Markup UK

Building a cloud-based visual operating system entirely based on XML

Leveraging the capabillities of the XIOS/3 and CloudBackend platforms

Daniel Arturrson, CloudBackend.com

Martin Nilsson, xios3.com

Interoperability and extensibility are keywords for XML. Why are we not seeing the same for software applications, web applications, mobile apps, and operating systems in general? Why can't I run an iPhone app om my Samsung TV? Why isn't Mac applications possible to run on my Windows computer? If we had interoperability and extensibility in a similar way that XML provides for data for software, we would not have any of these problems. Software would run across all types of devices, screen form factors, and operating systems – across desktop, mobile, smart TVs, and the infotainment systems of cars.

This paper discusses how to use the XIOS/3 Edge Application Platform and the CloudBackend Singularity Database to create a new XML - and cloud-based operating system complete with productivity applications, software development tools, and a file system – including extensive support for XML. While CloudTop XML OS is still under development, this paper provides a snapshot of the current state of the implementation. It challenges the perception of what XML can be used for.

1. Introduction

Many years ago, we set out with the goal to simplify development, making applications work across devices, and use XML to build applications instead of traditional programming. With a background in building an XML Application Server and an XML-based web server we had seen the promise of XML, and how the hierarchical structure of XML itself helped to solve many recurring problems. The time had come to apply XML to software that was supposed to run on Windows, Mac, Linux, mobile devices, and other types of devices.

We wanted to break out of the boundaries enforced by native compiled code for a specific processor architecture and the underlying operating system. We wanted applications that would run equally well on Windows, as on Mac or Linux. We wanted applications that could change their form factor to also run on mobile devices. We wanted the applications of the operating system to store all its data in tagged XML markup that nourished interoperability between applications and long-term storage and retrieval of information while simplifying search.

We concluded that it would not be good enough to create a cross-device, cross-operating system development framework, we needed new data storage technology to replace the outdated xomputer file systems with an XML repository, a new set of productivity

applications for office workers that breathed XML, and we needed a collaborative XML-based operating system to tie it all together to complete our vision.

This is a long-term vision and we are not at the end of it yet, but we are step by step humbly getting closer to realizing it. We got the XML development platform with XIOS/3, we got the XML repository and Singularity Database with CloudBackend. What we are missing is the completion of the productivity applications and the cloud-based XML operating system, CloudTop.

Figure 1. CloudTop cloud-based desktop and applications built entirely in XML

CloudTop is the answer to the question, how would you design a new operating system if you knew the Internet and XML existed? The OS would be delivered over the Internet and always be up-to-date without system updates. Any applications written in it would allow for collaboration with other people over the Internet. All information stored in the OS would be in XML format and any data exchanged for collaboration would be in XML. Applications would be able to open each other's data and transform it into its required format, possibly allowing a word processing application and a presentation application to only be two different views into the same XML document.

This paper will introduce a few applications built for CloudTop and some of the challenges with using XIOS/3 and CloudBackend to build CloudTop.

2. Finding a cure to the chaotic software landscape

Curing today's mess with incompatibility across operating systems and devices will require quite some dramatic changes. It will require software to be rewritten from the ground up using XML instead, data storage formats to be migrated to XML, and the willingness to embrace openness, as with XML your client source code will no longer pretend be a secret. Anyone may open the source code of your application and learn how to make their own, like how HTML works. If the server allows it they may also use XLink to link into your application and create a composite application or mashup using parts of your application.

The cure is nevertheless needed. Software-as-a-Service (SaaS) companies, enterprises, and mobile app developers spend far too much money on building web applications, iPhone apps, Android apps, and even possibly locally installed desktop applications. Rebuilding the same application for a multitude of form factors and operating systems is

expensive and time-consuming. Using XML only a single truth would be needed, a single source code that could feed all devices. This would reduce time to market through reduced development time, but it would save an equal amount on maintenance costs. Making the trade-off to make the code open in XML, might enable companies to build applications that they otherwise would not receive a budget for. Since many companies already have migrated their software to web versions built in JavaScript, which have an open visible source code, the openness part of the equation might prove to be less difficult than we initially imagined.

The last part is that for a new operating system to gain traction, it needs to have enough software with expected functionality to make a shift possible for an end-user. Gaining a critical mass of available software and making it easy to build software for the OS is therefore a crucial component. Our ace up our sleeve is the thousands of already available XML applications out there for almost any imaginable use case. If we can make them easy to register in the underlying XML repository of CloudBackend and then allow a user interface to quickly be built on top using XIOS/3, we would find a shortcut to get mission-critical software available on the operating system.

Effectively CloudTop makes its users create XML without seeing it. Any application running on the platform is thus like a custom XML editor, tailored for its specific XML application. The authored XML is stored in something that feels like a file system for the user, but in reality is the CloudBackend Singularity Database, which in the context of CloudTop acts as an XML repository.

3. What is CloudTop really?

CloudTop is a new cloud-based operating system and virtual desktop surface written in XML on top of XIOS/3 Edge Application Platform and the CloudBackend Singularity Database. It is delivered through any traditional web server without any server dependencies. It then runs within the web browser where the entire desktop surface is created from XML and additional applications are running as separate windows on top of that, all within a single web browser tab.

The type of virtual desktop provided by CloudTop is not a traditional operating system virtualization technology like Citrix or Microsoft Terminal Server. Traditional virtual desktop technologies run the host operating systems in the cloud on a server and then screencast the desktop and applications into a web browser or dedicated native client. With CloudTop the OS and its applications are all executing and running within the web browser thanks to the device edge computing capabilties of XIOS/3. The cloud and server are, thanks to CloudBackend, transformed into an authentication engine to load, save, and coordinate delta changes of XML documents for collaboration purposes.

Applications built for CloudTop can run entirely within the browser if built in XML and only using XML documents as its storage format. They can also interact with APIs in the cloud like XML Web Services (SOAP) or REST APIs (JSON or XML). Having portions of the application logic in the cloud behind an API will of course make the application more sensitive to disruptions and introduce latency for waiting on server responses. However, all user interactions can be quickly resolved and managed on the client, since the remaining application logic is written in XML and executed within the web browser on the device.

CloudTop has things you normally associate with an operating system such as a file system, processes, applications, security, users, command line interface, caches, user interface rendering, SDK for software developers, and network communication. It does not however include the lower levels of an OS like the kernel, BIOS, and things needed to make it boot a computer. Thus CloudTop can be said to include the higher levels of a traditional OS. Our reasoning here is that the most effect for end-users and developers is achieved at the higher levels, software SDK and development tools, and communication

with the outside world. The lower levels can be reused from Linux, Android, Windows, or any operating system. CloudTop only needs something that boots and opens a web browser like the open source WebKit browser and then starts CloudTop in fullscreen mode. One implementation of such an environment is the Google Chrome OS. The added benefit of this is that anyone with a web browser may run CloudTop without installing anything. VPN and security protocols can also be added to the lower-level boot OS, though relevant features are constantly added to the browser environment.

4. Changing the perception of a computer

The applications, the run-time state, the data, and the delta changes to data are all XML. This allows the run-time state of applications to be synchronized across devices and across different logged-in identities. The effect of this is that applications almost become like a virtual machine or container that can move between two physical servers while running, i.e. application virtualization makes moving applications between devices possible thanks to XML.

If you can move a running application window from one laptop to another, or to your phone, your computer is essentially no longer running on a single hardware. It is in the cloud and can manifest itself on any device and treat it like a piece of glass. Multiple devices can together form a multi-screen setup and parts of an application can run on different devices for a more user friendly-experience.

If applications can float freely between all devices and they together make up your total computing experience, then what is a computer? The XML-based cloud operating system makes this possible without any effort for the software developer to build an application. We see this as the true promise of cloud computing and ubiquitous computing, using the device's local processor, but treating all devices as screens to your cloud computer - pieces of glass.

5. Verifying our assumptions

While building on the CloudTop, we needed to verify that the cloud OS really could be used create everything from a presentation application to an Integrated Development Environment (IDE), to applications like mail, calendar, instant messaging, and photo viewer. Was really all the UI components needed to build such complex applications available in XIOS/3 XML languages for developing? Could the CloudBackend Singularity Database really serve the content needed for emails, calendar, photos, and word processing?

Along the way we have built some 30 applications on the platform and feel confident that the concept of applications being developed in XML and executing in XML works. Both from a development perspective, it is convenient to develop these applications and possible to maintain them, but more importantly, that it provides fast enough applications. It would be pointless if we had all the goodness of XML, but the applications became unbearably slow.

6. Looking through a few of the sample applications built

There would not be enough space in an article to go through all applications that have been built for CloudTop, but we will go through a few of them as each one of them introduces interesting and important concepts for the CloudTop OS and how it makes use of XML.

- XMLPad - Data Manipulation and Transactions
- Kanban - Hierarchical Data Model
- Contacts - Key/values, Meta-data, and Datatypes

- CloudTop - Combining Applications into a Desktop

The above list provides a brief introduction. Additional information is available through the CloudBackend service and the SDK Web UI (XIOS/3) and CloudBackend developer documentation. Using CloudBackend you can write and run applications on your own.

CloudTop is a separate project and not part of either the CloudBackend and XIOS/3 companies. They aim to release a new version of the CloudTop desktop using the latest XIOS/3 and CloudBackend by the end of this year. Any application developed as a single-page application on the CloudBackend Singularity Database today can run within the CloudTop desktop when it is released.

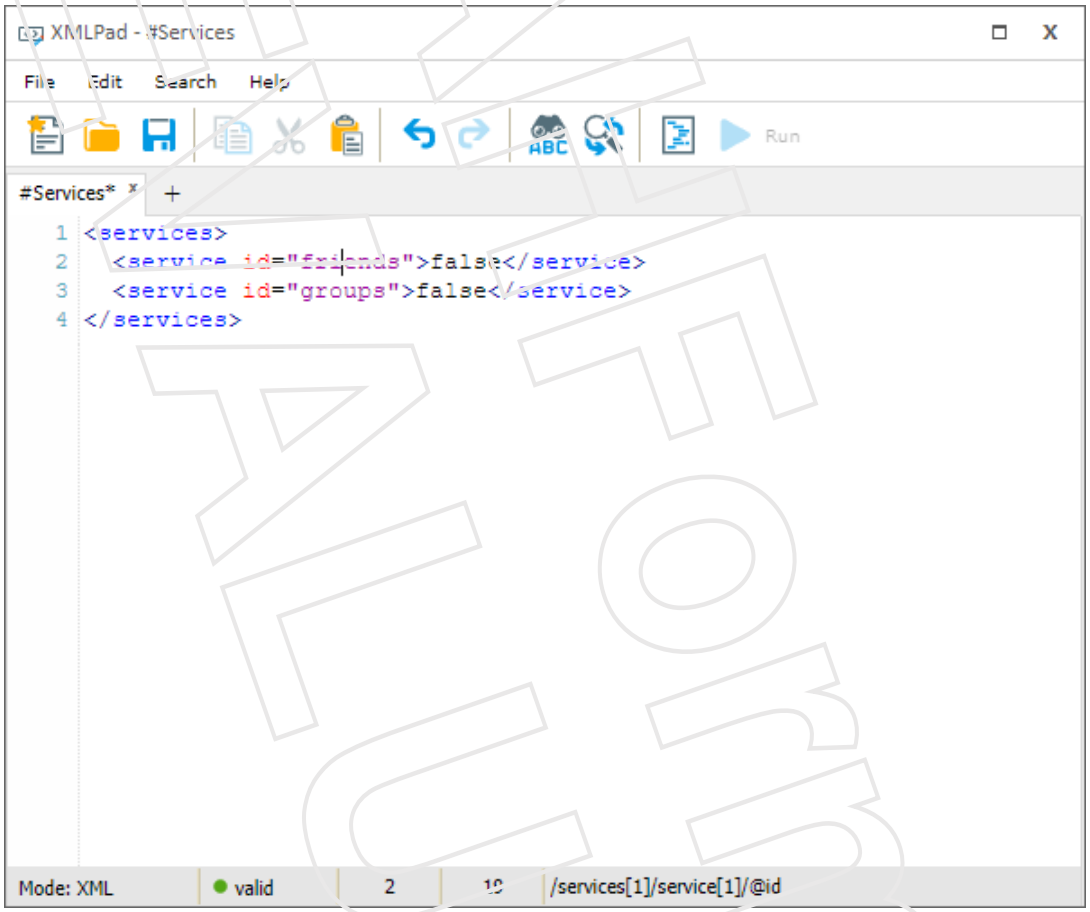
7. XMLPad - Data Manipulation and Transactions

One of the most important productivity applications on any system is the ability to write text in various forms, from the humble notepad to WYSIWYG (what you see is what you get) word processors, and code and markup editors with support for syntactic and semantic validation. Several such applications were made using XIOS/3 with various levels of sophistication, but let us look at “XMLPad”, a simple XML editor. At its core it is a presentation and input component that is bound to a document in memory. Every change in the document is automatically reflected on the screen for the user, but any input the user makes goes through the XIOS/3 transaction engine. These changes are expressed as a series of modifications at points in the document pointed out by XPath expressions. This allows anyone else interested in this document to get notified with only the delta of what has changed. If the document is shared with another user or device only these changes need to be transmitted.

There are many hidden details and edge cases on regulating change frequencies and coalescing change sets, but one of the more interesting for general use is how to handle syntactically invalid XML markup that inevitably happens when a user is typing. To solve that the offending section of the document can be wrapped in a special “not valid” namespace and encoded with entities or wrapped in a CDATA block, and can then be handled as any other XPath-indicated XML change set.

Besides the advanced applications like having servers or other clients apply these transactions elsewhere for collaborative applications or auditing change logs, they also can serve an immediate use for the application developer. A transaction log on a document can serve as a undo buffer if the transactions are applied in reverse. If then those transactions are moved to a different log they serve as a redo buffer.

Figure 2. The XMLPad application.



8. Kanban - Hierarchical Data Model

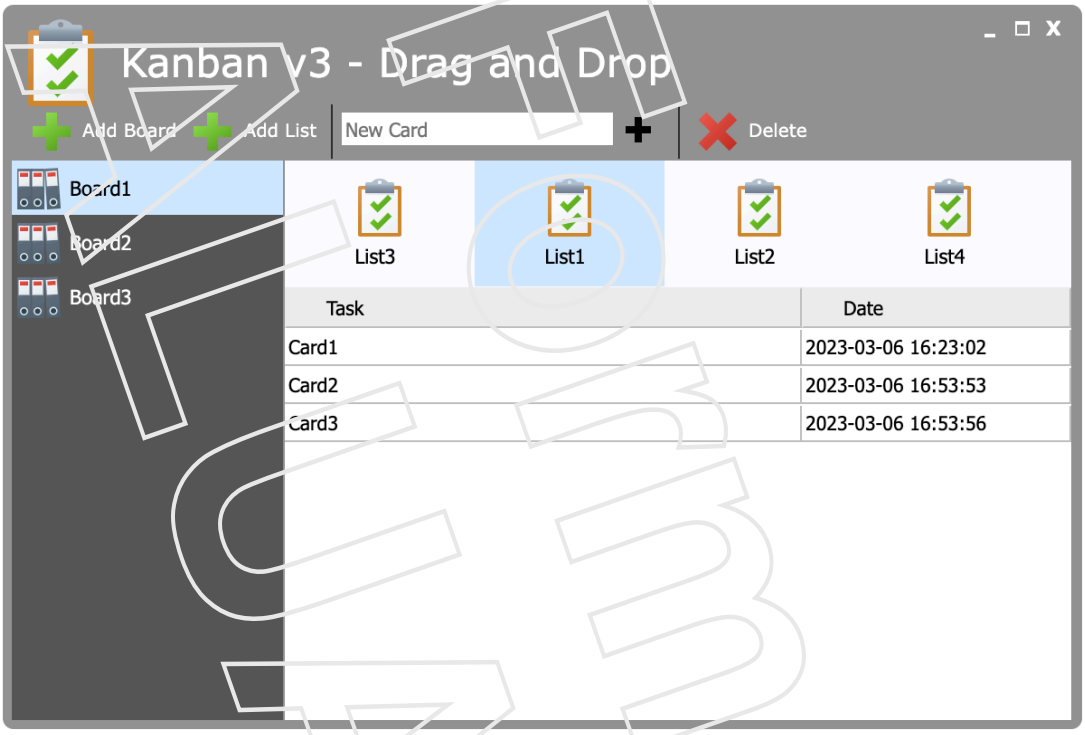
XML documents are already structured data of course, but putting XML documents into hierarchical structures has proven to be very beneficial for many applications. While XML documents are great for collecting individual related data points and processing them as a unit, there are some real-world considerations. The main one is in the security model where it is impractical to make only some parts of a document readable or writeable while still maintaining working XPath expressions, XSLT, etc. Besides access meta-data, other meta-data like timestamps are better managed outside of the document itself. While it isn't always clear-cut where the line should go between XML elements and individual documents, there is also great value in arranging the data in a conceptually different hierarchy despite being able to in practice put everything in a giant XML document.

This illustrates the need to have a data model outside of the XML documents, that brings additional meta-data, security, and possible sharing of XML documents into the picture. Being able to put several XML documents into a container, having sub-containers to classify different XML documents as something else and being able to move documents across containers to denote different meanings to them are all features desirable.

In our development tutorial we have our version of creating a Todo app, a drag-and-drop Kanban board application. For this application we decided to break out the data model into Board containers, and List containers in the CloudBackend database. Each Board represents one kanban view, each List represents the different swim lines on the Board. The XML files stored in the List containers represent the individual tasks. This makes it easy to move a task (card) between two lists, it is a matter of only moving the document to a new container.

Since XIOS/3 is built on the premise of using XML for all data interactions the containers residing in the CloudBackend database are exposed to the developer as XML atom feeds with container elements containing links to other container nodes in a tree representing the data mode of the database. This allows the "Board" list to bind to the node set of Board container nodes, the "List" list to the node set of the container for the selected Board, and finally the swim line to the respective Lists. Add to that a simple form dialog to create new tasks (cards) and a delete action and we have a rudimentary Kanban application in 100-200 lines of code.

Figure 3 The tutorial Kanban application.



The following is the entire code for the applicatio above.

```
<application name="myKanban3" icon="icon://rocket" instances="0"
theme="fabric">
  <view name="Kanban" title="Kanban v3 - Drag and Drop" width="600"
height="400" icon="icon://clipboard_checks" winstate="false">
    <style>
      #boards.iconlist .layout_submenu .text { color: #FFF; }
      #boards.iconlist .marble .listselected .text { color: #000; }
      #lists.iconlist .layout_sectionlist .text { margin-top: 4px;
font-weight: normal; }
    </style>
    <toolbar name="kanbanBar">
      <group name="leftBar">
        <buttonbox name="addBoard" text="Add Board" icon="icon://
plus"/>
        <buttonbox name="addList" text="Add List" icon="icon://plus"/>
      </group>
      <group name="cardBar">
        <input name="addCard" placeholder="New Card" icon="icon://
plus"/>
```

```
      </group>
      <group name="rightBar" align="right">
        <button name="deleteCard" text="Delete" icon="icon://
delete"/>
      </group>
    </toolbar>
    <panel name="MainPanel" type="column">
      <iconlist name="boards" width="150" height="100%"
layout="submenu" deselect="false" scroll="false" iconsize="24"
style="background-color: #555;">
        <rule match="fs:folder">
          <item text="{@name}" icon="icon://folders??color=fff"/>
        </rule>
      </iconlist>
      <panel type="row" width="100%" bgcolor="#fafafa">
        <iconlist name="lists" width="100%" height="70"
layout="sectionlist" scroll="true" deselect="false" iconsize="32">
          <rule match="fs:folder">
            <item text="{substring-after(@name, ' ')}" icon="icon://
clipboard_checks"/>
          </rule>
        </iconlist>
        <panel name="todoPanel" type="flow" height="100%"
width="100%" bgcolor="#ccc">
          <grid name="cards" height="100%" draggable="true">
            <row match="atom:entry">
              <column name="subjectCol" match="atom:title"
display="substring-before(.,'.xml')" label="Task" filter="true"
width="100%"/>
              <column name="dateCol" match="atom:updated"
display="translate(., 'TZ', ' ')" label="Date" filter="true"
width="140"/>
            </row>
          </grid>
        </panel>
      </panel>
    </panel>
  </view>

  <process name="Kanban - Process">
    <trigger view="Kanban" event="Loaded" step="init"/>
    <trigger view="Kanban" component="boards" event="Select"
step="selectBoard"/>
    <trigger view="Kanban" component="lists" event="Select"
step="selectList"/>
    <trigger view="Kanban" component="addCard" event="Select"
step="addCard"/>
    <trigger view="Kanban" component="addCard" event="Enter"
step="addCard"/>
    <trigger view="Kanban" component="addBoard" event="Select"
step="addBoard"/>
    <trigger view="Kanban" component="addList" event="Select"
step="addList"/>
    <trigger view="Kanban" component="deleteCard" event="Select"
step="deleteCard"/>
    <trigger view="Kanban" component="lists" event="Drop"
```

```
step="dropList"/>
    <trigger view="Kanban" component="boards" event="Drop"
step="dropBoard"/>

    <step id="init">
      <operation name="bind" value="tenant://Kanban">
        <component view="Kanban" name="boards" select="/atom:feed"/>
      </operation>
      <operation name="setSelection" value="">
        <component view="Kanban" name="boards">
          <item select="/atom:feed/fs:folder[1]"/>
        </component>
      </operation>
    </step>
    <step id="selectBoard">
      <operation name="bind" value="{#Kanban#boards#@path}">
        <component view="Kanban" name="lists" select=""/>
      </operation>
      <operation name="setSelection" value="">
        <component view="Kanban" name="lists">
          <item select="/atom:feed/fs:folder[1]"/>
        </component>
      </operation>
    </step>
    <step id="selectList">
      <alias name="testAlias" value="#Kanban#boards"/>
      <operation name="bind" value="{#Kanban#lists#@path}">
        <component view="Kanban" name="cards" select=""/>
      </operation>
    </step>
    <step id="addBoard">
      <operation name="confirm">
        <type value="input"/>
        <message>Please enter the name of the new Kanban board.</
message>
        <modal>false</modal>
        <title>Enter board name</title>
        <ok step="createBoard" text="OK"/>
        <cancel step="cancel" text="Cancel"/>
      </operation>
    </step>
    <step id="addList">
      <operation name="confirm">
        <type value="input"/>
        <message>Please enter the name of the new board list.</
message>
        <modal>false</modal>
        <title>Enter list name</title>
        <ok step="createList" text="OK"/>
        <cancel step="cancel" text="Cancel"/>
      </operation>
    </step>
    <step id="createBoard">
      <alias name="containerName" value="{!}"/>
      <operation name="filesystem" value="tenant://Kanban/">
        <create type="folder" name="{$containerName}"/>
```

```
      </operation>
    </step>
    <step id="createList">
      <alias name="containerName" value="{!}"/>
      <operation name="filesystem" value="{#Kanban#boards#@path}">
        <create type="folder" name="{$containerName}"/>
      </operation>
    </step>
    <step id="addCard">
      <alias name="cardName" value="{!}"/>
      <operation name="filesystem" value="{#Kanban#lists#@path}">
        <create type="file" name="{$cardName}.xml" rename="false">
            <content type="text/xml">
                <card deadline="" completed="false">{$cardName}</card>
            </content>
        </create>
      </operation>
    </step>
    <step id="deleteCard">
      <alias name="url" model="string"
value="{#Kanban#cards#atom:content/@src}"/>
      <operation name="decision" value="$url">
        <when test="'{$url}' != ''">
          <operation name="filesystem" value="$url">
            <delete permanent="true"/>
          </operation>
        </when>
        <otherwise>
          <operation name="confirm">
            <type value="message"/>
            <message>Please select a card first to delete.</message>
            <modal>false</modal>
            <title>No card selected</title>
            <icon>icon://delete</icon>
          </operation>
        </otherwise>
      </operation>
    </step>
    <step id="dropList">
      <alias name="dropSource" value="!"/>
      <alias name="dropSourceDoc" value="!" model="value"/>
      <alias name="dropTargetFolder" value="{#Kanban#lists#@path}"/>
      <operation name="decision" value="$dropSource">
        <!-- Check that we are not dropping a board or list, i.e. a
container -->
        <when test="local-name() != 'folder'" step="handleDrop"/>
      </operation>
    </step>
    <step id="dropBoard">
      <alias name="dropSource" value="!"/>
      <alias name="dropSourceDoc" value="!" model="value"/>
      <alias name="dropTargetFolder" value="{#Kanban#boards#@path}"/>
      <operation name="decision" value="#Kanban#boards">
        <!-- Check that we are dropping a list and not a board into
another board, only accept lists -->
        <when test="not(../fs:folder[@id = '{$dropSource#@id}']) and
```

```
'{local-name($dropSource)}' = 'folder'" step="handleDrop"/>
      </operation>
    </step>
    <step id="handleDrop">
      <operation name="decision">
        <!-- Check that what we are dragging is not the same as where ↩
we are dropping -->
        <when test="'{$dropSourceDoc}' != '{$dropTargetFolder}' and   ↩
'{$dropSource#@path}' != '{$dropTargetFolder}'">
          <operation name="filesystem" value="$dropSource">
            <move to="{$dropTargetFolder}" silent="false"             ↩
progress="true"/>
          </operation>
        </when>
      </operation>
    </step>
  </process>
</application>
```

9. Contacts - Key/values, Meta-data, and Datatypes

The CloudBackend database design pattern of having data objects in a searchable, hierarchical containers, which then contain objects that can have zero to many streams attached like an XML document, JSON, text, or binary, allows for a flexible data model.

Examples of what containers can represent could be; invoices, calendar, contacts, time reporting, conference room booking, recipe database, truck journal, and so on. Each stored object would then have an XML document and an XML schema that represent its structure.

To enable rapid searches, datatypes can be defined that automatically extract information using XPath expressions from XML documents as soon as they are stored in the meta-data of the object, which is stored as key/value pairs in the data model. This allows both searching and the listing of a container to result in an ATOM feed that contains all data needed to present the list using XIOS/3 built-in UI components or XSLT.

The ATOM feed structure can be bound directly to the UI components or to a component that applies an XSL transformation to present the content as HTML. Any changes to the data model of the ATOM feed will then automatically update the user interface, meaning that if a new XML document is added, deleted, or updated in the container, it will immediately be reflected in the user interface.

As an example, the contact manager application uses an XML version of the vCard format to store contacts in individual XML documents. The datatype declaration for these vCard files has the following meta-data extraction (indexing) instructions.

```
<index>
    <dc:firstname xpath="/vcard/n/given"/>
    <dc:lastname xpath="/vcard/n/family"/>
    <dc:nickname xpath="/vcard/nickname"/>
    <dc:homephone xpath="/vcard/tel/home"/>
    <dc:mobilephone xpath="/vcard/tel/cell"/>
    <dc:homecountry xpath="/vcard/adr/home/ctry"/>
    <dc:homemail xpath="/vcard/email/home"/>
    <dc:homestreetaddress xpath="/vcard/adr/home/street"/>
    <dc:homezipcode xpath="/vcard/adr/home/pcode"/>
    <dc:homecity xpath="/vcard/adr/home/city"/>
```

```
    <dc:businesscompanyname xpath="/vcard/org/orgname"/>
    <dc:businessdepartment xpath="/vcard/org/orgunit"/>
    <dc:businessphone xpath="/vcard/tel/work"/>
    <dc:businessfax xpath="/vcard/tel/fax/work"/>
    <dc:businessjobtitle xpath="/vcard/title"/>
    <dc:businessmail xpath="/vcard/email/work"/>
    <dc:linkedin xpath="/vcard/linkedin"/>
    <dc:facebook xpath="/vcard/facebook"/>
    <dc:twitter xpath="/vcard/twitter"/>
    <dc:card xpath="/vcard/@card"/>
    <dc:workmobile xpath="/vcard/tel/cellwork"/>
    <dc:businessindustry xpath="/vcard/org/orgindustry"/>
    <ni:photo xpath="/vcard/photo"/>
  </index>
```

Whenever a document is updated or uploaded these XPath expressions are applied to the data model in CloudBackend and the extracted properties are stored in the search index as key/value pairs. Just the ATOM feed itself is then enough to present a usable overview of the objects in the container collection.

```
<atom:feed xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:os="http://a9.com/-/spec/opensearch/1.1/" xmlns:dc="http://
xcerion.com/directory.xsd" xmlns:ni="http://xcerion.com/noindex.xsd"
dc:folder="562958574623716" xmlns:fs="http://xcerion.com/folders.xsd">
  <os:totalResults>1</os:totalResults>
  <os:startIndex>0</os:startIndex>
  <os:itemsPerPage>100</os:itemsPerPage>
  <atom:entry>
    <atom:title>Karl Hyltberg.xml</atom:title>
    <atom:published>2021-03-31T09:10:45Z</atom:published>
    <atom:updated>2023-05-29T11:06:39Z</atom:updated>
    <atom:link rel="alternate" type="text/xml" href="https:/
api.cloudbackend.com/v1/documents/562958574623716/4536952228/1"
length="1522" stream_1_length="1522"/>
    <atom:id>mid:10e6c65a4@xios.xcerion.com</atom:id>
    <dc:folder>562958574623716</dc:folder>
    <dc:document>4536952228</dc:document>
    <dc:root xmlns:dc="http://xcerion.com/directory.xsd">vcard</
dc:root>
    <dc:firstname xmlns:dc="http://xcerion.com/directory.xsd">Karl</
dc:firstname>
    <dc:lastname xmlns:dc="http://xcerion.com/
directory.xsd">Hyltberg</dc:lastname>
    <dc:businessmail xmlns:dc="http://xcerion.com/
directory.xsd">karl.hyltberg@xios3.com</dc:businessmail>
  </atom:entry>
</atom:feed>
```

Besides searching and filtering in data sets another common pattern is merging data sets for presentation. Multiple atom feeds can be combined and presented as a single view, which is extra useful when the different data sets are owned by different identities. Think of this as doing a join operation in a traditional relational database.

One example where this is very useful is in the Calendar application where not only different personal schedules can be combined into a single view, but schedules shared

by other people can be integrated as well. This is done by allowing calendar layers represented as one container for each calendar layer, through the security and sharing capabilities of the data model (using Access Control Lists, identities, and sharing) to be shared across identities.

This means that the developer of a calendar application only have to design the data model with layers as containers, the XML application for a calendar event and then start storing XML documents in containers. If a layer is to be shared, the container for that layer is then simply shared with the identities or groups that should have access.

10. CloudTop - Combining Applications into a Desktop

Stepping outside of the individual applications there is a lot of extra infrastructure needed to make applications run together within the same runtime environment and browser tab. Assuming the actual execution, rendering, and sharing of resources are solved problems, to really unlock the individual application's potential they need to cooperate and be able to open in multiple instances similar to how several emails or Word documents can be opened in a desktop operating system.

There are at least three areas of cross-application integration when we have isolated to a single system: datatype controls of actions, moving of data through user actions, and real-time sharing of resources between running applications and across different clients and identities.

Datatype-based actions are commonly implemented in operating systems as a filetype register, where based on some meta-data property such as filename extension or mime-type property, the system could find the appropriate file icon and associated application to open it. We have already described how the file type-based meta-data extraction of properties works, but the datatype manager can do more. A file type, besides the user presentation data like description and icon, contains rules for how to identify a file type. For XML that is through one or more of namespace, root node, file extension, or mime-type. The definition can also define default applications for actions like open, edit, and preview that the desktop application can call without knowing what application will take over.

That kind of handover is simple as the application is given references to full documents that it is pre-arranged to handle. If however the user decides to drag-and-drop something or copy-and-paste it the application must be able to handle subtrees or fragments of documents. The approach XIOS/3 and CloudBackend has opted for is that all data within the system is handled by reference. Every document has a unique URL. To that we add an XPath to denote the relative root, useful when documents are inlined in other documents, and a set of XPaths that represents a selection within the document. Thus a drop or a paste action in a UI component in one application is just a reference to the same data as in the first application. This have some similarities to the XPointer and XPath standard, but combines a base selection with additional selections relative to the base selection. Think, a file list and selecting a couple of files.

Finally we have the real-time sharing of resources enabling inter application collaboration, but also collaboration between users across multiple clients, but with access to the same XML document in the data model. With the abstracted data model and security model, it becomes straightforward and ties into the architecture of intelligent UI components of XIOS/3. Since all applications are accessing data by reference, the built-in XML transaction manager just looks at all incoming changes and sends out change notifications to applications and any listeners in the cloud, if it happens to observe that a node is subject to change.

All internal bookkeeping and state is of course happening through XML documents, which themselves are subject to inspection. The System Manager application simply binds

the various internal documents, like process list, application list, document cache, and transaction log, to grid UI components that display the content as tables. Things become a lot easier when everything is XML.

Figure 4. The System Manager.