Open XML Court Interface (OXCI)
Architecture Proposal

Document version: 1.0
Author: Richard Himes

Abstract

This document is a draft proposal for an approach to the design of the Open XML Court Interface (OXCI). It is submitted as a seed for comments so that the design group can discuss and agree on a general methodology.

Table of Contents

1. Introduction

The design of OXCI will be a team effort involving participants with a wide range of experience and technical ability. It is desirable to tap into this experience at every phase of development. The intent of this document is not to cast ideas in stone, but rather to serve as a point of discussion. The goal is to use the best design practices.

2. Architectural Requirements

This system will be used by courts and vendors with vastly different architectures, so the design needs to be flexible. It is desirable for the design to be language-independent and object-oriented. The architecture should be defined in terms of a generic API (application programming interface) rather than specific code. The design should be clear, not mysterious. To some extent, these requirements (abstraction and clarity) conflict. Thus, the design needs to be presented from several different viewpoints.

3. OXCI Requirements

OXCI will serve as the Electronic Filing Manager (EFM) module in diagram 3.1.

Figure 3.1

EFP is the electronic filing provider. It represents the client (attorney) side of electronic filing. The EFP wraps the filing(s) in a Legal XML envelope and transmits it to the EFM. There can be many EFP systems communicating with a particular EFM. The method of transmission is unspecified, but could be via e-mail, HTTP, FTP, or other means. An EFP must submit filings using the Legal XML Court Filing data format standard.

The CMS is the case management system. It represents the court side of electronic filing. There will probably be only one CMS per EFM, but theoretically, the EFM could distribute filings to multiple CMS systems in a state or region. The CMS is responsible for accepting or rejecting a filing. Other standard CMS functions (docketing, reporting, workflow, etc.) are outside the scope of this specification.

The requirements of OXCI (EFM) include:

- Listen (or poll) for and accept a filing from an EFP
- Authorize the EFP vendor and connection
- Perform basic policy validity checking
- Translate the filing for the CMS
- Notify the CMS of the filing
- Accept the confirmation or response from the CMS
- Translate the CMS confirmation for the EFP
- Forward the confirmation to the EFP

Basic policy validity checking will require project coordination with a separate design effort know as Court Policy Management. Remaining policy will be enforced by the CMS or human intervention. Sample basic policies are “Accept only one filing per message”, “Do not accept external document references”, and “Must be on list of accepted document types for this court”. The CMS or clerk staff may perform further checks such as “The filing attorney is not a bar member”, “The filed document doesn’t conform to court local rules”, or “The fee payment has not been authorized”.

Translating a filing for CMS may involve reformatting the XML and writing it to a DMS (document management system). This will vary from court to court. Figure 3.2 includes the CPM (court policy management) and the DMS system objects.

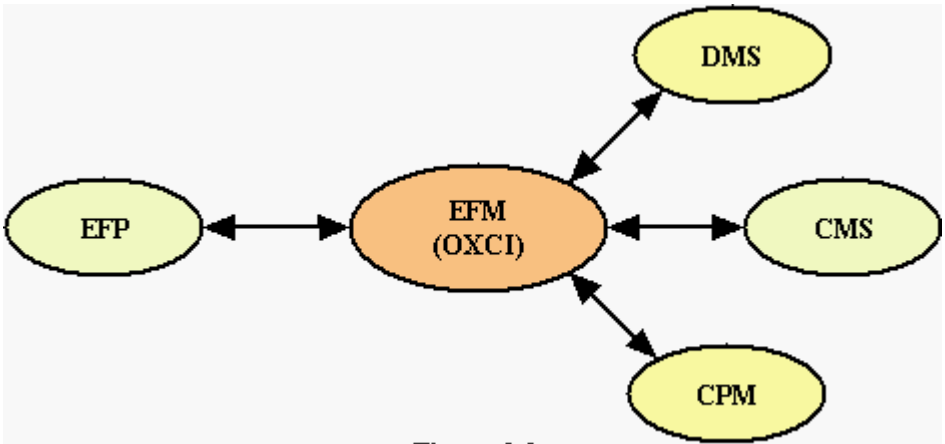

Figure 3.2

4. Objects and Interfaces

Objects will be defined based on UML (Unified Modeling Language) constructs and the OMG IDL (Object Management Group Interface Definition Language.) IDL will occasionally be augmented with sample Java implementations for clarification.

5. Top Level Objects and Interfaces

From the diagram in figure 3.2 and the high level requirements, we can construct figure 5.1, the top level class diagram:

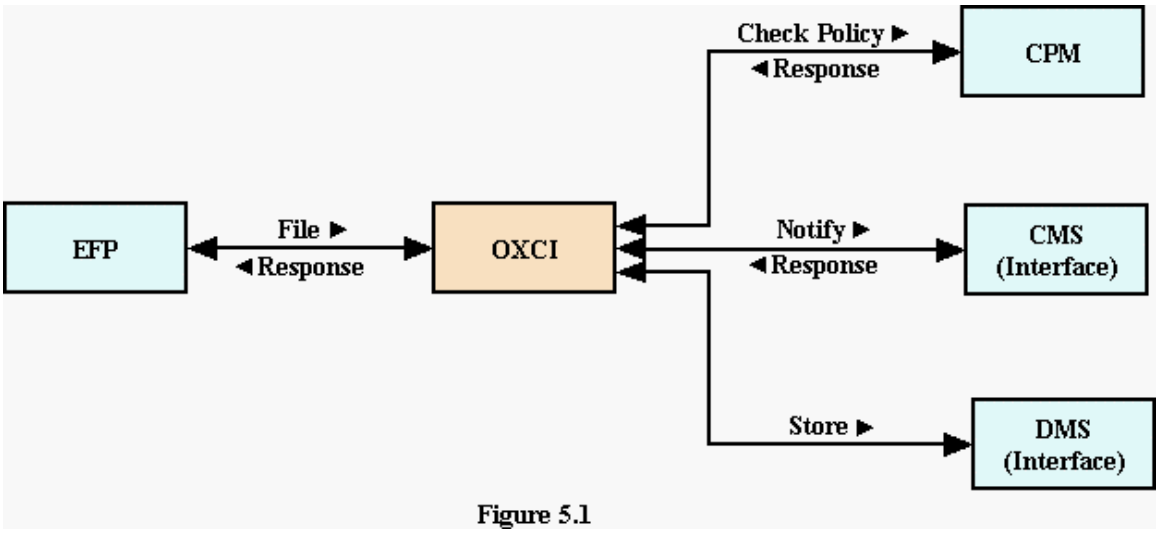

Figure 5.1

Note: A generic response is indicated. As currently defined in the Legal XML Court Filing specification, the response is a confirmation XML structure which indicates that the filing was received (filed), accepted (docketed), partial (portions of the filing(s) were rejected), deferred (waiting human review), and rejected (with reason text).

Note: CMS and DMS are shown as "Interfaces" because these objects will not be the actual court CMS and DMS. They will format information suitable for processing by the local CMS and DMS and will be court-specific, but will process standardized messages from the OXCI implementation. The word "interface" here has a different meaning than the design specification application programming interfaces (APIs) of OXCI. The context of the former is an interface by a particular OXCI implementation to another system. The context of the later is a language independent API for OXCI methods.

Figure 5.2 shows an interaction diagram for a filing. Forward time is directed downward in the diagram. The asterisks before messages (e.g. * Store Filing) indicate repetition. Note that there may be more than one filing in a CourtFiling message, so these messages are repeated for each filing.

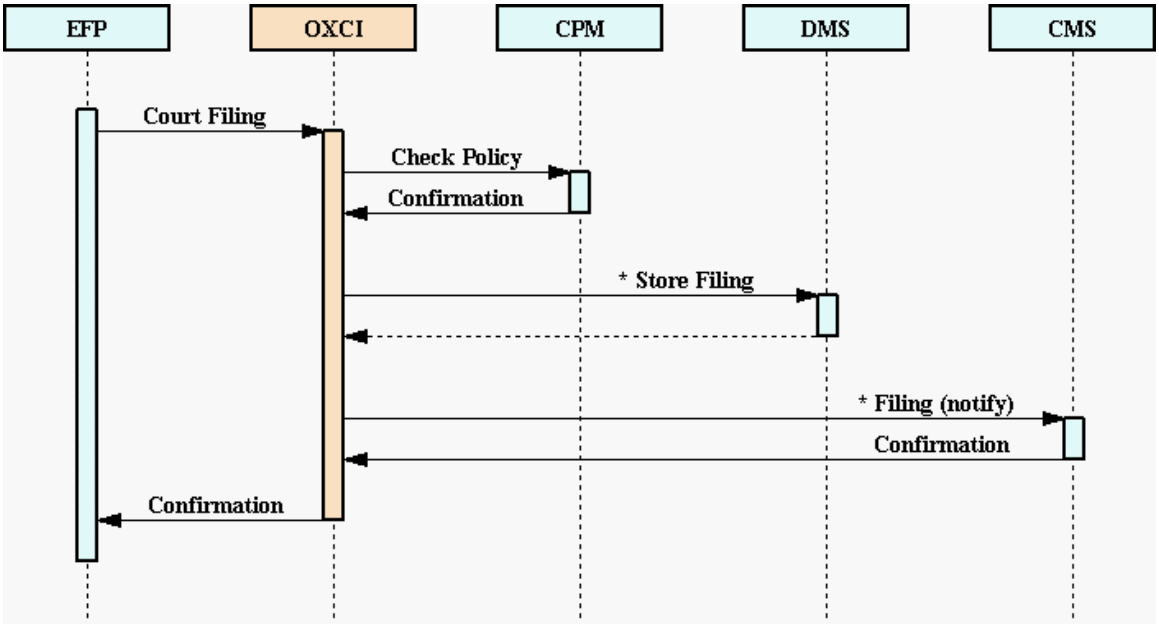

Figure 5.2

The interface Oxci shown below contains a submitCourtFiling member function (also known as a message or a method.) It is passed a CourtFiling object (see section 6.) It returns a Confirmation object (not detailed in this document.) The "in" means that courtFiling is an input parameter, that is, it is read only.

```
interface       Oxci  {
        Confirmation   submitCourtFiling ( in CourtFiling courtFiling);
}
```

The other top level interfaces will be defined in a similar fashion. Note that a program invoking this object doesn't need to know how this function is implemented (coded) internally, the programming language that is being used (IDL can be compiled into numerous languages), or even the location of the object (it could reside anywhere on the Internet.) The only visible portion for systems using this object is that it contains a function called submitCourtFiling that accepts a CourtFiling object and returns a Confirmation object. Thus, we only know how to invoke this function, and what we can expect in return. The CourtFiling and Confirmation objects, of course, will represent the corresponding XML documents defined in the Electronic Court Filing standard. In actuality, what is received at the court server is a message in LegalEnvelope format, which may contain a CourtFiling element. Thus, there is another level of processing that has been omitted in this discussion for sake of simplicity.

6.0 XML Objects and Interfaces

All of these system objects will need to access portions of the CourtFiling XML elements. To facilitate this access, each element in the XML document should be defined as an object. All XML elements will be derived from an OxciElement, which implements the DOM interface Element (see http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html) There will be a similar definition for OxciDocument, which isn't discussed here.

```
interface       OxciElement : Element {
        void            setElement (in Element element);
        Element         getElement ();
        Element         getFirstChildElement ( in wstring tagName);
        Element         getNextSiblingElement ( in wstring tagName);
}
```

The member functions are:

- setElement – Set this object to represent a specific instance of an element
- getElement – Retrieve the element represented by this object
- getFirstChildElement – Get the first child element having this tag name
- getNextSiblingElement – Get the next sibling element having this name

These are just sample member functions, and more will be defined.

The CourtFiling element is defined as:

```
interface       CourtFiling  : OxciElement {
        ElementEnumerator   getFilingEnumerator();
}
```

The CourtFiling interface implements (is a subclass of) the OxciElement interface, so it inherits all of OxciElement functions. Other elements of the Legal XML Court Filing Standard will be defined in a similar manner. The function getFilingEnumerator returns an object of type ElementEnumerator, which allows the multiple Filing elements contained within CourtFiling to easily be obtained one at a time:

```
interface       ElementEnumerator() {
        boolean         hasMoreElements();
        OxciElement     nextElement();
}
```

Below is sample Java code using the concepts that have been discussed. It is assumed that if there is an error, a function will throw an exception (this facility is beyond the scope of this discussion.)

```
public class Oxci_ implements Oxci {

        public Confirmation submitCourtFiling(CourtFiling courtFiling) {
                //* Setup code omitted
                confirmation = cpm.checkPolicy (courtFiling, confirmation);
                filingEnumerator = courtFiling.getFilingEnumerator();
                while (filingEnumerator.hasMoreElements()){
                        filing = (Filing)filingEnumerator.nextElement();
                        dms.storeFiling(filing);
                        confirmation = cms.notify(filing, confirmation);
                }
                return confirmation;
        }
}
```

Note that confirmation is passed to checkPolicy and notify as an input object. This is because Java doesn't use output parameters for integrity control. Thus, all parameters will be defined as "in" in IDL. The confirmation object is constructed during different steps in the process. For example, confirmation for a single filing will be set as each filing is processed (the Confirmation object "confirms" multiple filings.)

7.0 Conclusion

This document presents the basic design concepts proposed for OXCI architecture. The sample architecture was simplified to enhance communication of the ideas, and the final design will differ from these examples. An attempt was made to use tools that represent best of practice to define the architecture. However, this is a moving target, and suggestions are welcome.

8.0 Glossary

CMS – Case Management System
CPM – Court Policy Management
DMS – Document Management System
EFM – Electronic Filing Manager
EFP – Electronic Filing Provider
IDL – Interface Definition Language
OMG – Object Management Group
OXCI – Open XML Court Interface (serves as EFM)
UML - Unified Modeling Language
XML – Extensible Markup Language

Please send comments to Richard Himes <rhimes@nmia.com>