Integrating XML with legacy relational data for publishing on handheld devices

David **Lee**

Abstract

Epocrates is the industry leader in providing clinical references on handheld devices. Much of the clinical content resides in "legacy" relational databases which are the backbone of the infrastructure for managing the lifecycle of content publishing. A key constraint of publishing content to handheld devices is that the IO channel can be very slow and the content can only be pushed during synchronization; a period where the user is often in a hurry.

Recognizing modified data since the last sync requires support at the database level for aggregating data at the device 'record' level managing deltas and change dates for the aggregated data.

Because XML data may reference relational data, it is not sufficient to manage changes to the XML data in order to determine if the final content is changed, therefore managing changes to the final (binary) content is required in addition to changes to the XML.

This presentation is a case study and high level discussion of the architecture and work flow used by Epocrates for publishing heterogeneous content to handheld devices.

Table of Contents

1. Introduction

1.1. Common Terminology

There are a few terms which are difficult to avoid, so they are defined here.

- PDA - "Personal Digital Assistant". In this paper refers to handheld devices, running a general purpose operating system, specifically the Palm/OS or Pocket PC OS (Win/CE).
- Monograph - From clinical terminology, refers to the information describing a single drug or preparation. In this paper refers more generally to the information describing a single drug, disease, lab test, preparation or other clinical entity. Roughly analogous to a section, or single reference work about a specific topic. A single data source usually provides a collection of Monographs of a particular type.
- PDB - From the Palm/OS terminology, a "PDB" is a "Palm Database". A very simple variable length record format with a single 16 bit key index.
- Syncing - The process of synchronizing a server's database with a PDA's database. New records on the server are inserted into the PDA database, modified records updated and records which no longer exist on the server are deleted from the PDA.
- Sync Server - The online servers in the Epocrates data center responsible for managing syncing to the PDA's.
- BLOB- "Binary Large Object", a database terminology meaning a field that can store large binary data in a single field.
- ICD9Codes – "International Classification of Diseases, Ninth Revision". An international standard coding system of diseases.

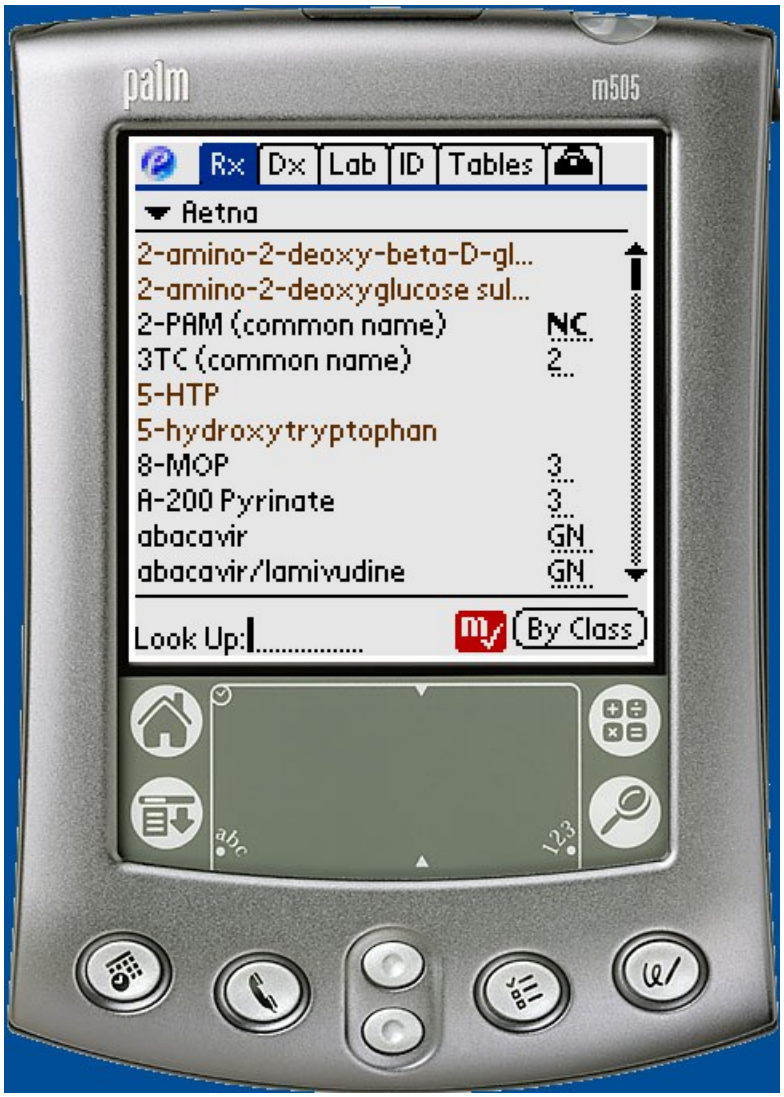
2. Background "The Problem Space"

Epocrates' core product is a medical reference tool which runs on PDA's. Data is entered by a team of clinical experts using various in-house tools which store the data into an Oracle SQL Database. This data is highly structured, cross referenced, and updated continuously. This data is published to over 475,000 active subscribers by means of a "syncing" protocol. When synchronizing while connected to the internet, Epocates' servers are contacted and new and updated data is pushed to the PDA.

The "Problem" occurred when we wanted to incorporate new sources of data which was not relational, and not always in our control. In one case this data was in XML format from a third party, in another the data was entered offline in a Microsoft Access(r) based tool, and then exported as XML. The data need to be seamlessly integrated into the "Legacy" SQL system, with minimal effect on the existing publishing/syncing workflows. In addition, the data is tightly linked to the SQL data such that changes in the SQL data can trigger changes in the new data. An additional challenge is that the XML data is a mixture of structured (indexing and cross referencing) and non-structured ("markup").

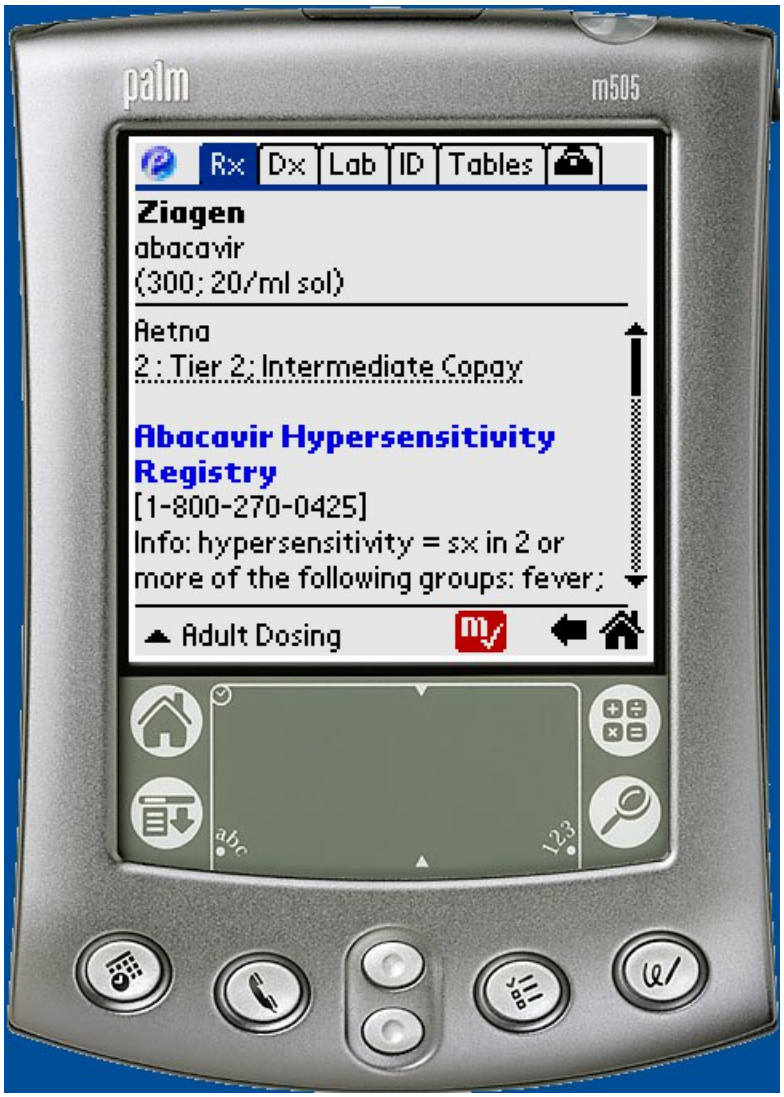
2.1. Key Characteristics of the Application

Epocrates Essentials is a multi-platform clinical reference application designed for handheld devices. The following screen shots exemplify some of the key issues involved which are not usually a primary concern on desktop or server applications, mainly due to limited screen space, CPU and memory. Shown are two screens from the "RX" application within Essentials.



This screen shows a list of available "monographs", when tapped open up a detail screen. Note limited amount of space for the list, not wide enough for long names.

Figure 1. Home screen of RX application.



The entire monograph is split into multiple scrollable pages with a combination of structured data and markup (rich document text).

Figure 2. One page of the detail screen for a single monograph in the RX application.

Key characteristics of the application include the following:

- Runs on handheld devices (Palm OS(r) and PPC).Future; runs on Smart Phones and other small format devices.
- Limited memory capability (8MB typical)
- Simple, very constrained database.
- - 64kbytes maximum per record
  - Single 16 bit record indexing.
  - No SQL, no string indexing, no compound keys.

- Limited CPU power on many devices.Highly efficient client side code required.
- Very small display (160x160 on palm devices) severely limits the amount of data that can be presented on one screen, and strongly effects data design decisions.
- Synchronization speed (time to update new data) critical.Number of records updated is primary factor in sync speed, followed by total amount of changed data.
- Very high performance required on "sync servers" to accommodate large number of subscribers synchronizing.
- Client code does not understand raw XML.
- "Linking" of related content.Sections of one monograph may refer to others (say by drug name).
- "Late Bound Linking".Not all linking can be resolved at the time the data is entered.For example, a link to a drug not yet in the database may need to be resolved later.

2.2. Key Characteristics of "legacy" data

The "legacy" data is non-XML data stored in a relational database. Key characteristics of the "legacy" data include:

- Highly structured and referential
- Stored in Oracle SQL database
- Constantly changing, primarily by manual editing.
- Specifically designed schemas and content for presentation on PDA's
- Entirely "in house" but very difficult to change workflow, representation or tools.
- Part of a very complex workflow for publishing data to large subscriber base.

2.3. Key Characteristics of "new" XML data

"New" data to be integrated into the database and workflow is XML data. Key characteristics include:

- One large XML Document containing many (sometimes thousands) of "monographs".XML documents may be very large (15MB - 150MB).
- Periodic updates (monthly, quarterly) with unknown amount of changes. Often 50% or greater of the monographs change, although with very little change within each one is typical.
- Schema may or may not be in our control.Not always the "ideally designed" XML schema.
- Schema likely to change unexpectedly.
- No control over content whatsoever.
- Referential data within monograph.Sections from one monograph may refer to other sections in the same or other monographs.
- Referential data to other data sources.XML content may "link" to content in the relational database; for example linking to drugs or lab tests by name.

- Both structural and "markup" elements.Some elements contain structure, indexing and annotation ('structural') while other elements contain textual content and 'markup' (bold, lists, paragraph format).

2.4. Legacy Publishing Workflow

The following is a simplified diagram of the legacy publishing Workflow, to which any new components must be integrated.

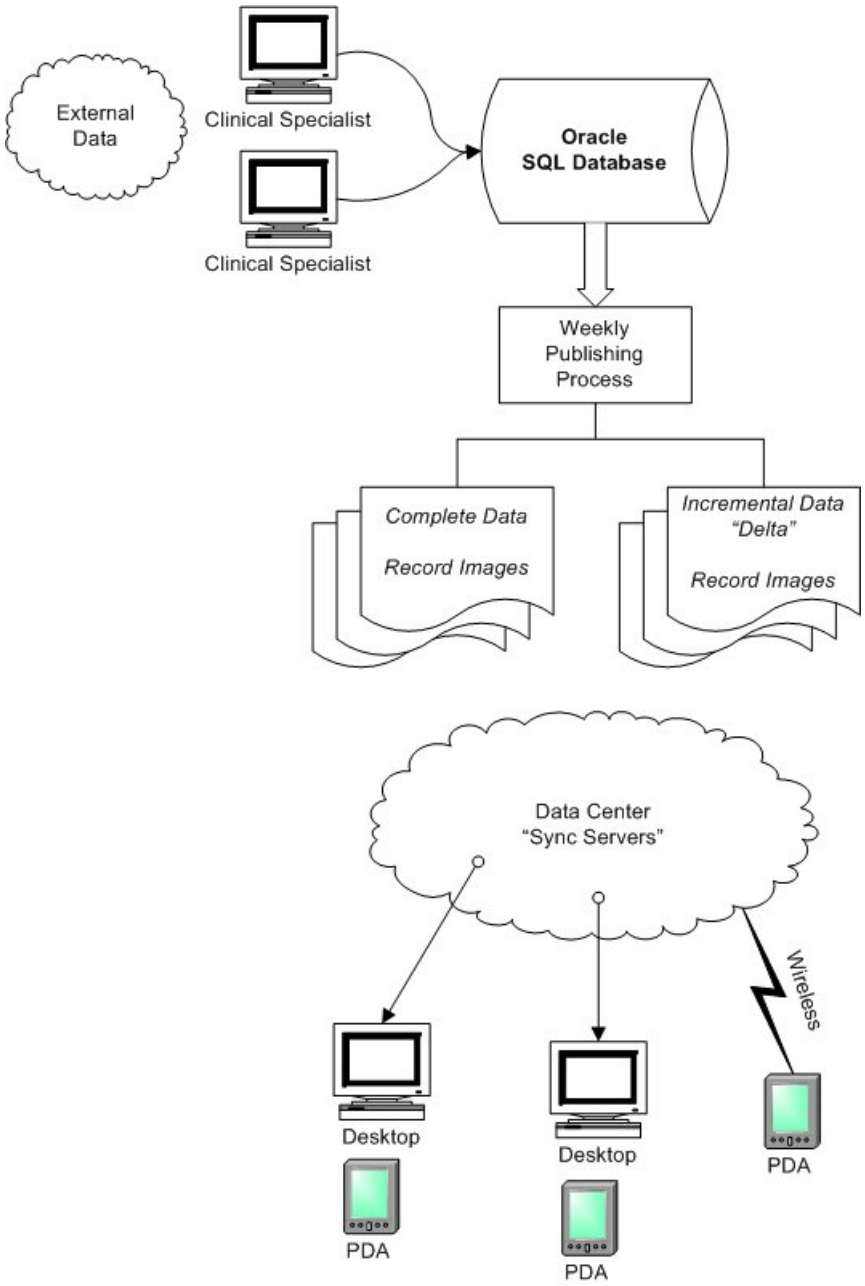


Figure 3. Legacy Publishing Workflow

The legacy publishing workflow contains the following key components

2.4.1. Manual Entry

Clinical Specialists peruse external data sources, apply judgment and manually enter data into the Oracle database using in-house data entry applications. Thus occurs continually and in parallel by multiple specialists.

2.4.2. Weekly Publishing Process

Weekly, a complex "publishing" process occurs which reads a snapshot of the database and produces two sets of output data (device database record images). This format is designed so that it can be efficiently sent to PDA's with minimal server processing.

- Full Data - This contains the complete dataset and is used for publishing to new subscribers, or subscribers who are extremely out of date.
- Incremental Data - This contains a weekly incremental ("delta") set of data used to publish to subscribers in an incremental fashion.

2.4.3. Sync Servers

The Epocrates Data Center contains a pool of "Sync Servers" which communicate with subscribers desktop or PDA's directly (in the case of wireless syncs). The sync servers are responsible for determining licensing, determining the proper subset of licensed content, selecting delta or full data, and pushing the required data to the PDA's.

2.5. Workflow Requirements

The following are the core requirements for integrating new data into the workflow.

- Integrates into current workflow with minimum changes to existing process and data structures.
- Reliable Change detection at the device DB record level.Required for producing incremental data accurately and efficiently.
- Support for deferred detection of dependencies.Dependency of data is not always determinable at the time the data is entered.
- Minimum performance hit on server during synchronization.
- Accurately manage changes when new data is updated.
- Minimum changes to DB schema or workflow if XML schema changes.
- Extensible design which can be reused with future content sources.

3. False Starts

The following approaches were attempted to solve these problems, but failed to address one or more requirements.

3.1. One Big Blob

An early idea was to simply store the entire XML as a single "BLOB" column in the dataset.

3.1.1. Pros

- Very simple solution, easy to implement

3.1.2. Cons

- Deferred almost all processing to the sync server, which is already the most loaded and performance critical component.
- Impossible to detect "changes" at the database level.Any change to any monograph causes a change to the entire set of data.
- Solved no significant problems over simply having the XML in the file system.
- No relational coupling to existing data.
- Difficult to search at SQL level.Blob columns are inefficient to search.
- No structure or indexing at the SQL level.

This idea was discarded early because it did not solve any significant problems.

3.2. Full Normalization

The DB team took a stab at "Full Normalization" where every XML element was mapped to a SQL Table, (plus additional tables handling the many-to-many relations). This was considered the "Ideal Representation" by the DB team.

3.2.1. Pros

- "Ideal" fully normalized data representation
- Full data referential integrity.
- Fine granularity of modification detection.

3.2.2. Cons

- Very large number of tables. Highly complex schema (over 150 tables).
- DB Schema is not resilient to minor changes in XML schema. An XML schema change in a single element or attribute propagates to all components of the system requiring a huge effort to manage.
- "Markup" type elements are very difficult to represent cleanly and simply as relational data.
- Extremely difficult to implement.Writing and testing the code to load the database, and extract the data would be a tremendous effort.
- Very bad performance.The primary performance factor in a client-server database application is the number of queries (not the amount of data transferred).Data for publishing to the device had to be constructed from a large number of queries and very complex SQL required.

This approach was discarded primarily due to its complexity, time to implement, and performance problems.

3.3. One Blob per Monograph

A refinement on the "One Big Blob" design was to split up the XML file into its component "monographs" and then store each as a single "BLOB" column.

3.3.1. Pros

- Monograph level changes managed in the DB.(One monograph can change and be detected without causing a full resync of all the data).
- Fairly simple to implement
- Corresponded well to the type of changes expected and mapped well to the record structure used on the device.

3.3.2. Cons

- Difficult to search at SQL Level
- Referential data not exposed at SQL level
- Significant processing deferred to sync server.

Although an improvement over the "One Big BLOB" design, this design was discarded due to it not solving enough problems to be useful.

3.4. XML Database

Oracle has a native XML Database extension which could be used to store XML directly.

3.4.1. Pros

- Provides an efficient and architecturally clean XML storage.

3.4.2. Cons

- No in-house experience with this technology
- Difficult to integrate with existing tools (JDBC, ODBC)
- Constrains ability to migrate to a new database provider in the future.
- Most of the code that needs access to the XML needs it in program space (e.g. as a DOM tree), not database storage.Database support for structured queries into the XML is not required.

This solution was rejected primarily due to no experience with the technology, difficulty with integrating to existing software, and considered a high risk with minimal benefit.

4. "Common Object Model"

To design a better solution, I took a step back and modeled the legacy SQL data, the new XML data, and an attempt to predict future data into one model, the "Common Object Model". This model is a Design Pattern for both SQL and XML clinical data at a level of abstraction focusing on only the detail needed to fit the data within our workflow and presentation structure.

This model focuses on the key components found in all our clinical data.

4.1. Document Structure

A "Document" is all data from a particular data source. In the "Legacy" data this is the relational data representing the entirety of the clinical database.

A Document comprises a Collection of Monographs, Internal and External indexing (cross referencing) and classifications ("Class" and "Subclass") information.

A "Monograph" is the data corresponding to a single "Topic" (Drug, Disease, …) and is composed of some attributes ("Title", "Sub Title", …), some structure (Section, Sub Section) and markup (semi-structured text).

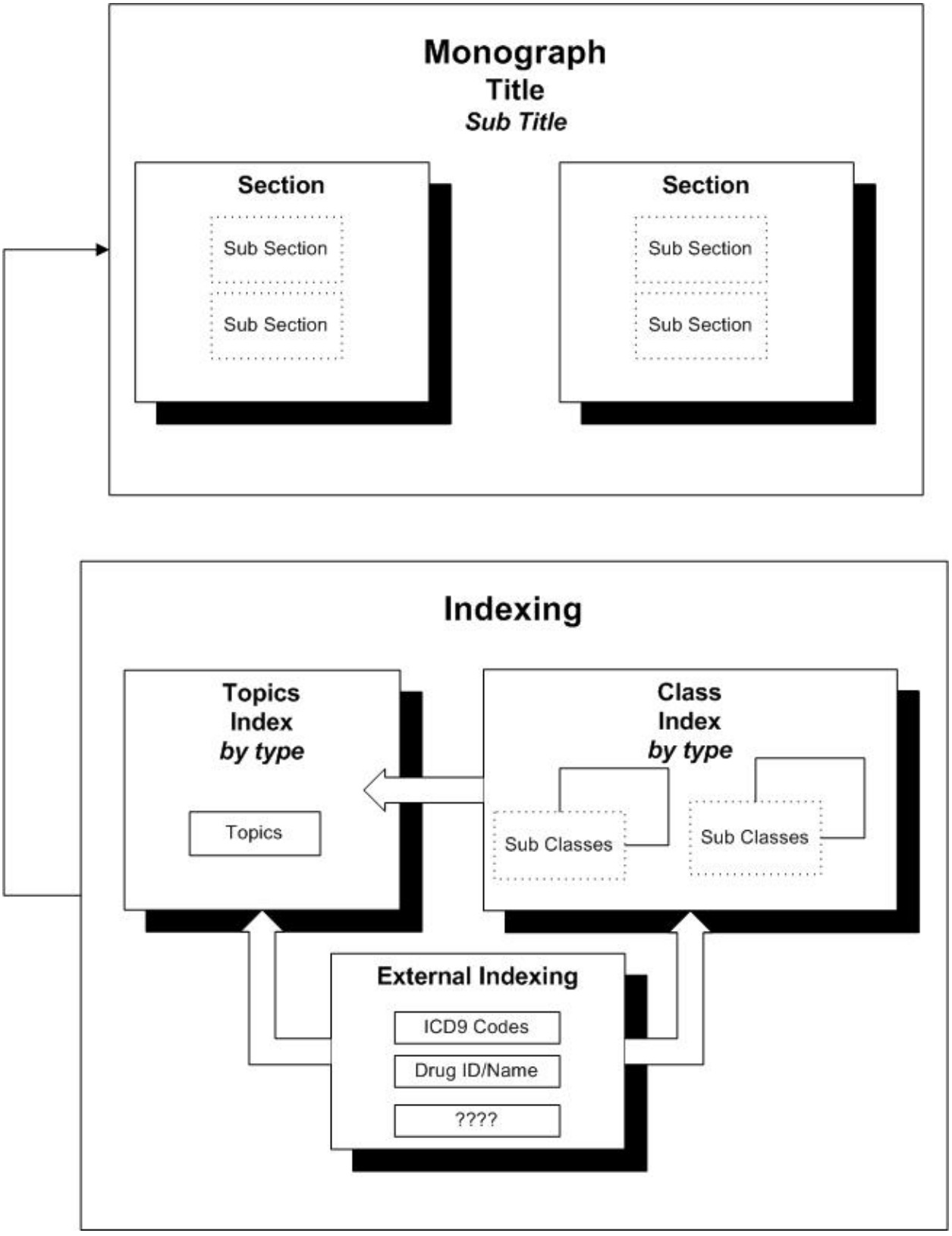


Figure 4. Common Object Document Structure

A key issue is that the components of a "Monograph" are generally stored in a single record in on the PDA, whereas the Indexing and Classification data is stored in other records. A special, but critical, consideration is that even when the monograph data is constant, the resulting device DB record can be affected by changes to the relational data, both in other monographs and outside its document (in other document sets). An analogy to this dependency is HTML "server side includes" where the HTML source document may be unchanged, but the resulting document is dependant on other data.

4.2. XML Mapping

A simplified model of how XML maps to this *Common Object Model* is diagramed below.

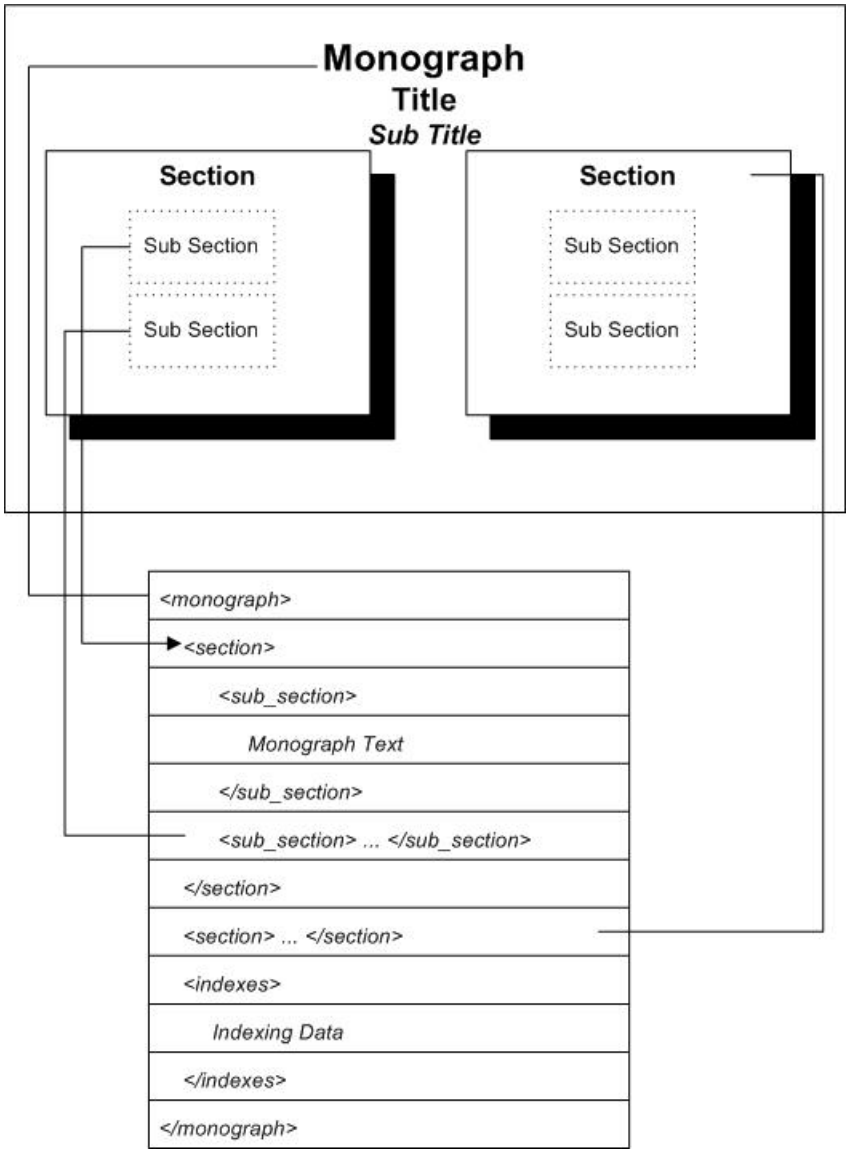


Figure 5. Common Object XML Mapping

This shows an idealized mapping of XML to the Common Object Model. In practice, the XML schema is usually not under our control, but follows a similar pattern, that of a collection of "monograph like" elements which contain structure (sections, sub sections) and markup, as well as separate elements which describe indexing.

4.3. Database Mapping

A simplified DB schema mapping for the Common Object Model is show below.

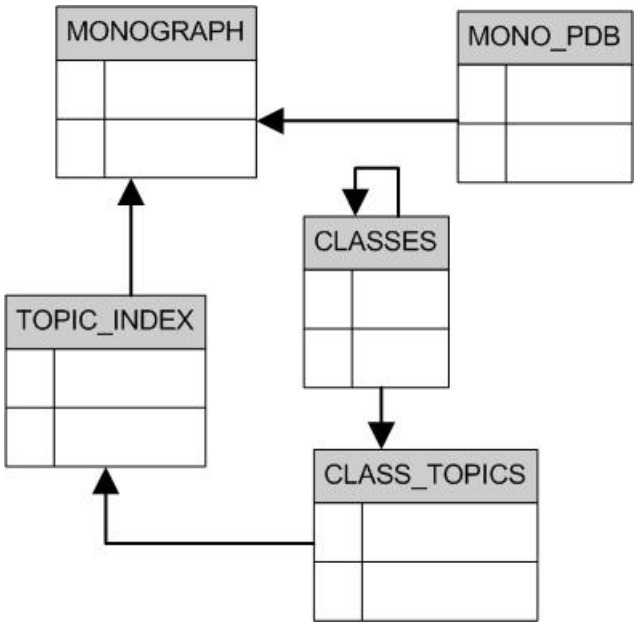


Figure 6. Common Object Database Mapping

This shows an idealized mapping from the Common Object Model to a Relational Database model. This structure is resilient to minor changes in the XML schema. It represents indexing and classification elements as explicit tables, and stores the entirety of the monograph as an XML Document Fragment within a blob field.

- MONOGRAPH Contains the XML representation of a single monograph along with key attributes and elements.The XML Document Fragment is stored in a blob field and key elements are stored as separate fields.
- MONO_PDB Contains the binary "device DB Record" representation of a single monograph.This record can be sent unchanged to the device.This contains a fully prepared dataset with complete document formatting and link resolution.It needs to be regenerated every time any changes occur anywhere in the database.
- TOPIC_INDEX Maps 1-N Topic Index elements to a MONOGRAPH
- CLASSES Defines the Class and Subclass structure and lists.
- CLASS_TOPICS Defines the collection of TOPIC_INDEX elements contained within a class.

4.4. Application Mapping

An example of how the Common Object Model maps to the real application is shown below. The *Monograph* is represented as series of scrollable pages. Each page corresponds to the *section* of the monograph. *Sub Sections* are repres-

ented as one or more paragraphs and are linked to by a list of the sub section headings at the top of each screen. The content of each *sub section* is markup which can include presentation formatting and links.

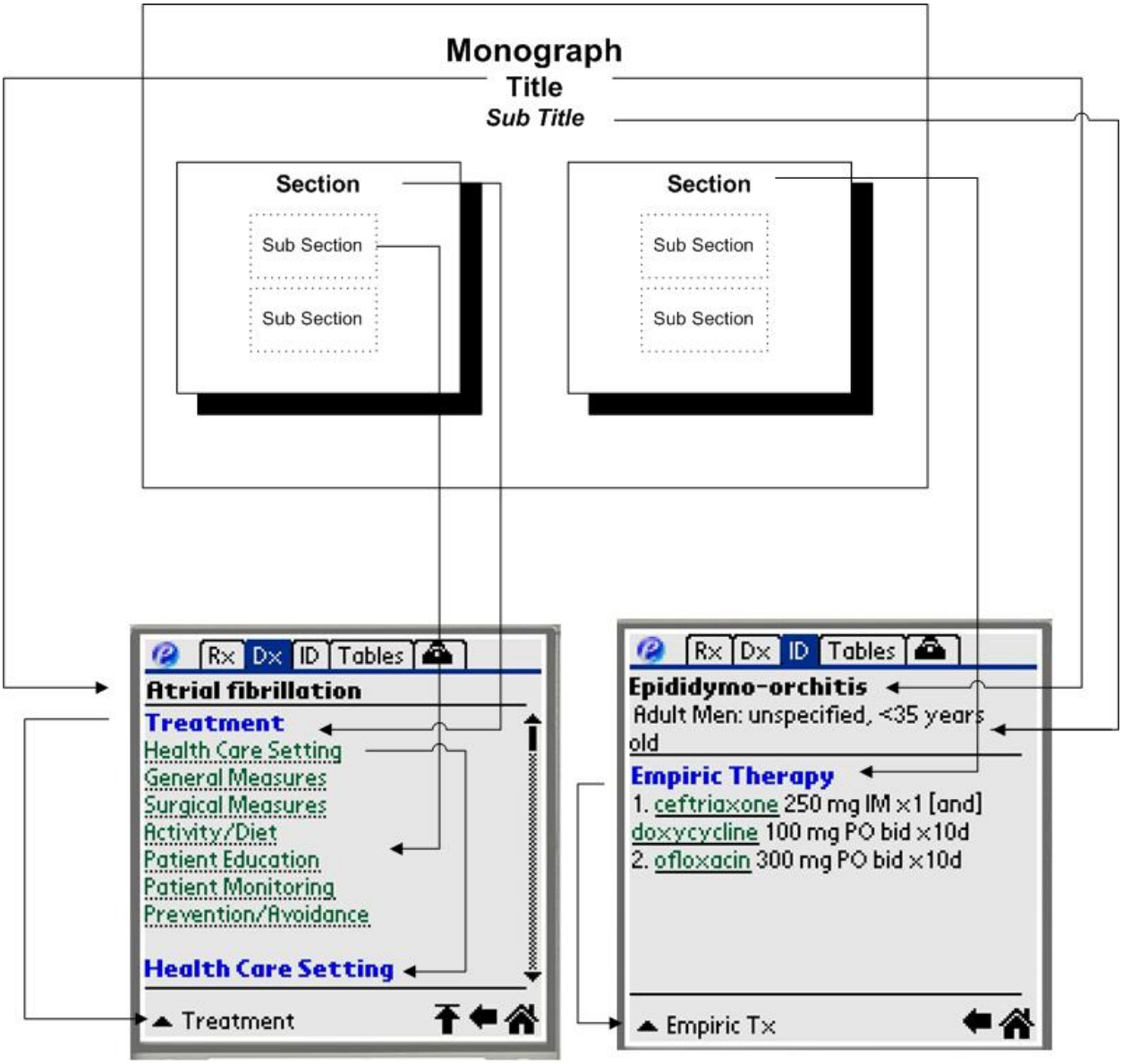


Figure 7. Common Object Application Mapping

5. Final Design

The "final" design chosen for integrating the XML data into the current system was designed using the Common Object Model as a Design Pattern incorporating ideas from the "One Blob per Monograph" design and adding direct relational support for the key components of a monograph which are structural or referential, primarily if they were needed *outside* the monograph itself.

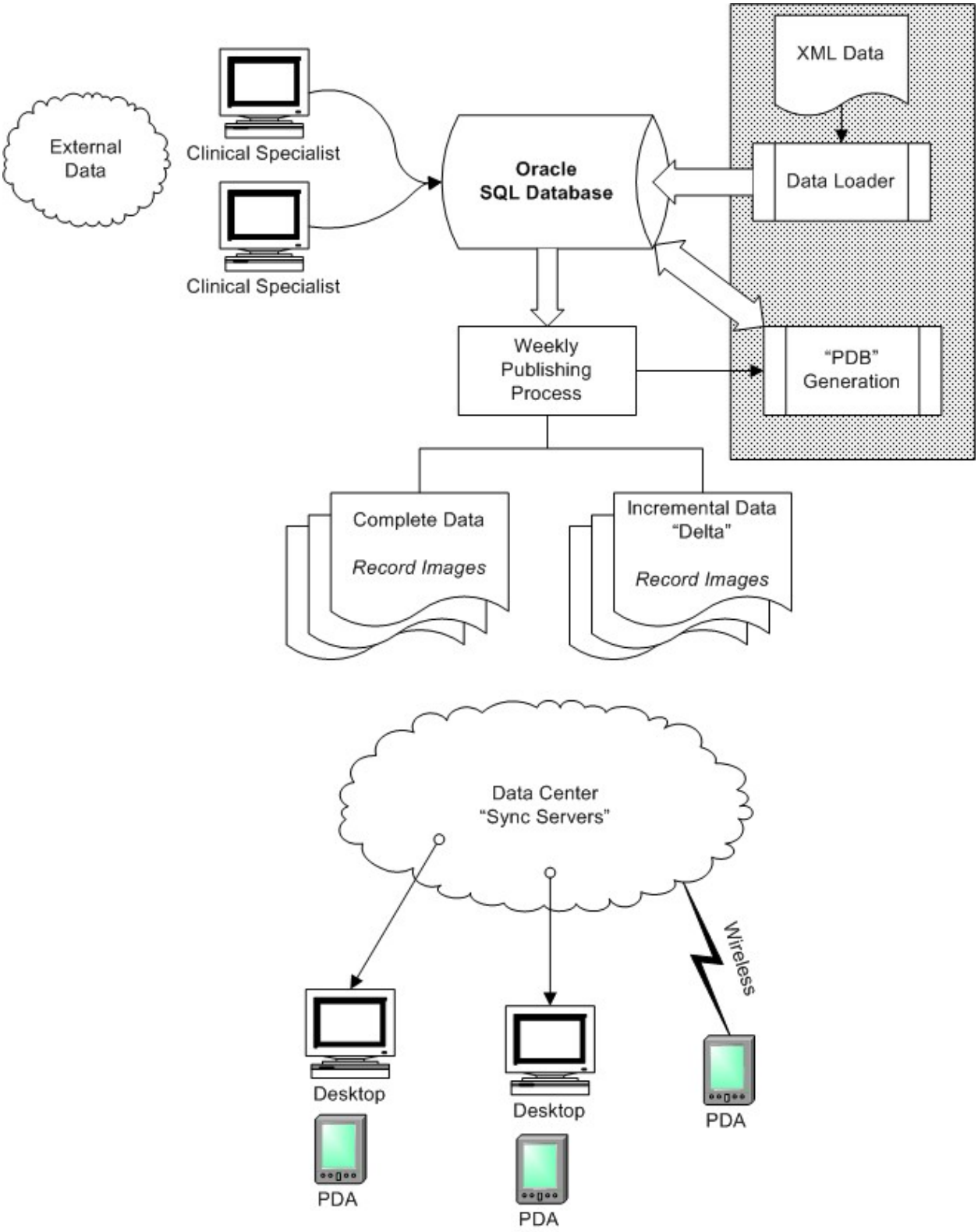
Data sources are typically large XML files (one or more). For each type of input data, the structure is analyzed to fit within the Common Object Model. In particular we look to segment the data into Monograhs, Indexing, and Classific-

ation components. Ideally, the Monograph component is an entirely self-contained XML Document Fragment and the Document can be split into a collection of these independent fragments, along with the indexing and classification data (which is either within each monograph, or refers to a different monograph within this document or in another dataset entirely).

5.1. Workflow Processes

The End-to-end workflow of incorporating XML data into the existing workflows required the addition of these major components. This same design pattern is reused with new sources of data, with changes to the specific implementation but following the same general pattern.

The modified workflow incorporating updates from external XML sources is diagramed below.



The gray area indicates the changes to the workflow for incorporating the new XML data.

Figure 8. Modified Publishing Workflow

The new components to the workflow are the "Data Loader" and "PDB Generation".

5.1.1. Data Loading

Data Loading is performed once for the initial load of a new data source, as well as every time new data is to be incorporated into the system. New data is loaded into memory in a java application as a DOM tree. The document is broken into two types of data, Monograph, and Indexing/Classification and stored in the database.

5.1.1.1. Monograph Data

Each monograph is extracted as a self-contained XML Document Fragment and stored in a single BLOB column. Primary attributes are extracted from the monograph XML. Examples of attributes which are extracted as columns include

- Unique ID
- Title
- Description
- Type
- Publish Date / status

The remainder of the monograph is considered "markup" and has no relational usefulness.

Some pre-processing is performed on the XML. The XML is loaded into a DOM tree, processed, then serialized out as a document fragment and stored in a BLOB column. This includes conversion of Unicode characters and entities to ISO8859, with mapping of non-supported symbols to special strings and fix up of some know problems in the XML (e.g. converting incorrect ISO country codes).

5.1.1.2. Indexing / Classification

Indexing and classification data, whether contained within the monograph document fragment, or externally (e.g. in a index or table of contents) is extracted and stored in normalized relational form. This data is useful in a structural/relational representation in the database to handle cross referencing across applications.

Examples of indexing and classification data include

- Table of Contents
- Aliases / Synonyms
- "Referenced by" attributes
- "Related monographs"
- Class / Subclass relationships

5.1.2. "PDB Generation"

"PDB Generation" occurs during the weekly publishing process (sometimes more frequently). A snapshot of the production database is taken and post processing of the XML stored in the MONOGRAPH table is performed. The post processing includes the following steps, after which the resultant device DB record is stored in the MONOGRAPH_PDB table as a blob.

5.1.2.1. Late bound linking resolution

Linking resolution which cannot be performed during the data loading is deferred until PDB Generation. This allows changes from other data sources which have occurred after the initial data load to be incorporated in the linking algorithms.

5.1.2.2. Document Formatting

The monograph XML is parsed and formatted into the format required by the client. Depending on the application this could be any number of client specific formats. As client software is improved this formatting logic and representation may be changed.

5.1.2.3. Serialization and Compression

All data to be stored in a single PDB Record is serialized. Some client applications allow compression of all or parts of the PDB record.

5.1.2.4. Smart Update

The final resultant record is compared (via an MD5 checksum) to the last stored record for that monograph. Only if there are changes is the database updated. This allows the next stage of the weekly processor to accurately create differential PDB's, optimizing the amount of data required to be sent to the client during sync time.

6. Post Mortem

This design pattern was designed when we had only 1 new source of external XML data. It was designed with the intent of supporting other sources as needed. Since the initial rollout, there have been 3 new data sources of entirely different classes of data which were not foreseen at the time of the original design. The design pattern has been applied to all of these data sources with very good results. Each new set of data required some modifications to the details of the design to accommodate special needs and the structure of the data; however the overall design pattern has shown to be robust.

Of particular note, one of the new data sources had no markup data whatsoever, it was entirely structural, indexing and classification data. In another case there was some markup type data but it was not needed for this project so was discarded during the data loading process. In both these case the same design pattern was used, however there was no "Monograph" required, hence no need for data in XML format to be stored, rather the entire Document was mapped to relational data.

6.1. Lessons Learned

6.1.1. Split up Large XML files

Most XML tools have a horrible time with large XML files. They seem to be designed for "10k" files. When given a 10MB file or a 100MB file they are useless. Look for ways to split up XML documents into smaller manageable chunks. As part of the "Data Loading" we use a Java program to read the big XML file into a DOM tree then extract out self contained document fragments and serialized them as standalone XML. This has worked very well.

6.1.2. Don't assume "All or Nothing"

If you find that an "ideal" solution of storing XML in a relational database using full normalization of every element is unwieldy, don't give up. Storing some parts of the XML as structured data, and others as "BLOBs" can work quite well, especially if you process the XML components in a programming language that supports XML well.

6.1.3. Process XML within a programming language

Even though you store XML within a relational database, consider doing processing of the XML within a programming language that supports XML well. We found that using SQL, even stored procedures, to do complex processing was vastly more complex and a huge performance hit compared to using Java. For example, attempting to do string matching of drug names embedded within markup against a drug database was 1000x slower as a stored procedure, and much more complicated, then by loading the entire XML and drug data into a java program, processing the data in memory and writing it back, even when dealing with tens of megabytes of data and using a slow database connection.

6.1.4. Look for the distinction between "Structure" and "Markup"

Much of the XML we receive is a mixture of "Structure" and "Markup". For example, one dataset is essentially a conversion from printed format to XML with very little structure added. By modeling Structure and Markup separately you can optimize the representations for each. Structure is generally useful as separate relational data, whereas markup is generally useful only for presentation. Sometimes structure (or annotation) is embedded within markup (for example <A> tags in HTML) and can benefit from being extracted and stored in separate table fields.. Recognizing the difference can make the integration to the Legacy World a lot easier.

6.1.5. The ‘Real World' is a compromise

Even pure XML data is *rarely* ideal. In our case, none of the XML data sources originated as XML, rather they are exports of other data, whether in our own tools or 3rd party tools. Often the XML schema represents the original structure (such as relational, or document) rather then a structure which would be used if the XML was authored directly.

When trying to integrate XML with legacy data it is *never* ideal. Recognize from the beginning that your solution will be a compromise. Look for ways to add minimum friction to your existing schemas and workflows. Look for common abstractions even when the details vary vastly. Don't be afraid to mix technologies when appropriate.

Biography

David **Lee**
Epocrates, Inc. [http://www.epocrates.com/]
727 Poplar Lane
Jasper
Indiana
47546
United States of America
dlee@epocrates.com

David Lee has over 20 years experience in the software industry responsible for many major projects in small and large companies including Sun Microsystems, IBM, Centura Software (formerly Gupta.), Premenos, Epiphany (formerly RightPoint), WebGain. As senior member of the technical staff of Epocrates, Inc., Mr. Lee is responsible for managing data integration, storage, retrieval, and processing of clinical knowledge databases for the leading clinical information provider. Key career contributions include Real-time AIX OS extensions for optimizing transmission of real-time streaming video (IBM), secure encrypted EDI over internet email (Premenos), porting Centura Team Developer, a complex 4GL development system, from Win32 to Solaris (Gupta, Centura), optimizations of large Enterprise CRM systems (Epiphany), implementation of ecommerce systems for on-demand digital printing and CD replication (Nexstra).