

# Pipelines: Plumbing for the next web

Ian Forrester, BBC Backstage, [pipelines@cubicgarden.com](mailto:pipelines@cubicgarden.com), <http://www.cubicgarden.com/blojsom/blog/pipelines/>

## What is a pipeline?

For the purposes of this paper, a pipeline is a metaphor for a series of tasks or operations that run in sequence. In their most abstract form, pipelines are made up of pipes. These pipes can be tasks, processes, actions, etc. Each pipe has an input, a middle, and an output. The middle definition really depends on the context in which the pipeline exists.

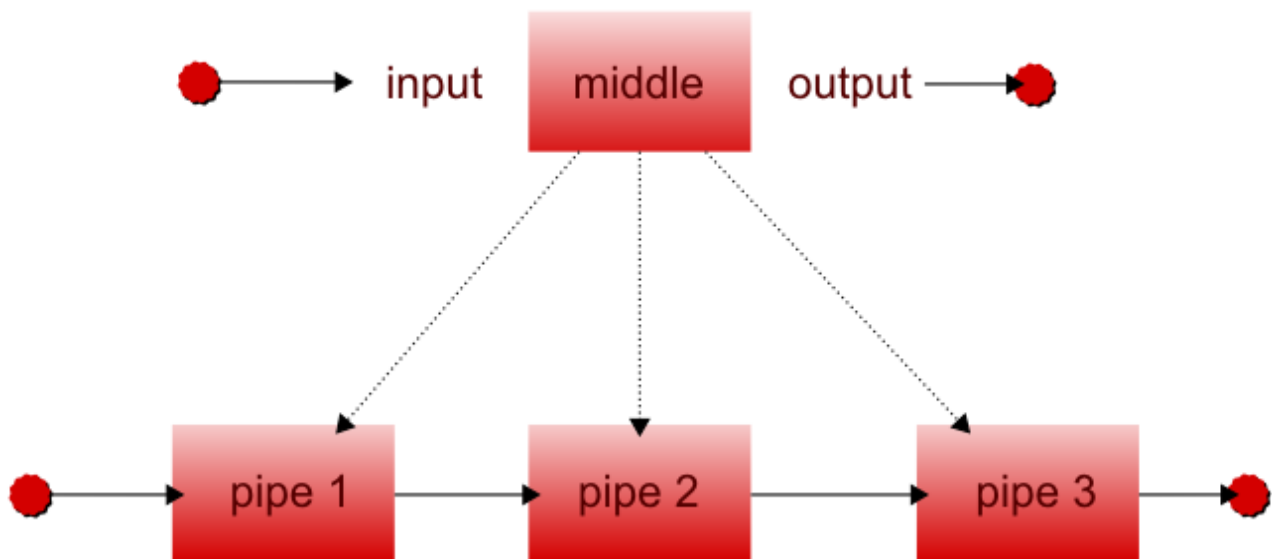
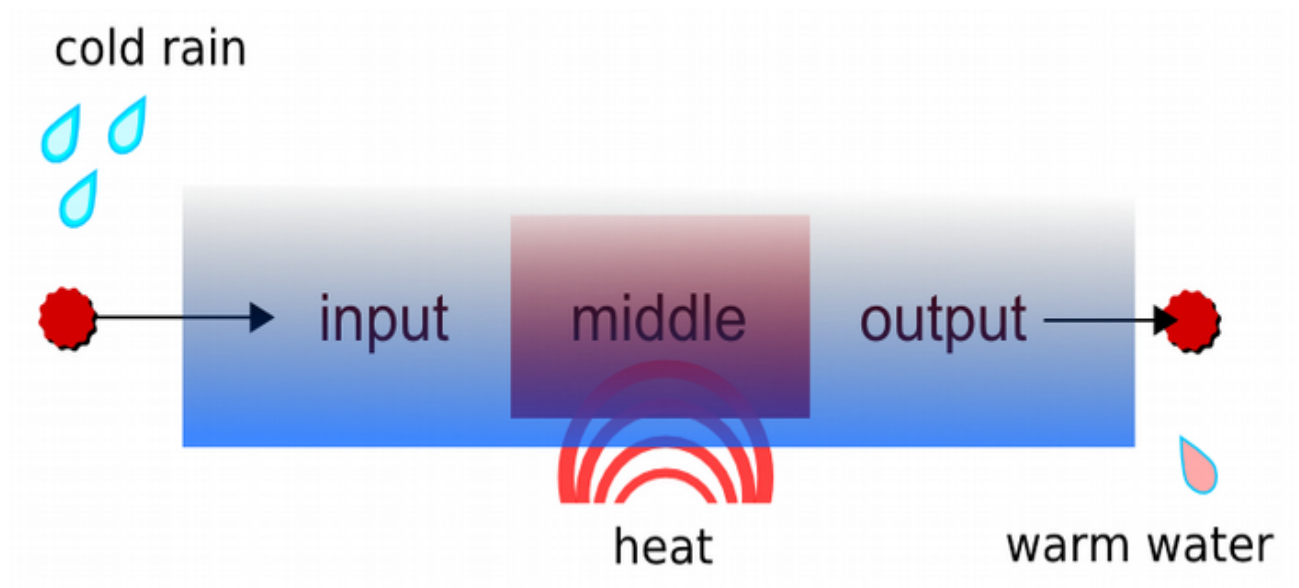


Figure 1. What is a pipe and what is a pipeline?

For the moment, pretend the pipe is a real water pipe. This pipe might simply let water flow from one side to the other. Alternatively, the pipe could have a large funnel system at the input that actually collects fallen rain water, and on the output side there could be a bucket. Rain water would roll down the pipe and drip out of the other end into the bucket for collection. We would then define this pipe with a name and possibly a description; the name could be **rain water pipe** and our description **a pipe that collects rainwater and drops it in a bucket**. This is the most basic pipe-- something with an input and an output that has something flowing through it. The middle always exists, but may not have a function.



*Figure 2. Rain water pipe*

We could add a heated middle section to this **rain water pipe**, so as the rainwater rolls down the pipe it is gently heated. The addition of this feature makes it a different pipe, so we should rename it and change its description. With multiple pipes, it is possible to put them together in a preferred sequence to create new combinations. A series or sequence of pipes should also be named and described. Small pipes are made into small pipelines, which in turn can become part of larger and more complex pipelines.

There are a few rules that make pipelines work:

- Pipes must have an input and an output
- Pipes can not loop on to themselves
- Pipes should always be as simple as possible
- The middle of a pipe can have multiple parts if necessary
- Pipes can have conditions
- The input, middle and output of a pipe can have parameters

So what has all this got to do with the future of the web?

If we change the context of our pipeline to a virtual pipeline (bearing in mind a pipeline is simply a sequence of tasks) we end up with a easily definable tasks that work like building blocks.

## Where did pipelines come from?

### Unix pipelines

The history of pipelines goes back to a system called the PDP-11, but was fully exploited in UNIX operating systems in 1972.

The concept was brought to Unix by Douglas McIlroy who used a similar physical pipeline analogy to bring the concept to Unix. In Unix, each process has a set of inputs and outputs known as stdin and stdout. These are known as standard streams that connect the input to the output.

The introduction of standard streams represented a major change in Unix---abstraction from devices. Before standard streams, the different links between low level devices, such as accessing the hard drive, had to be programmed explicitly. The abstraction allowed programmers to avoid writing low level C code.

From the Wikipedia entry on Unix Pipeline,

## *Pipelines: Plumbing for the next web*

```
curl "http://en.wikipedia.org/wiki/Pipeline_(Unix)" | \
sed 's/[^a-zA-Z ]/ /g' | \
tr 'A-Z' 'a-z\n' | \
grep '[a-z]' | \
sort -u | \
comm -23 - /usr/dict/words
```

- First, **curl** obtains the HTML contents of a web page.
- Second, **sed** removes all characters which are not spaces or letters from the web page's content, replacing them with spaces.
- Third, **tr** changes all of the uppercase letters into lowercase and converts the spaces in the lines of text to newlines (each 'word' is now on a separate line).
- Fourth, **grep** removes lines of whitespace.
- Fifth, **sort** sorts the list of 'words' into alphabetical order, and removes duplicates.
- Finally, **comm** finds which of the words in the list are not in the given dictionary file (in this case, /usr/dict/words).

It is key to remember that each one of these lines or pipes is functional and useful by itself. For example, curl takes a url as an input, grabs the HTML page in the middle, then makes it available as a output. This output is then passed on to the next pipe. It would be possible to change the pipe for example to something like this:

```
curl "http://en.wikipedia.org/wiki/Pipeline_(Unix)" >wikipedia.html
```

This outputs the results to a file called wikipedia.html. It might be more useful to have the output to file as another pipe, so this function could be used separately and dropped into other pipeline combinations.

Unix pipelines are generally easy to follow, but are not particularly self-explanatory. However, there is another huge advantage to using a Unix Pipeline syntax over muddling it into a monolithic program. The implementation of Unix pipelines is highly efficient and has built in buffering, error control, and queuing. For the user, it is quick to write, simple to run, and fairly easy to debug.

From its past in Unix, pipelines have been built into operating systems such as BeOS, Dos, Windows NT, and Apple.

## **Visual Pipelines**

Up until recently, pipelines have been in the domain of Unix and Applescript hackers, mainly as a command line operation. However, graphical user interfaces have also benefited from pipelines.

ROX Desktop provided a icon that could be dragged and dropped to the file system instead of showing the user a "save" dialogue box. The same icon could also be dragged on to another application. In pipeline terms, the input was the raw unsaved data, the middle was a mover or copier, and the output loaded into the second application.

Adobe have also done a similar thing in their series of applications over the years. In Adobe Photoshop 4, Adobe introduced Actions, which were more like macros. They also introduced an application link with other Adobe applications. In practice, this meant one could be working on something in Photoshop and with a click of the button, switch to After Effects with the exact same piece of work. Saving was unnecessary because all of the data was transferred to the new application. Adobe have recently improved upon this application linking with a product called Adobe Production Studio, which is a higher level controller application for Photoshop, After effects, Encore, and Premiere. The interchange of data without saving is not truly a pipeline; it is more like automation and macro behaviour.

## **Macro automation**

The worlds of pipelines and macros overlap quite a bit, and to distinguish them we must define what “macro” means. In this case, a macro is an abstraction that defines how a simple input is recorded as a sequence of actions, which can be replayed again. They tend to work in one application, hence the name “application macro”. Some of the most popular applications using macros are in the Microsoft Office suite.

Using the Macro recorder application in earlier versions of Word and Excel stored the macro in a sub language called Visual Basic for Applications (VBA). This meant users could grasp the language by studying the results of the macro recorder and modifying the results. The power users started to build up a list of useful VBA macros, which they would share with friends and colleagues. Unfortunately, VBA was pretty powerful and had the ability to do most operating system functions. Couple that with automatic execution of macros in Word and Excel files and the whole platform spawned a list of macro viruses, which insured system administrators turned off VBA support in the next Office installation.

There are certain things that separate the worlds of macros and pipelines. The parts of a macro do not function on their own, while a small pipe from a pipeline is still functional. Furthermore, macros tend not to have conditions. Macros are actions with no inputs or outputs. Pipelines are made up of smaller pipes and are definable.

Macros vs. pipelines is best illustrated with an example. A long time ago I used to work as a Photoshop retouching artist, and Photoshop 4 launched with actions (or macros). This meant the usual repetitive task of opening the images, opening the levels, changing the curves, and looking at the levels again before saving could be saved as a macro. In actual fact, it did not help at all and turned out worst pictures imaginable. The original image would have to be reopened and the task started again manually. What was needed was a pipeline setup, where the values could be changed, and those values would affect other pipes down the line. In this example, the first task of opening the levels would inform what would happen in the curves in the next pipe.

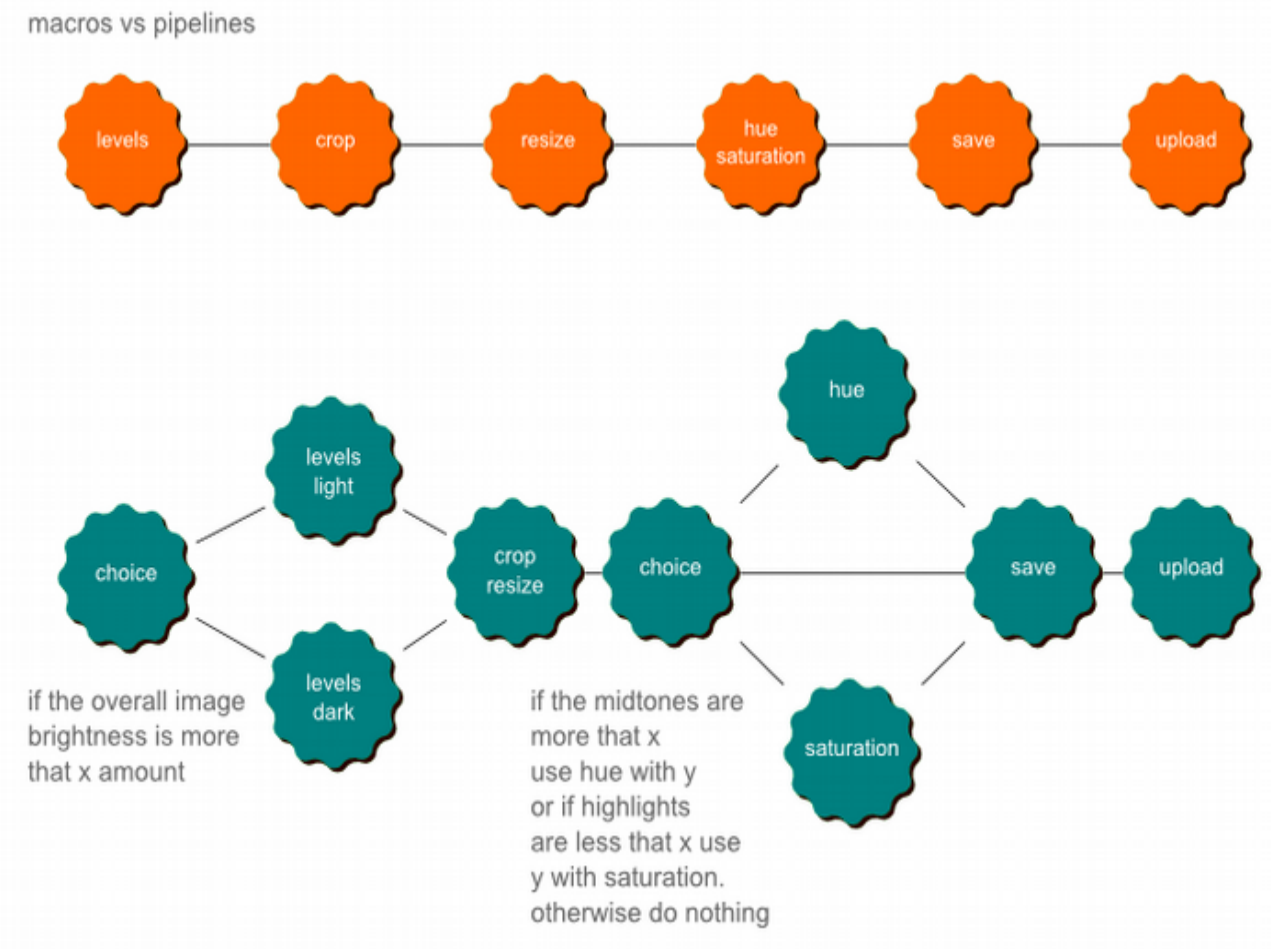


Figure 3. Macros vs Pipelines

Before throwing out the idea of macros, there are some things that can be learned from them:

- The ability to record and modify afterwards is useful for those who just want to automate.
- The ability to share a file is important for adoption.

## Apple Script and Automator

While not a direct ancestor of pipelines, Apple Script and Automator are like a distant cousin to pipelines, in that they embody the concept of “flow”. Apple Script grew out of the hugely popular Hypercard project in the late 1980s, and is basically a scripting language for Mac OS X, in the same way that Visual Basic is a scripting language for Windows.

Apple Script is written with a Natural Language metaphor and generates text files that can be shared around like Word macros. What makes Apple script interesting is 3 things:

- It is highly integrated with OS X, which means literally anything can be done with a good script.
- There is an extraordinary number of scripts and resources online.
- Automator software makes it possible to create Apple Script without learning the language.

From Wikipedia:

Automator is an application developed by Apple for Mac OS X that implements point-and-click (or drag-and-drop) creation of workflows for automating repetitive tasks. Automator enables the repetition of tasks across a wide variety of programs, including the Finder, the Safari web browser, iCal, Address

## *Pipelines: Plumbing for the next web*

Book and others. It can also work with third-party applications such as Microsoft Office or Adobe Photoshop.

Although Automator uses AppleScript and/or Cocoa, it requires no expertise in these languages whatsoever. However, the concept would be familiar to those used to Unix pipelines: the output of the last action becomes the input of the next (though the user can optionally choose for an action to ignore the input from the previous action). Unlike Unix pipelines, however, Automator workflows are sequential: each action is completed before the next action in the workflow begins.

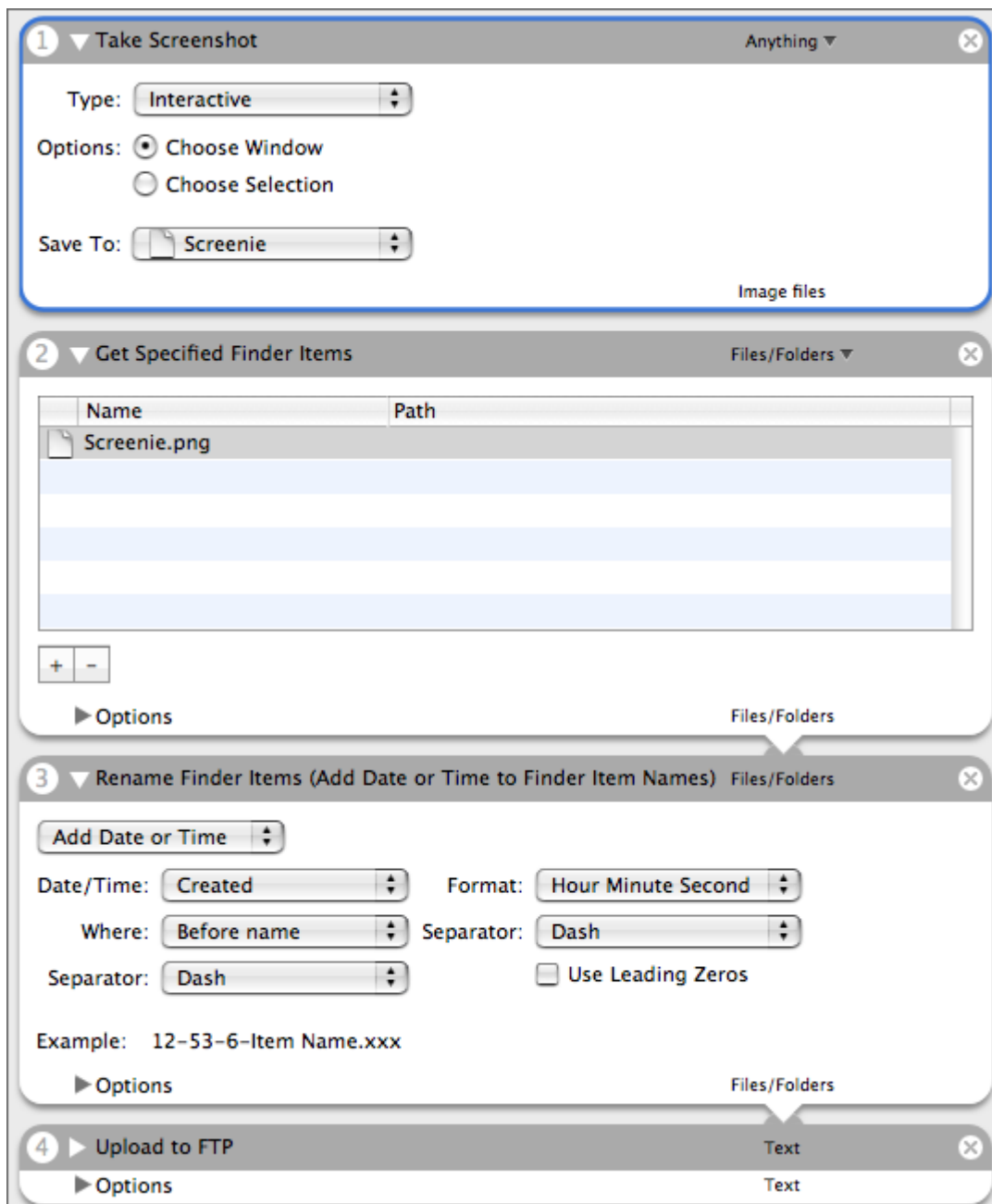


Figure 4. Screenshot of Apple Automator

Cocoatron, the suite of Automator Actions for Mac OS X, brings drag-and-drop simplicity to creation and automation of complex XML Processing Pipelines. It is easy to understand why Automator is so powerful.

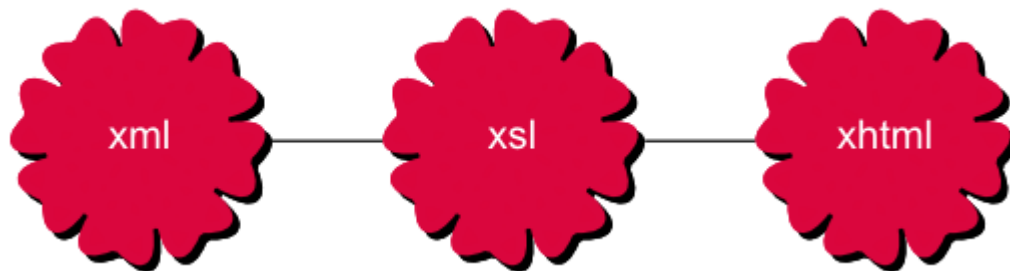
There are Windows alternatives to Automator:

- Script Ahead is aimed at the system administrator. It turns all its scripts into real Visual Basic Script, which is useful, but not cross-platform.
- Automise, like Script Ahead, has the problem of being directed at the system administrator. It also lacks real internet type processes; it can grab an http file, but has no https, proxy, xpath, or xpointer capabilities. One advantage Automise has over Script Ahead is the ability to play with XML files.
- Automate6, also is too focused on system administrators. It does have HTTP(s) get and post, but cannot do anything useful with XML files or streams.

## **XML pipelines**

Traditionally, pipelines have been used in the XML world to deal with the complexity of transformations. XSL 1.0 was only made to do one to one transformations. For example:

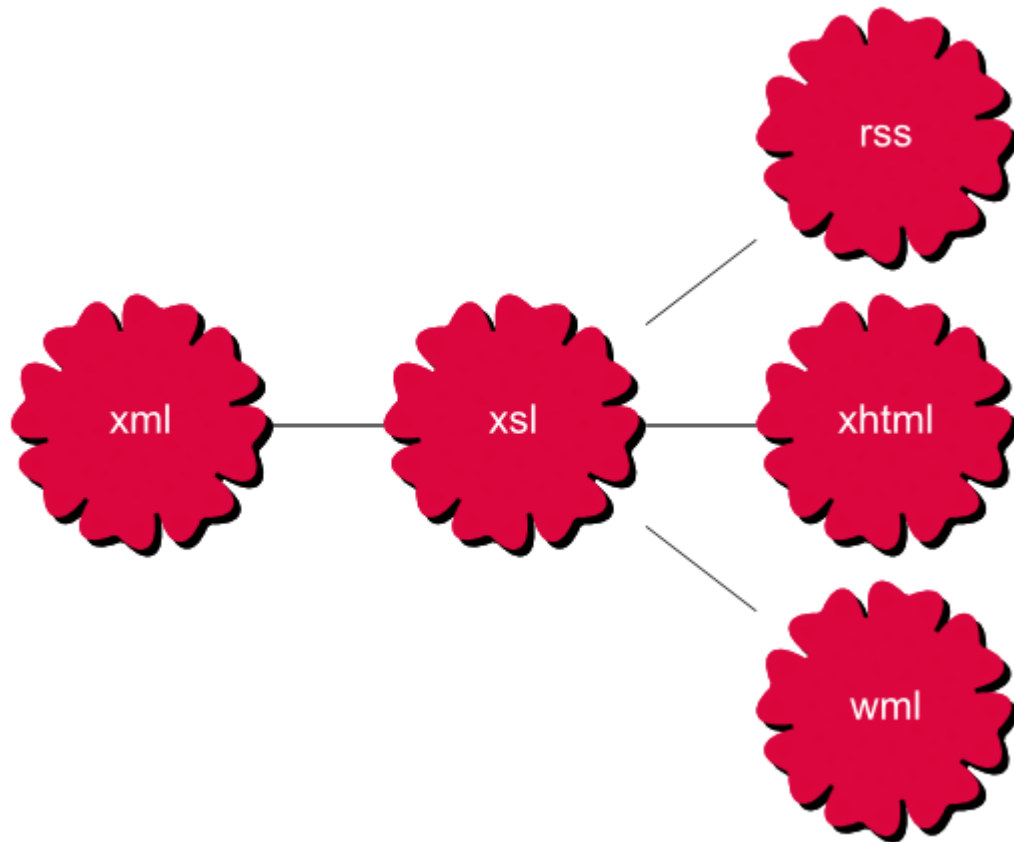
simple xsl transformation



*Figure 5. simple xsl transformation*

If one wanted to create three different formats from that same XML:

multiple output xsl transformation

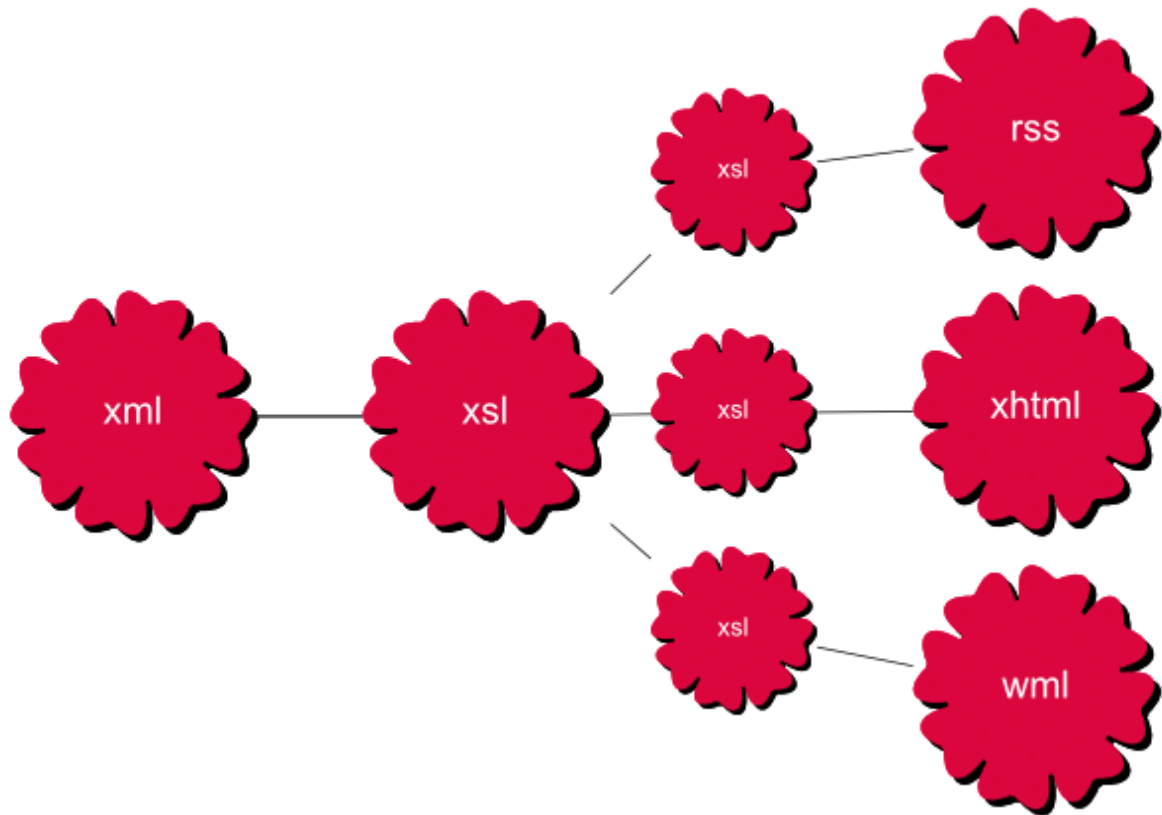


*Figure 6. multiple output xsl transformation*

The above example is possible, but would increase the load of the XSL and add another level of logic to identify which output format was needed. XSL can still just handle this using an XSL import and XSL includes.



multiple output xsl transformation using imported xsl



*Figure 7. multiple output xsl transformation using imported xsl*

For complex document generation, there needs to be an overall framework that manages the whole process; this is where pipelines fit perfectly.

multiple output xsl transformation using a pipeline

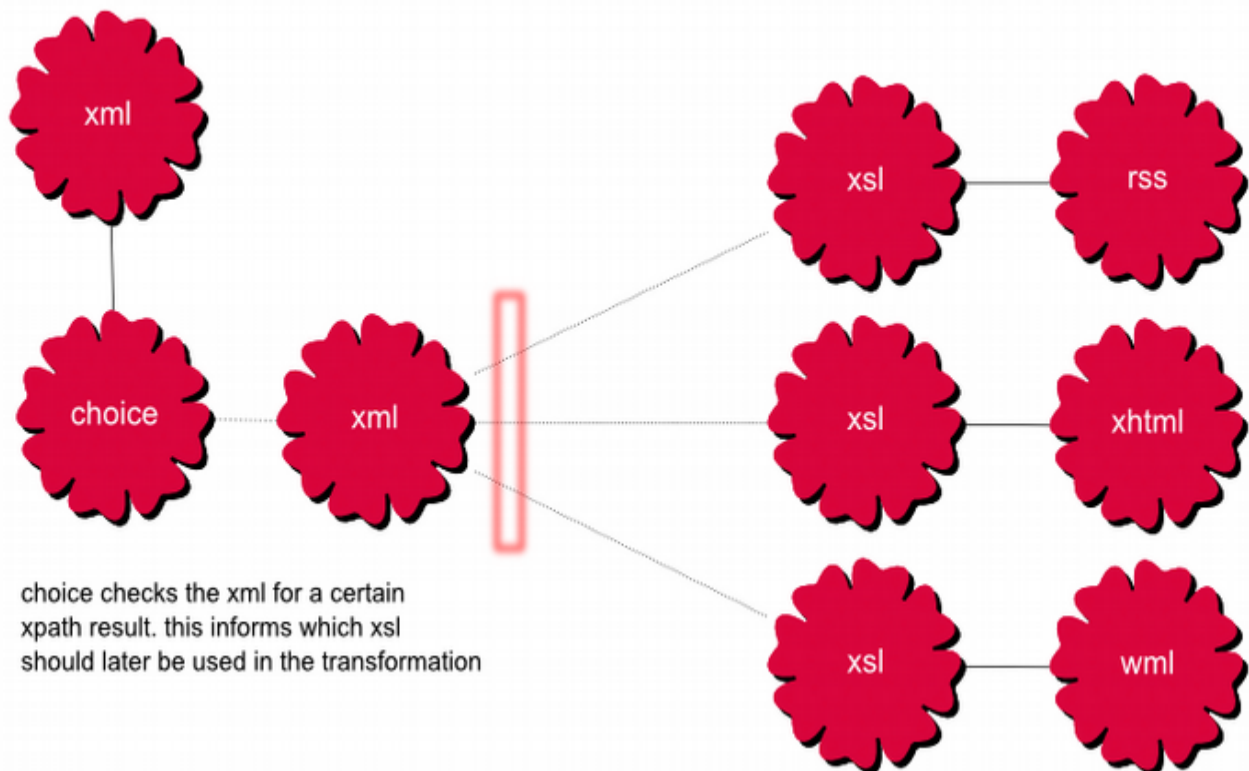


Figure 8. multiple output xsl transformation using a pipeline

Dr Jeni Tennison at Xtech 2005 wrote a fantastic paper about the need for pipelines in XML processing. Her section on reasons to use XML Pipelines has been useful in the past to help explain why pipelines are so important to complex processing.

If you try to carry out complex document-generation such as that described above using a single transformation step, you quickly run into problems. Programs that carry out complex transformations are unwieldy, hard to understand and therefore hard to debug and maintain, simply because they need to do so much.

In situations where the same XML gets transformed to a range of output formats, using separate programs for each transformation leads to more problems. You get repetition of code: whether a bank statement is rendered in HTML, XSL-FO or plain text, the content still has to be filtered and grouped in the same way. You get version-control problems: if the filtering or grouping of the data in the display needs to change, then every program needs to be updated to match.

Pipelining is a technique that helps you break up transformations in a way that maximizes their simplicity (so that they're easy to write and maintain) and reusability (so that the same code is used whenever the same operation needs to take place). In a pipeline, a complex transformation is broken up into several simple component transformations, with the output of earlier components becoming the input to later components. Not only are the individual transformations easier to write, but creating a pipeline helps you to debug the process as a whole by testing the input/output of individual components in the pipeline. What's more, at run-time, the output from one step in the pipeline can be cached and reused, which saves processing time.

## *Pipelines: Plumbing for the next web*

Dr Tennison mentions a few of the frameworks that centre around pipelines, Cocoon being the framework most familiar to myself. She also talks about several experimental pipeline definition languages that do not yet have a framework:

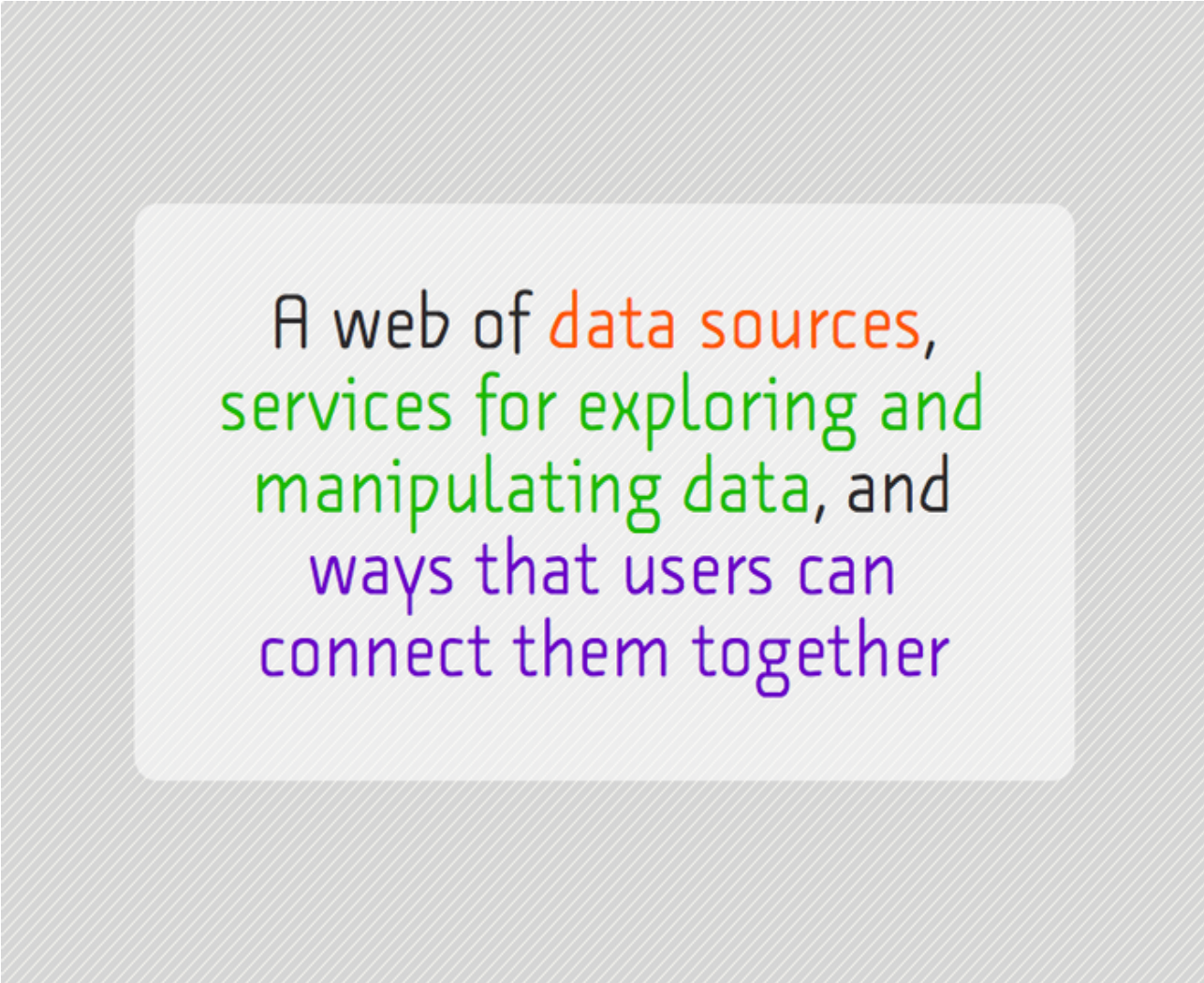
- XML Pipeline Definition Language (XPDL)
- SXPipe
- XPipe
- A Streaming Transformations and Glue framework (STnG)

The most interesting of these was XPDL, which was being passed around the W3C. Currently it is a working draft called Xproc.

### **Web (user-generated) Pipelines**

On the surface, Web pipelines are exactly the same as XML pipelines but applied to a different context. The context is the whole web and the user's desktop and network.

Tom Coates in his presentation – Native to a web of data, talked about the future web looking something like



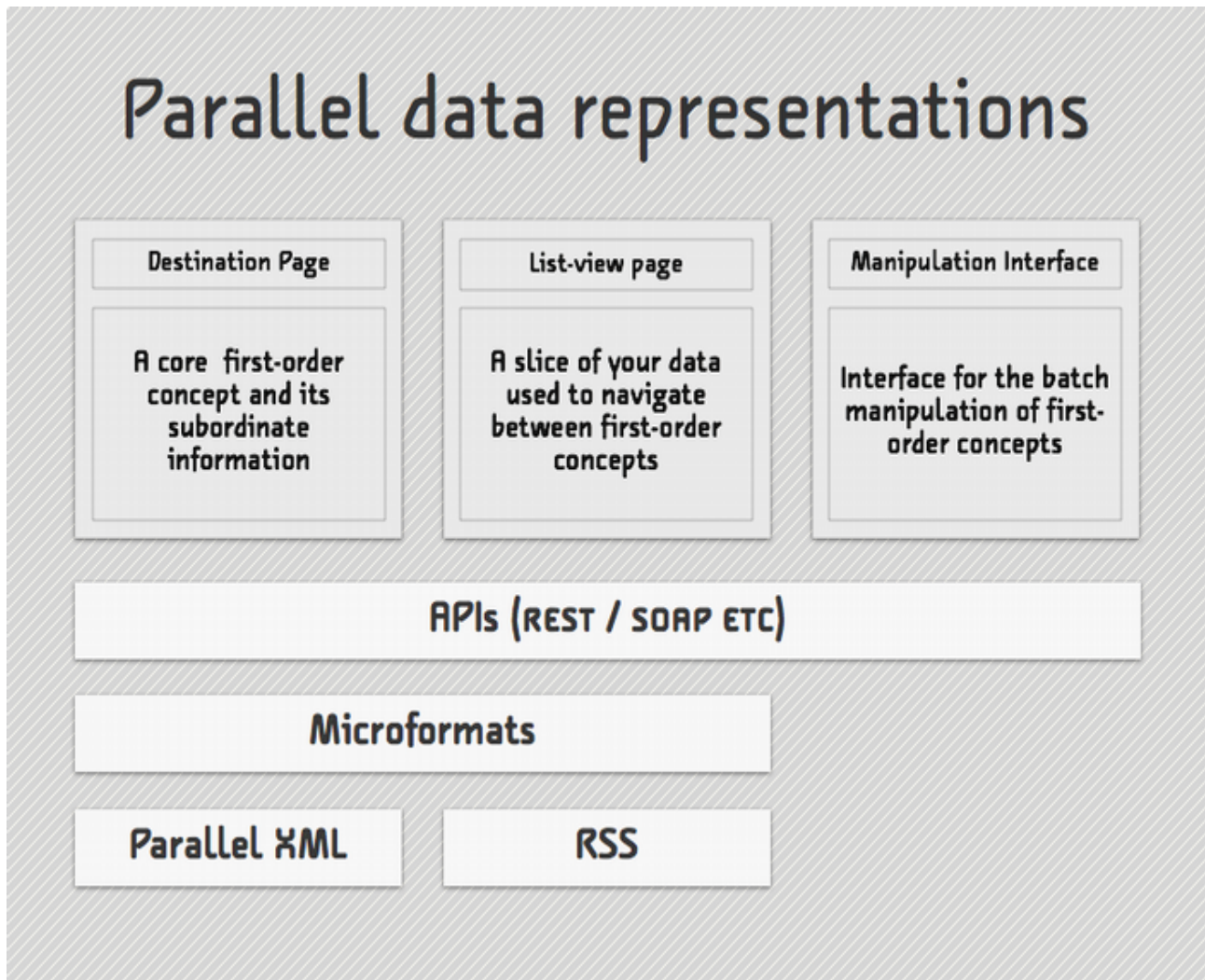
A web of data sources,  
services for exploring and  
manipulating data, and  
ways that users can  
connect them together

*Figure 8. A web of data sources, services for exploring and manipulating data, and ways that users can connect them together*

User generated pipelines is the last part of the puzzle, connecting services together in an automation way

## The Cloud (Internet)

The web is still far from machines by themselves understanding and making reasonable choices based on what is online. However, there are many community projects trying to standardise their own pool of data. They are using simple concepts like tags, permalinks, microformats, guessable urls, xml/rss representation, and restful apis to build a much more semantic web.



*Figure 9. Parallel data representations allow machines to access what humans access*

Tom Coates calls this Parallel data representation. In the end this means the web is becoming much more machine and pipeline addressable.

## Your Space (Desktop)

Locked away on your desktop and network shares is some of the most valuable data to you. Everything from your media collections to emails and contacts. These data pools are very rich and very unique to yourself.

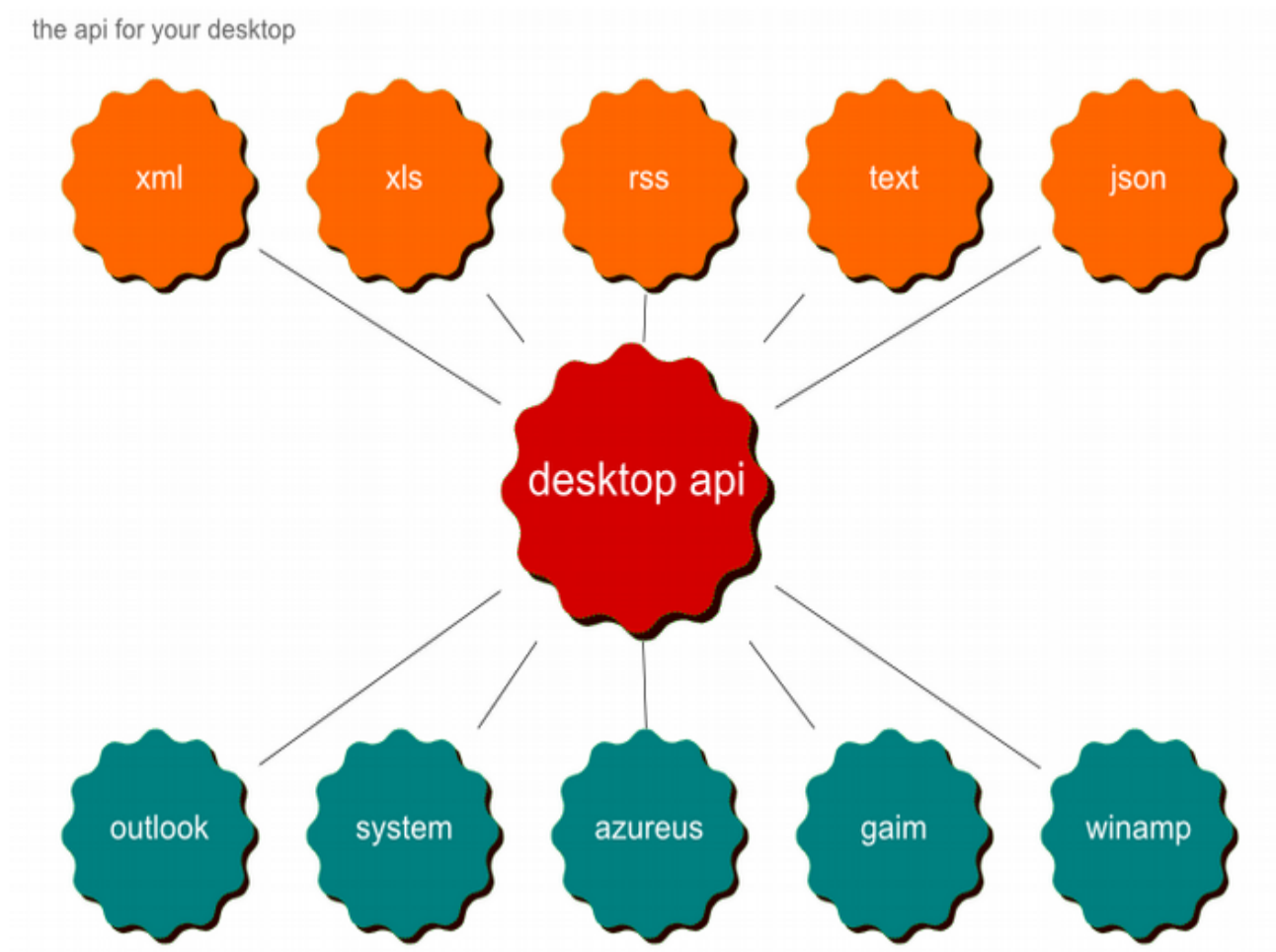
There have traditionally been three ways to get at this rich data:

- Learn C or Java, then hope the developer wrote a good API for the application in question
- Export the data out of the holder program or hope it is stored in XML
- Make the content web accessible

Now there are applications that are turning your space on the desktop and network into a rich store of addressable data, just like the cloud.

The first category is desktop search engines such as Google Desktop, and the second is a new category of interface layers between the file system/applications and the desktop. One example of the latter is called RSS Bus.

RSS Bus has a whole host of features but the most interesting one is its ability to expose core windows services and application data to the world as a addressable local feed. It is like a global desktop restful api, the same kind you can find online for

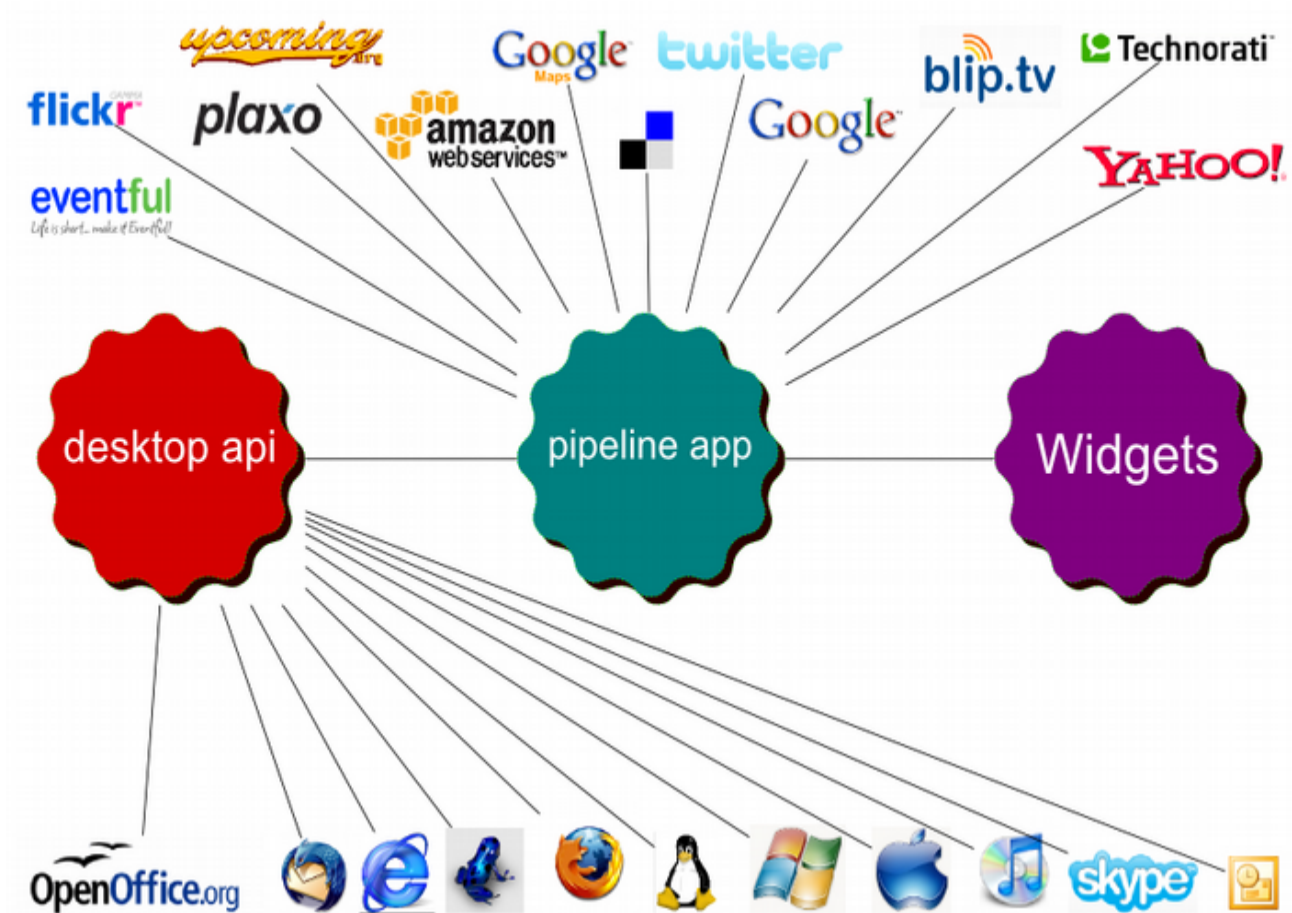


*Figure 10. The global desktop api*

### **Flow (Cloud and Space)**

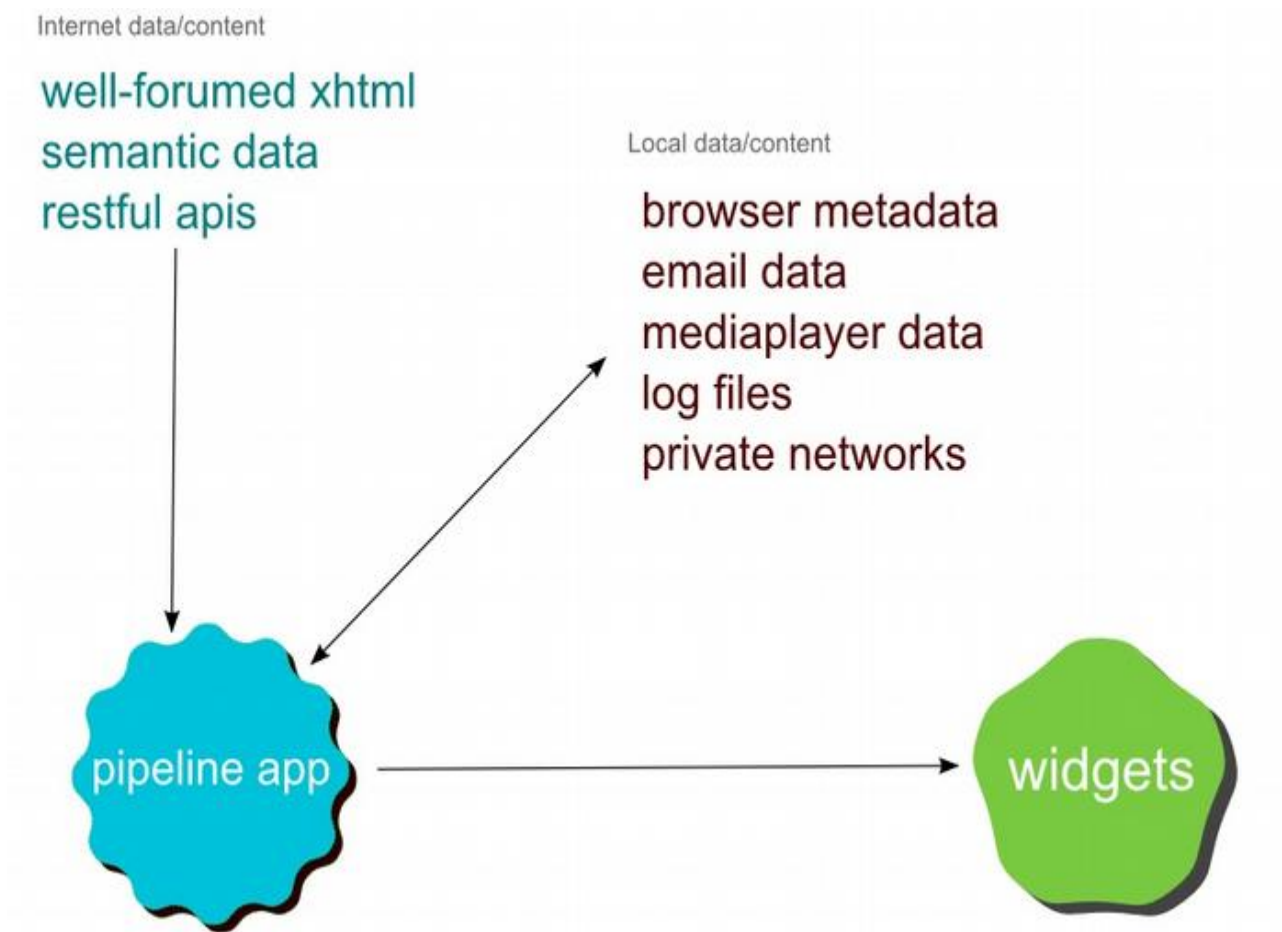
“Flow” is the project name for a theoretical, user-centric, user-generated pipeline framework. It combines the media, feeds, APIs, and data from the web with the data and media on your desktop to give automation across both worlds.





*Figure 11. Flow, when the desktop and cloud combine*

The object of the pipeline framework is to deliver a user experience like the Apple Automator application with the core principles of web pipelines at the heart.



*Figure 12. Using Widgets for the display into pipelines*

In Figure 15, I have split apart the desktop api, pipeline application and widget front end. This isn't necessarily ideal, I have been using RSSBus, Cocoon and Google Widgets to mock up how "flow" would work. In Figure 20, I draw up how Yahoo! could build a "flow" set-up using a different set of components.

In Figure 16, I recommend using widgets to control the actual pipeline application. The reasoning is to make pipelines a solid part of the desktop experience. Windows Vista and Apple OS X now come with some kind of widget engine built in, by placing the pipeline control at the widget level. Users can execute and change parameters without knowing anything about the pipelines structure. This for most people, will be their experience of user generated pipelines.

pipeline tasks as widgets

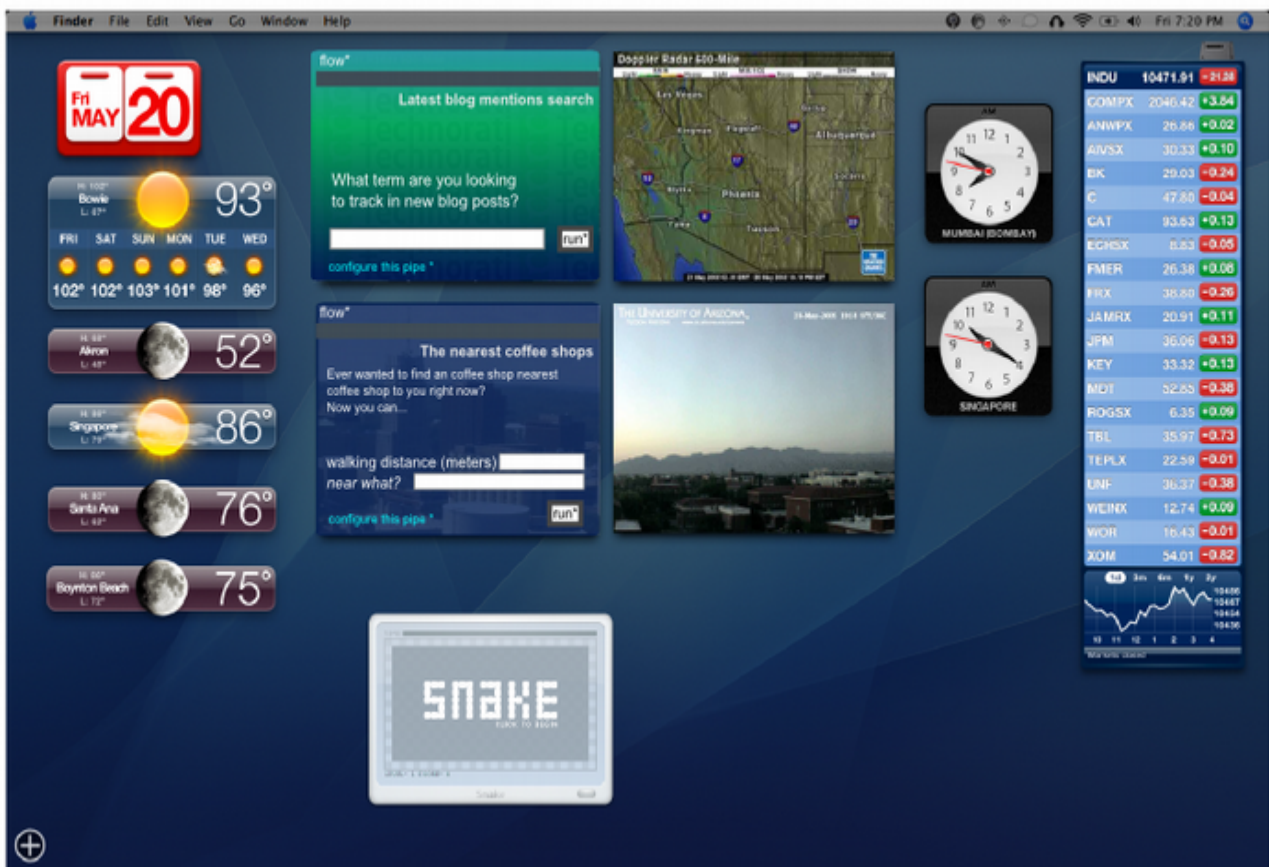


Figure 13. Mock up of Pipeline tasks as widgets in OSX Dashboard

This works in the same way as the Yahoo! Pipes service. At first look the user will never know about the pipeline system underneath. However with time and a little experimentation it will be useful and allow

I expect there to be many “Flow”-like applications coming to the market in 2008.

## Core User-Generated Pipeline Principles

### Graphical and Shareable

The problem with pipelines up until recently is the lack of a graphical interface. Once a pipe or small pipeline is built, it makes sense to graphically represent this for ease of manipulation and to avoid burdening the architect of the pipeline. While some may prefer not to use graphical representation, it should at least be possible using the pipeline language.

Conversely, the graphical interface must be able to produce standard pipeline language XML, which allows the user to modify it with a text editor, post it on the web, and send it around in instant messages or emails.

In addition, another layer should be included which is more details about the actual task. This would naturally be described in XML using the **p:doc element**.



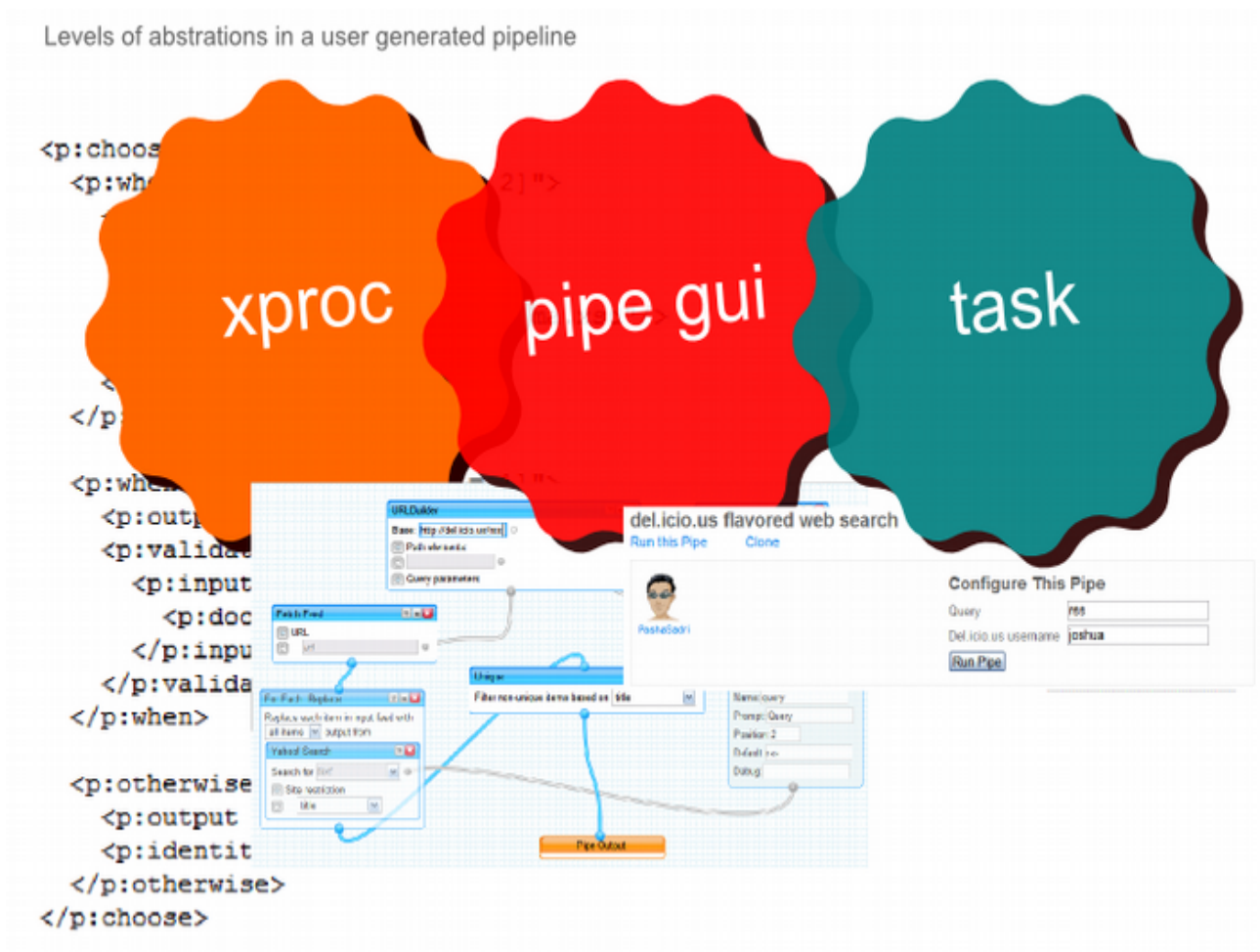


Figure 14. Levels of Abstraction from the XML

## Definable and Standard

XPROC has nearly everything needed to make a good standard for XML pipelines and web pipelines. A W3C recommendation by Winter 2007 is possible if all goes well.

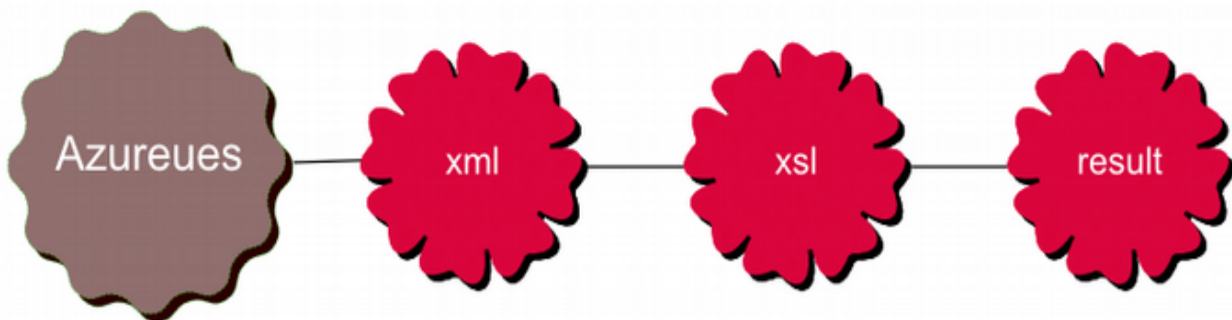
## Open and Non-proprietary

User-Generated pipelines should use an open standard like XML for its core structure. This also means it will be naturally cross-platform. Web pipelines should be clear about their function, and not rely on name spaced extensions.

## Pipeline/flow examples

Xproc pipeline

```
<p:pipeline name="Azureus_view">
  <p:input port="document" href="file:///192.168.0.2/
downloads/Azureus_Stats.xml">
  <p:input port="document" href="file:///192.168.0.2/
downloads/Azureus_Stats.xsl"/>
  <p:output port="result" href="file:///127.0.0.1/E$/temp/>
<p:output port="result" href="ftp://user:pass@127.0.0.1:21/>
</p:pipeline>
```



*Figure 15. Making the Azureus log viewable online*

## Optional Components

One area of not much discussion in the Xproc specification is Optional Components. The components examples in the specification include an XSL formatter, http grabber, Relax NG and XML Schema validators, etc. These are fine for a XML pipeline environment, but the same markup could be used to define how to interact with web services, online services, applications, etc.

Once you have a good store of these optional components, they would just show up in the graphical user interface as another option.

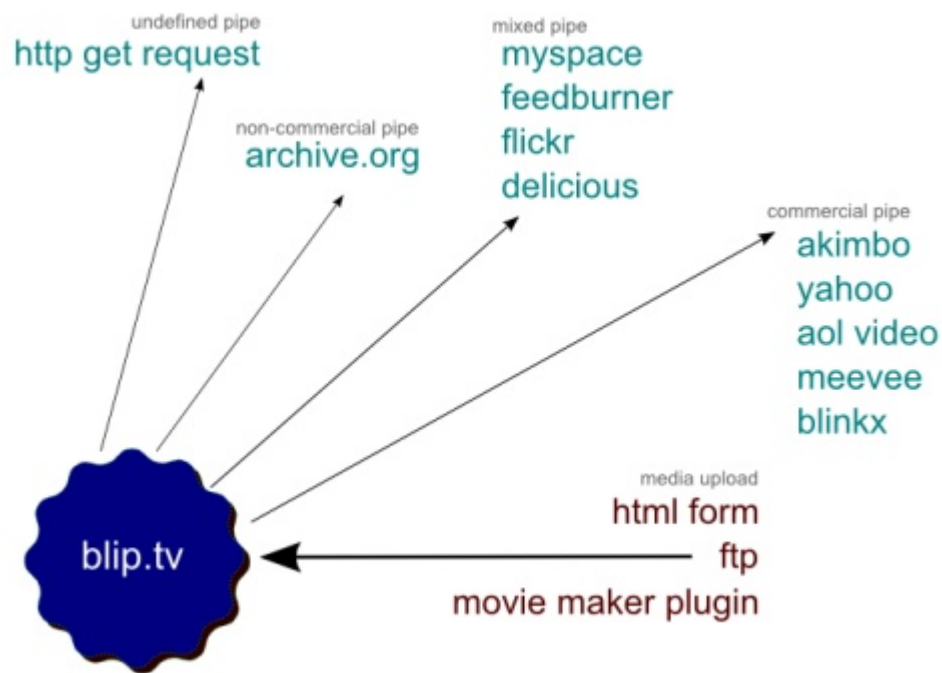


Figure 16. Blip.tv a online video sharing site, exposes all its sharing options as an optional component

Looking at Figure 16, we can see it will accept 3 different inputs and with the correct output parameter, send the results on to another service.

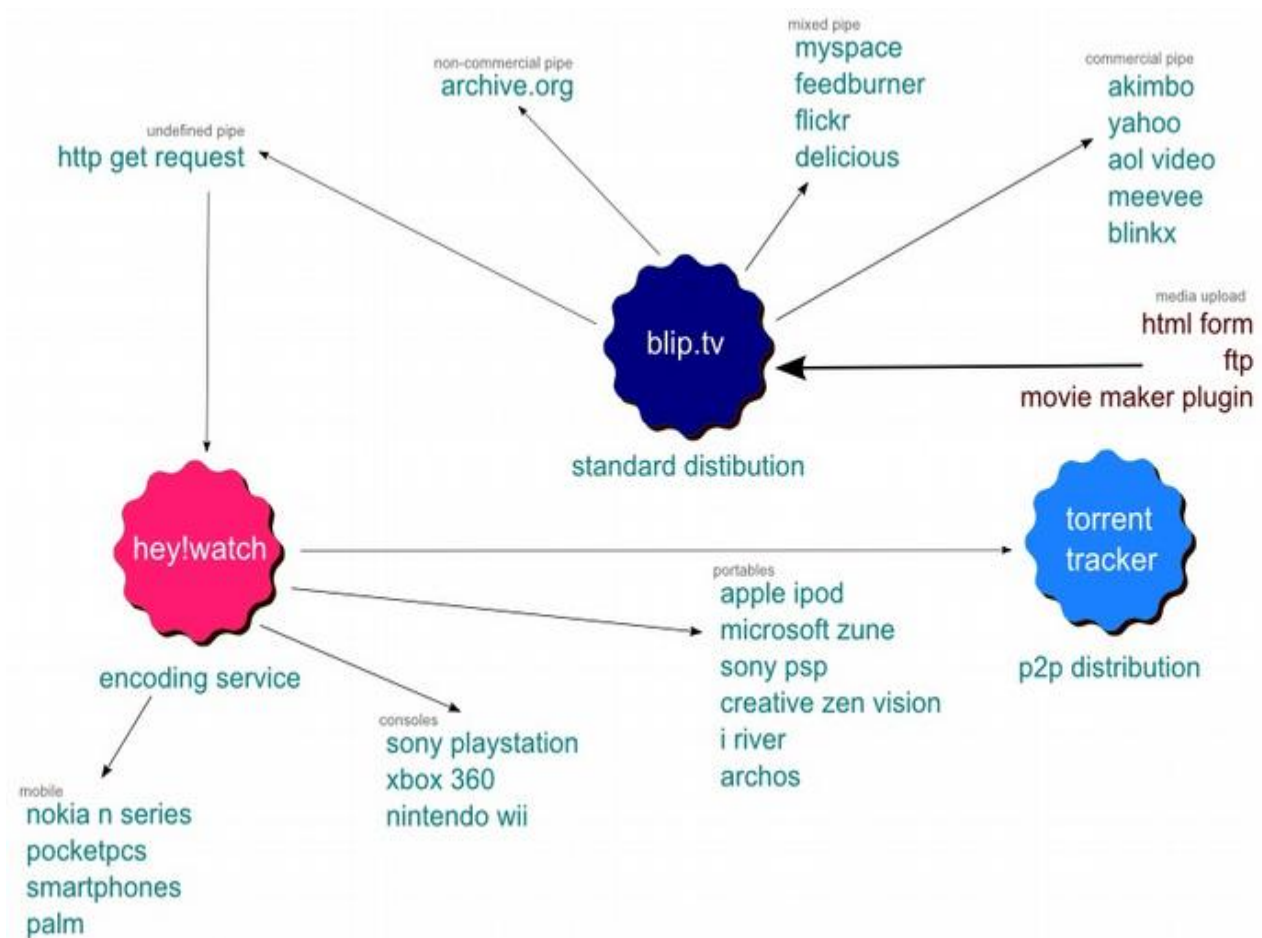


Figure 17. Example of passing content around different services via a pipeline.

Figure 17 looks like a complex pipeline but only because it shows all the different possibilities. For the end user, using there video pipe widget. The options would just be a listed tick box of where the media would end up. This is also not new. Flickr.com for years has allowed the transferring of photos and data to different services and applications. Its comprehensive API means developers of applications can use flickr.com as nothing more that a storage container if the user has paid his/her monthly fee. Amazon as youwill see later are doing the exactly the same. This makes passing data and media from one service to another simple if you can write the code. However using a well written optional component and the right parameters, it will be easy for users to tie the services together for their own gain.

**flickr** GAMMA™

Signed in as cubiogarden (34 new) Help Sign Out

Home You Organize Contacts Groups Explore Search everyone's photos Search

## Third-party applications you're using

[Your Account /](#)

[Check out what's new on the Services page...](#)

You've given permission for the third-party applications listed below to interact with your Flickr account. If you want to stop using one of them, click its "Remove permission" link below.

Application	Permissions	
<a href="#">QOOP</a>	read	<a href="#">Remove permission?</a>
<a href="#">Yahoo! Picture Frame Widget</a>	write	<a href="#">Remove permission?</a>
<a href="#">Flickr Uploadr (Windows)</a>	write	<a href="#">Remove permission?</a>
<a href="#">moo.com</a>	read	<a href="#">Remove permission?</a>
<a href="#">Preloadr by neximage</a>	write	<a href="#">Remove permission?</a>
<a href="#">ecto for Windows</a>	read	<a href="#">Remove permission?</a>

### About Permissions

External applications can make use of 3 different permission levels:

- **Read**  
You will be able to see all your photos via the application. (This includes your private photos.)
- **Write (and Read)**  
You will be able to see all your photos, upload new photos, and add, edit or delete photo metadata (titles, descriptions, tags, etc.).
- **Delete (and Write, Read)**  
You are able to delete Flickr photos via the application.
- **No permissions**  
If no permissions are requested, the application will only display public photos.

Figure 18.a. Flickr.com allows sharing to other applications and services via its comprehensive API.



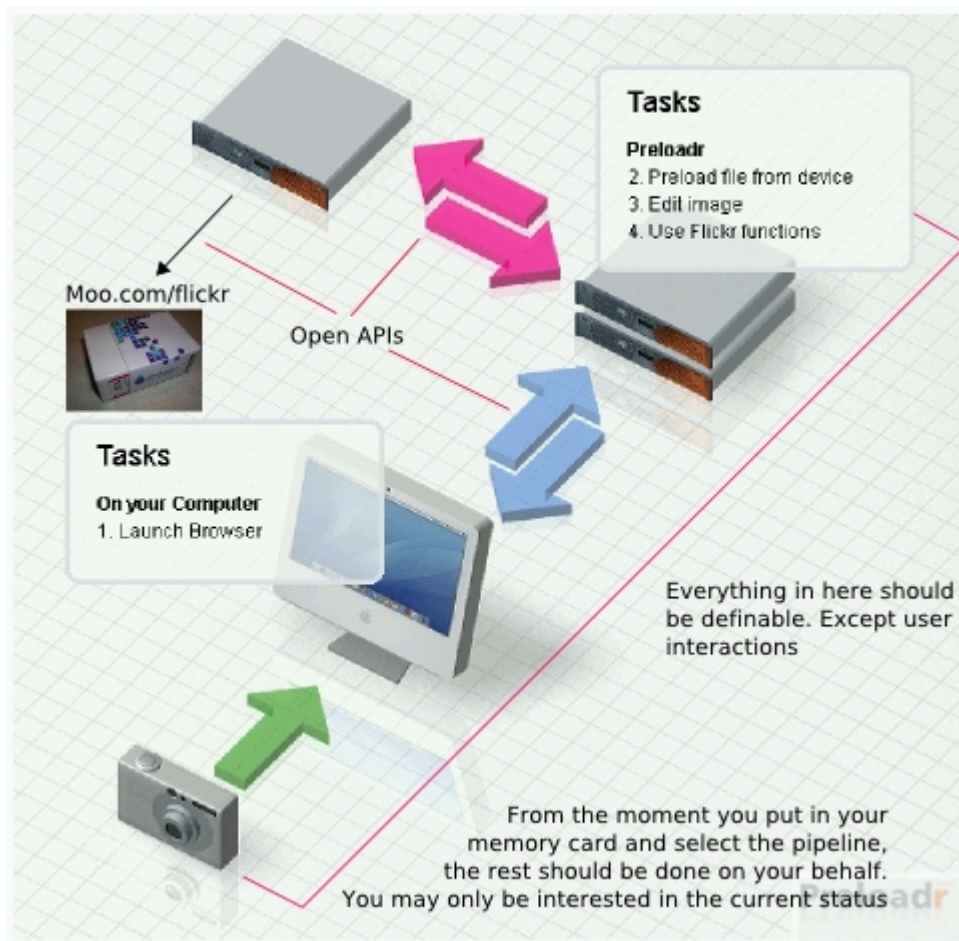


Figure 18.b. My abstract shown using Preloader's example and extending it with Moo.

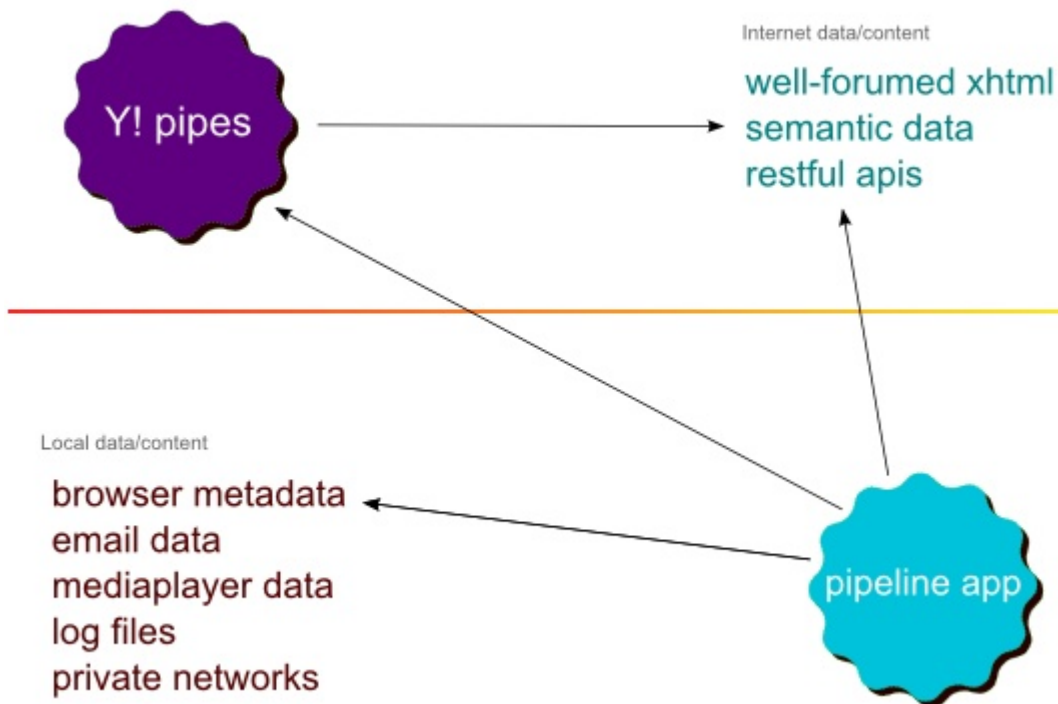
## First with the flow?

In the previous section it was easy to see how using XPROC we could define pipelines outside of the scope of a xml pipeline. But this is only half of the battle. Someone needs to build the framework to support this pipelines.

## Yahoo! Pipes

One of the biggest innovations in pipeline technology recently has been Yahoo! Pipes. Yahoo Pipes is an online service by Yahoo!. It was launched in early 2007 as a beta project.

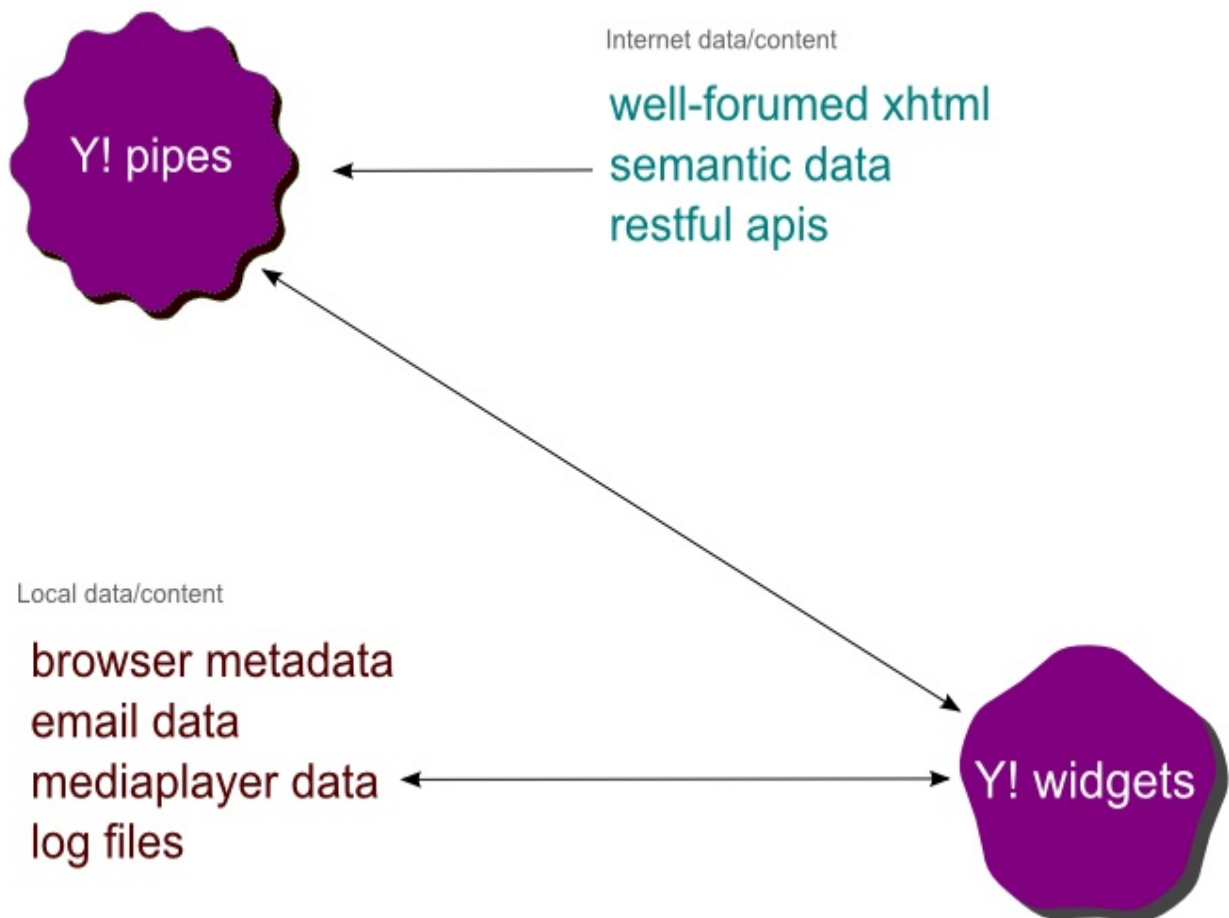
On the surface, Yahoo! Pipes seems to have done everything correctly, but in actual fact what they have built thus far is a very sexy interface to web data mixing tool. This is not too far removed from what Feedburner, RSS mix, Feedcombine, and others have been doing for a while. Yahoo! Pipes, while deserving credit for pushing pipelines into the mainstream, is very limited due to its lack of access to the desktop space and its proprietary nature.



*Figure 19. Why Yahoo! Pipes is very limited in its relevance to the user*

## Yahoo! Desktop Pipes?

While looking for the perfect “Flow” application, I discovered that Yahoo! Widgets would make a great platform for the desktop space. The widget engine already has access to nearly everything on the desktop and it has an authenticated path back to the main server via the Yahoo! system. It also has many deep API hooks into different desktop applications (such as Outlook, iTunes, Winamp) and the operating system. Due to the authenticated path back to Yahoo!, it would be simple to combine Yahoo! Pipes with Yahoo! Widgets to create a “Flow” application.



*Figure 20. How Yahoo! Pipes and Yahoo! Widgets could work together to provide a “flow” type application*

Unfortunately, even if Yahoo! combined Pipes and Widgets in this way, it would remain proprietary and non-standard, at least in the short to medium term. This means that only Yahoo! Developers could build on these two core platforms. XPROC support seems to be a long way away, as Yahoo have created their own pipeline definition. At present, this potential application comes closest to my vision of “flow”, but is still lacking in terms of the core principles of user-generated pipelines.

## The ecosystem of pipelines

Amazon.com recently launched a series of web services beyond their catalogue of shopping goods. No one could quite understand why Amazon would get involved in web services that have nothing to do with their core e-commerce business. Jeff Bezo CEO at Amazon, has received a lot of attention for his plans to run other people's businesses.

The new services Amazon launched are:

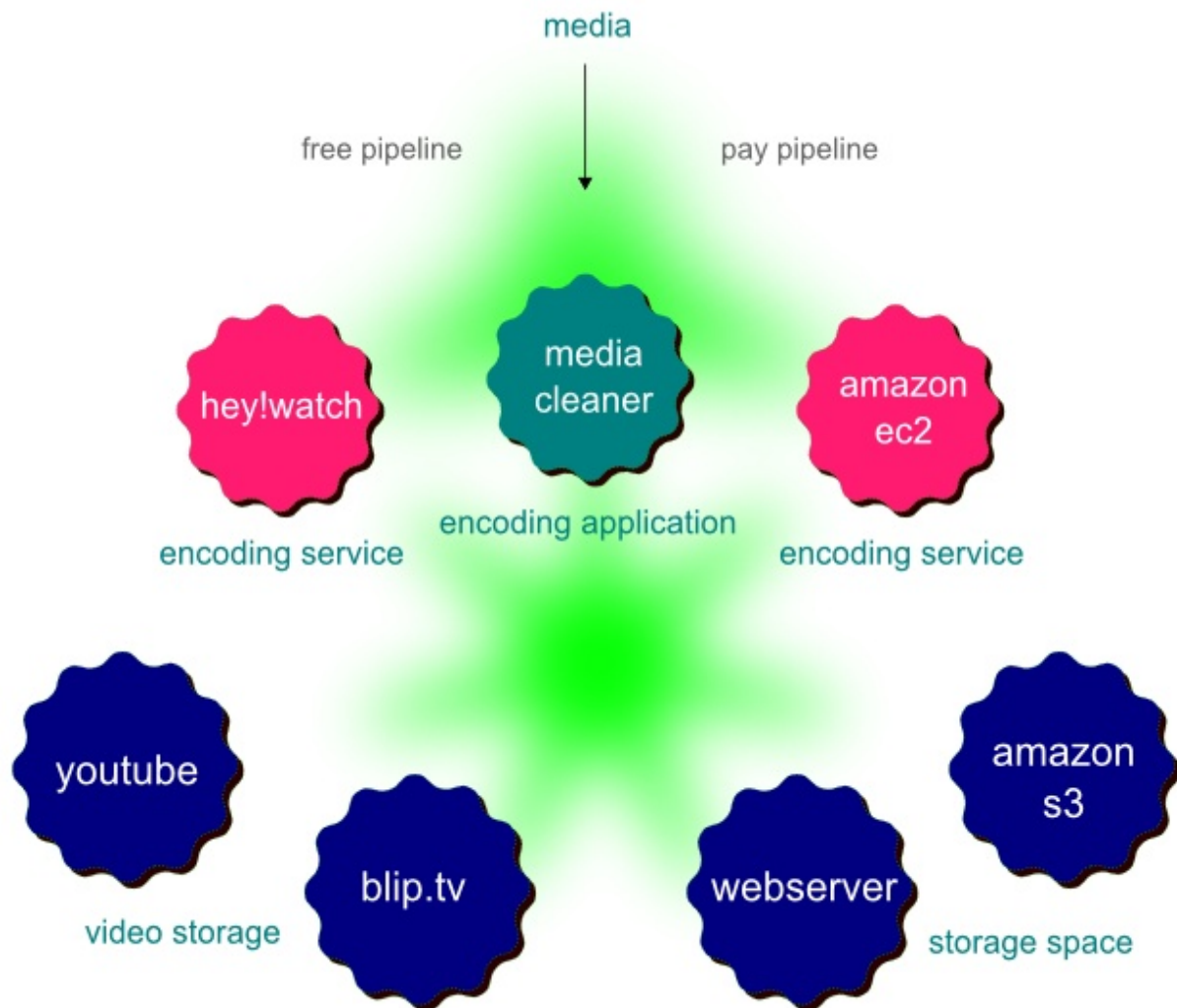
- Amazon Simple Storage Service – S3
  - Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites.



## *Pipelines: Plumbing for the next web*

- Amazon Elastic Compute Cloud - EC2
  - Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Just as Amazon Simple Storage Service (Amazon S3) enables storage in the cloud, Amazon EC2 enables "compute" in the cloud. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.
- Amazon Mechanical Turk (Beta) – Turk
  - Today, humans still significantly outperform the most powerful computers at completing such simple tasks as identifying objects in photographs – something children can do even before they learn to speak. However, when we think of interfaces between human beings and computers, we usually assume that the human being is the one requesting that a task be completed, and the computer is completing the task and providing the results. What if this process were reversed and a computer program could ask a human being to perform a task and return the results? What if it could coordinate many human beings to perform a task? Amazon Mechanical Turk does this, providing a web services API for computers to integrate Artificial Intelligence directly into their processing.
- Amazon Simple Queue Service - SQS
  - Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable hosted queue for storing messages as they travel between computers. By using Amazon SQS, developers can simply move data between distributed application components performing different tasks, without losing messages or requiring each component to be always available.

How do these services relate to Pipelines? Pipelines built by users can give the users choices about which services to use, depending on circumstance.



*Figure 21. User Choice of Service.*

In the example above there are different choices about how to encode footage and store it. Most people would choose to encode it on their local machine with an application like Media Cleaner. However, if the person is using a laptop and does not have the raw processing power or time to spare, they need another way to get their footage out on to the internet. There are free services like hey!watch that will encode footage to almost any format, but it can take more than 24 hours. Soon there will be companies who will use EC2 to build a encoding service, which will mean a very quick turnaround and an SLA (for a price). Storage would work the same way, in that there will be choices between less reliable free services and more reliable pay services.

In relation to pipelines, the execution of a choice is as simple as using one pipe or another. In the encoding example, the choice comes down to wanting it now or wanting it later.

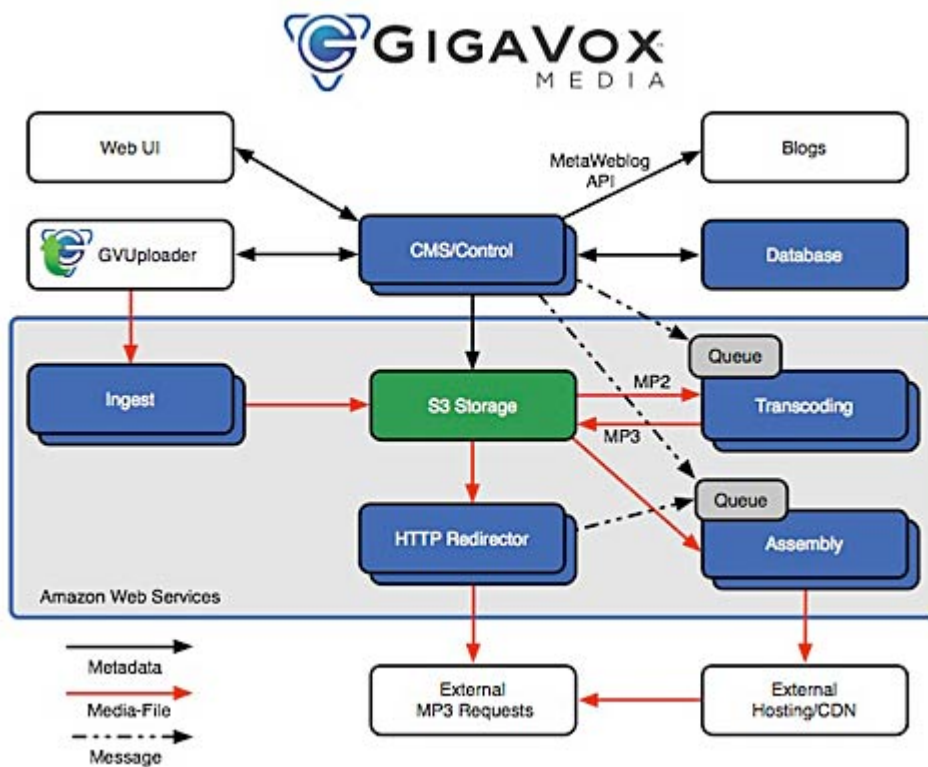


Figure 22. GigaVox media

Amazon's approach is also attracting others who see the power of being able to swap bits in and out such Gigavox media who run the award-winning IT conversations.com. More web services like this will come on to the market. Those who choose not to allow API access to everything the service does will find themselves unable to compete.

## In Conclusion

The trend is moving towards some more control at the user end, on-line services are allow more control via their fat APIs while they sit back and charge users for use. On the desktop, application are also finding piles of user data which have been locked away in large proprietary stores. Everything from Word documents to Email is now becoming addressable and discoverable. Even those chunks of data which refuse to be addressable are being revealed by applications like RSS Bus.

Once all these APIs and feeds are available to the user, its a matter of time before services and application are tied together for the benefit of the user instead of the service involved. In Figure 18, flickr actively allow there services to be chained together via their API because they make their revenue from people storing photos on their servers. Others like Blip.tv are doing the same but why does the user have to wait for the service to open up like the ones mentioned before. Specially now there are services like Mashery and Yahoo Pipes which can consume on-line services on behalf of the user.

The final part of the puzzle is the pipeline. Learning from XML Pipelines and other pipelines before them. It is easy to imagine a future where pipelines are shared like Grease-monkey scripts today.

I have outlined one way this could all fit together, "Flow". There is a lot to be worked out still like Identity management, Single Sign-on, Micro-payments and Licences. However I'm expecting most of these things will fit together in the near future. As for user generated pipelines, they will exist in one form or another. Pipelines have been so useful before and will continue to be useful in the future.

## **Bibliography**

Native to a Web of Data – Tom Coates

The Evolution of the Unix Time-sharing System – *Dennis M. Ritchie* Bell Laboratories, Murray Hill, NJ, 07974

Managing Complex Document Generation through Pipelining – Dr Jeni Tennison, Jeni Tennison Consulting Ltd

Dashboard demonstration – <http://www.flickr.com/photos/evanosherow/14856080/>

Gstreamer opensource multimedia framework – <http://gstreamer.freedesktop.org/features/>

XML in Gstreamer – <http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/chapter-xml.html>