. By operating an XFDL document as a ‘state machine’, it is easy to take a snapshot of the document at any moment of its execution.
4.4. XFDL is human-readable plain text

A core design tenet of both XML and XFDL is human readability. File formats for transaction records obviously need to be machine-readable, but can be more ‘future-proof’ if they are also human-readable [19]. XFDL is designed to create transaction records whose life span may be much longer than any particular Internet technology or vendor. Organizations need to maintain these records as evidence of their business process for many years, and, as such, cannot depend on opaque binary data formats for this purpose.

4.5. XFDL is a publicly accessible open standard based on XML

XFDL derives several benefits from having a human-readable XML syntax. Any document format, such as XFDL, which claims to be human readable must be a publicly accessible open standard. This is necessary to ensure that the semantics of the XML elements are known. As a natural consequence of this, organizations can share structured information gathered by XFDL documents with other organizations without a costly translation process. This is also valuable for organizations that want to share data among internal departments whose systems may be based on fundamentally different ontology. Furthermore, because it is built on XML, XFDL is easy to generate and manipulate using any of the XML-enabled tools on the Web, and it is easier to find human resources who have the skill set necessary to create software to perform these manipulations. While openness is important for any Internet protocol, the concepts of ‘data mining’ and ‘future-proofing’ information are especially prescient for a language that creates transaction records.

4.6. XFDL is extensible, allowing application-specific/server-side logic

Most languages based on XML are not themselves extensible, but rather are composed of a very definite set of keywords, rules, and so on. Although the most important function on a non-repudiable transaction — the digital signature — occurs on the client machine, the full lifetime of a transaction can be quite complex and involve processing by modules other than a forms viewer. In a ‘fat-client’ application design, these modules will run on the client machine, whereas the ‘thin-client’ application architecture will have these modules on the server side. In either case, the ability to add custom functions to XFDL computes and, in general, to use XFDL’s computation engine within custom items and options designed for modules other than a viewer is a key component in managing a transaction record’s lifetime from a single source document.

For example, custom items, options and functions can describe complex interactions between a form and a server-side ODBC database. They can also be used to specify workflow logic for the form based on content in the form. Indeed, most of the essential functions of electronic commerce transactions occur on a server after the client has provided authorization, and XFDL’s computation engine facilitates these operations by letting the transaction document have greater control of its own processing.

5. Advanced discussion of XFDL

As stated previously, the primary design goal for XFDL was the creation of secure, auditable, and non-repudiable transaction records for Web-based information and electronic commerce transactions. However, the creation of viable transaction records is only one piece of the puzzle for organizations that wish to move their paper forms to the Internet. A complete range of features designed to reflect the ‘real world’ business need for electronic commerce forms is required. These features are outlined in the stated design goals for XFDL [8]. This section discusses XFDL in more detail, and also discusses a number of these other design goals.

5.1. Structural overview

The root element of an XFDL form is surrounded by <XFDL> and </XFDL> tags. Each page of the form is surrounded by <page> and </page>. The items in a page represent widgets such as buttons, fields, labels, checkboxes and pop-ups, but also invisible objects like digital signatures and custom items.
A button, for example, would be surrounded by `<button>` and `</button>`. A button on the form will have several options, represented by subordinate elements. For example, if the button displays the text ‘Submit’, then the option `<value>Submit</value>` would appear in the button.

Fig. 2 shows the complete text for a small form. The benefits of XFDL over other solutions involving technologies like Java are best seen in larger forms found in business and government, but Fig. 2 has most of the structural components that are simply iterated to build those larger forms. The example form does the familiar Pythagorean Theorem calculation, which determines the hypotenuse length of a triangle given the lengths of its sides.

5.1.1. Root element, version, page, item, and relative scoping

The XFDL element has a mandatory attribute called version that indicates the language version of XFDL to which the form complies. This controls which keywords may be available, for example. Each page and item has a mandatory attribute called `sid`, which stands for scope identifier. In order to support the computation system, each XML element must be uniquely identifiable. XML contains a method for doing this, but it requires an identifier that is unique within the document, so it is not scalable to the large forms typically found in business and government. This is important when trying to cut-and-paste logic while building forms, but it is especially pertinent when creating XFDL forms that dynamically duplicate a new row of items while running in the viewer. For example, a purchase order that grows dynamically must duplicate all line items. In this case, all computations must have relative scope so that the new computes will apply to the new items, rather than to the originating items.

5.1.2. Options in the form global, page global, and item content

Before the first page element, and before the first item of each page, an XFDL form can contain ‘global’ options. These are option elements that generally provide default settings for the options in each item. Their structure is the same as options appearing in an item, so only item options will be discussed here. Even an item as simple as a label can have many options, such as its text value and font specification as shown in the Title label of Fig. 2. Additional options such as `image` and `itemlocation` describe precisely where the label should be on the form. A number of other item-specific options control a variety of behaviors. For example, a field can be made read-only (`editstate`) or contain input validation and formatting (`format`).

The `sid` attribute is not required and has no meaning in options (or any elements they contain). Options do not need a `sid` attribute because the element tag acts as the scope identifier. According to Tim Bray [9], this dichotomy causes XFDL to have a ‘look-and-feel’ that XML users have come to expect.

5.1.3. Element depth for arrays and computes

Some options, like `value`, contain simple character data, while others, like `fontinfo`, contain sub-elements. This latter type of option exemplifies XFDL’s notion of an array. However, elements for simple character data may need to contain a computation that calculates the character data, such as field C in Fig. 2. The `content` attribute is used to indicate how an element’s content will use element depth.

A unique scope identifier is optional for array elements. The tag `ae` is a reserved tag used by array elements that have no scope identifier. Such elements can still be referenced in computes using ordinal position. For example, the result of `pl.Title.fontinfo[facename]` evaluates to “Times”, but `pl.Title.fontinfo[0]` is also “Times” because XFDL arrays are zero-based and can be accessed by ordinal position even if they are named.

5.1.4. Compute structure and syntax

A computed element contains the sub-elements `cval` and `compute`. The content of `cval` is the current value of the computation given by the compute element. The `cval` plays an important role in transaction non-repudiation because its contents are frozen once a digital signature is applied to the surrounding element. This helps XFDL to create a ‘snapshot’ of the transaction.

XFDL includes the standard arithmetic operators as well as the ternary decision operator (`?:`). The six comparators, logical-and, logical-or and logical-not are available for creating conditions, and nested
<XFDL version="4.0.0">
<page sid="p1">
  <label sid="Title">
    <value>Pythagorean Theorem</value>
    <fontinfo content="array">
      <facename>Times</facename>
      <ae>18</ae>
      <ae>bold</ae>
    </fontinfo>
  </label>
  <field sid="A">
    <label>Enter side length A:</label>
    <value>3</value>
  </field>
  <field sid="B">
    <label>Enter side length B:</label>
    <value>4</value>
  </field>
  <field sid="C">
    <label>Hypotenuse length C is:</label>
    <editstate>readonly</editstate>
    <value content="compute">
      <![CDATA[
        A.value <= "0" || B.value <= "0"
        ? "" : sqrt(A.value*A.value + B.value*B.value)
      ]]> 
    </compute>
  </field>
</page>
</XFDL>

Fig. 2. An XFDL form.

decision logic is permitted. XFDL also provides numeric, financial, and string manipulation functions, as well as the ability to define new functions.

Operands are specified by a relative scoping notation that is rooted at the center by the option identifier. For example, fontinfo is the center of p1.Title.fontinfo[0]. The static reference operator (.) is used before this option to ascend the element hierarchy, and array brackets are used after the option to descend the element depth of arrays. The reference returns the specified element’s simple character data content, or the content of the cval sub-element if the specified element is computed. Hence, computed and non-computed options can be seamlessly integrated.

A reference need not list the entire element chain above the option. Using relative scoping, the unspecified portion of a reference is assumed to be equal to the ancestor elements containing the reference. For example, in Fig. 2 the reference A.value in the compute of field C’s value is assumed to refer to an item A appearing on the same page as item C.

5.1.5. Use of cdata with computes

Some computation operators, such as logical-and (&&) and less-than (<), are forbidden by XML as character data because more sophisticated parsing techniques would be required to decide whether & is part of an entity reference or when < is part of a start or end tag. Hence, computes that use XFDL decision logic are often enclosed in an XML CDATA section.

5.1.6. Compute parsing requirements

The XFDL syntax rules allow a compute to contain a mathematical expression or a decision statement, yet a conditional expression can also begin with an arbitrarily long mathematical expression. Hence, a recursive descent parser will not suffice; an SLR(1) or better parser is required [1]. However, modules that simply need to process the XFDL form
as an XML document do not need an auxiliary parser because the compute can be treated as character data.

5.2. XFDL and DTDs

The DTD notation offered by XML is useful for some XML extension languages, but creating a DTD for a language with a mandate of XFDL’s size is not possible. DTDs were problematic from the beginning of XFDL’s design because a DTD has the same expressive power as a regular expression, so XFDL computations (whose recognition requires a parser rather than a lexical analyzer) could only be validated as PCDATA. However, the extensibility of XFDL (the ability to create custom items and options) is the largest problem. A third obstacle was the inability to enforce tag uniqueness but not element order at the option level.

5.3. Reduced server-side processing via input validation/formatting and calculations

One problem with paper-based systems is the high probability for introducing incorrect data into the system. For instance, a person may write an improper date. Then the error is duplicated by data entry clerks, and eventually integrated across the enterprise into departments such as purchasing and inventory. These front-end errors are costly. In large organizations, forms may go through dozens of steps before reaching their final destination. Input errors are expensive in this situation because they delay the stream of processing or cause the process to be rolled back and re-initiated. XFDL minimizes these types of errors. For instance, a form field can be set up to accept only dates or only integers in a certain range. XFDL defines a wide variety of data types, including dates, integers, floats, strings, and so on. XFDL also allows for a variety of checks to be performed, including range, length, and template value comparisons. Furthermore, XFDL computations also reduce server-side processing by preventing math errors on calculated fields.

5.4. Binary enclosures with Base-64 encoding

XFDL offers an internal flat-file system for enclosures. Enclosures can be of any type, and are grouped into ‘folders’ for ease of use. Because enclosures are signed, saved, and transmitted as part of the XFDL document, they too become part of the transaction record. This is useful for applications such as legal forms that require free-form statements as supporting documentation, or online recruitment forms that require résumés to be attached. Note that these enclosures often contain binary data, so they are converted to Base-64 [17]. This is not human readable, but enclosures often represent digital images or sounds which cannot be stored in a human-readable format. Attachments of other file formats are also possible, with human readability being sacrificed for a smoother transition from opaque formats.

5.5. Precision layout for dense business and government forms

Many organizations, especially those in government and in regulated industries like insurance and health care, have devised exhaustive rules for governing the appearance of paper forms. It is important that these organizations have a method for precisely reproducing the appearance of paper forms on the Web, when even small changes in font color, size, and the layout of boxes and lines can seriously impact the legal viability of the form record [2]. Unlike HTML and Java-based forms, XFDL documents allow precise recreation of the layout and appearance of paper forms on the Web. Although the acceptance of Web technologies such as CSS [5] and XSL [12] promise to offer greater interface control, these technologies currently only address the need for precise presentation, not the need for viable transaction records. For instance, a style sheet cannot currently be inextricably bound to user-entered data.

Some organizations also require the ability to precisely print forms, even after instituting a Web-based forms processing system. XFDL documents support precision printing for these situations. This facilitates smoother transition from paper-based and ‘fill-and-print’ processes.

5.6. Digital signature security and flexibility

XFDL defines a technology-neutral digital signature interface that allows it to work with most common digital signature systems, such as RSA, DSS,
biometric tokens, and so on. For cryptographic signatures, the signer’s public key certificate is included with the encrypted hash. In addition to verifying the encrypted hash, the certificate authority signature on the signer’s public key certificate is also verified using a certificate authority certificate on the verifying box. This simplifies the delivery of the signer’s public key while preventing substitution attacks [7].

The digital signature hash value is not just computed on the entered data, as is the case with HTML forms. The XFDL elements that contain the content, context, and structure of the original form are ‘locked down’ by the digital signature, in much the same way that a paper form is considered to be immutable once signed. The XFDL computes on signed values are also discontinued, and the cval element content represents the computed value at the time of signing.

XFDL also supports multiple overlapping signatures that apply to different sections of a form. Many complex paper forms require multiple levels of input and approval. In this case, a multi-section form is used to gather the appropriate information and signatures. XFDL duplicates this facility with signature filters that specify either by inclusion or exclusion the pages, items, and options to be signed by a particular signature. Each signature item can have its own signature filter options, and the elements specified as being signed can overlap elements signed by other signatures and can also include elements not signed by other signatures. In practice, an XFDL form can be routed through an electronic workflow system, with each user adding the required information and digital signature until the document is complete.

6. Conclusion: forms, XFDL, and electronic commerce

“Soon, to the degree that the Web continues to evolve toward richer data representation and proprietary systems gain Web interfaces, XML will mediate the recreation of reality in cyberspace.” [19]

This paper has examined the role that paper forms play in the business process, as well as the role to which they will evolve in business-to-business electronic commerce. For centuries, organizations have used documents to make legal representations. For decades, most of these documents have been forms. Effective electronic commerce technology must accommodate the business process. XFDL is a language that endeavors to ease the transition from paper-based and fill-and-print systems to the Web by mirroring real-world requirements, rather than forcing organizations to adapt their business processes to new technology.

Acknowledgements

Special thanks to David Manning, CTO of UWI.Com, for direction and information. Thanks also to Steve Shewchuk, Lori Waters, and Paul Chan.

References

[9] T. Bray, Personal communication to J. Boyer during the design phase of XFDL.
[10] T. Bray, J. Paoli and C.M. Sperber-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C recommenda-


Barclay Blair works in Marketing Communications at UWI.Com. He received a Professional Writing degree from the University of Victoria, and has been writing and speaking about the Internet since the inception of the World Wide Web.

John Boyer is a software development manager at UWI.Com and a co-editor of the XFDL specification. He is also a doctoral student in the Computer Science Department at the University of Victoria.