Transcoder Final Report

By: Wilbert Kraan, Mark Power

Contact: Wilbert Kraan (w.g.kraan@ovod.net)

29 April 2010

Table of Contents

2.

1 Acknowledgements

The transcoder project was funded by the JISC, under the auspices of the Learning & Teaching committee, with all software development work by Knowledge Integration, Ltd.

2 Executive Summary

The transcoder project set out to develop and trial a cloud based service that can change one type of educational content package into another (i.e. transcode). Such a capability would help overcome the current proliferation of incompatible content packaging formats, which can be an obstacle to the dissemination and re-use of packaged learning content.

The aim was to both test the technical feasibility of such a transcoder, as well as explore the business case for deploying it as cloud-based Software as a Service (SaaS). In that regard, both the initial set-up phase as well as potential future business models were explored. Possible further developments of the service were also considered.

The transcoder software was developed after a pattern established by a similar, advertising supported service for other kinds of file formats. The pattern is centred around a manual upload and download of packages, in order to facilitate maximal transcoding capacity without bottlenecks. The software itself was developed to be hosted on Amazon web services, for a variety of reasons that include flexibility of deployment, price and sustainability.

In the event, demand for the service turned out to be low, which may have something to do with limited usage of packaged content in the JISC community, exacerbated by limitations of the design of the transcoder's interface, and possibly also the range of conversions that were developed and are currently available. As a consequence, the low start-up investment costs as well as the scalable exploitation costs of cloud computing offered clear advantages for this type of service. Also, at this, or even much higher levels of usage the ongoing costs required to keep the service going are low. Finally, feedback

from stakeholders indicate that a few simple extensions to the software could make the service more useful.

3 Background

3.1 e-Learning Content Packaging specifications and profiles

One of the main strands in the early development of e-learning was the idea of learning objects (Wiley & others 2002). The definition of it was famously variable, but many interpretations centred around the idea of an interchangeable, structured piece of content that was re-usable by allowing itself to be disaggregated and re-aggregated with other learning objects to become a new resource.

For that reason, a lot of development effort went into the creation of a standardised format for these objects, starting with IMS Content Packaging (Anderson & McKell 2001). One thing quickly became clear: different stakeholders had different expectations about what features should be supported by such a packaging format. The initial consequence was that the IMS Content Packaging standard became quite flexible, so that it could support a very wide range of uses, pedagogies and implementation strategies. The disadvantage was that interoperability between different implementations suffered as a result.
To solve that problem, the Advanced Distributed Learning initiative developed the Sharable Content Object Reference Model (SCORM) as the first mainstream Content Packaging profile (Bailey 2005). It succeeded in guaranteeing a much better level of interoperability between implementations, mainly by restricting Content Packaging's flexibility, and thereby the range of uses and pedagogies it could support.

Meanwhile, partially as a consequence of the JISC X4L programme, and partially because of practice established in JISC - CETIS codebashes, an informal plain or 'safe' Content Package profile was established between willing market participants (Kraan 2005). This informal profile was largely identical to the Content Packaging profile defined for the NLN, and was equivalent to SCORM version 1.2, but without some of the single, self-paced learner focussed features of that profile. Major VLE vendors, particularly those with large market shares in the HE market, implemented their own versions of IMS Content Packaging that interoperated very poorly with other profiles or versions of the specification.

In the mid-2000s, then, the situation had coalesced into one where there was a workable level of interoperability in and around SCORM among willing vendors, and a range of uninteroperable packaging formats centred around vendor specific platforms.

Then, two more profiles were developed: a later version of SCORM called 2004 that included a sophisticated but difficult to implement sequencing feature, and, somewhat later, a partial copy of Content Packaging by IMS itself. That copy, Common Cartridge, was meant to unify all Content Package versions that were not SCORM by both supporting a few features that were important in HE, FE and schools, and by deliberately breaking compatibility with tools that support versions of Content Packaging.

The result was that the market was divided between at least six different Content Packaging profiles that were not at all or only partially compatible:

- WebCT implementation of CP
- Blackboard implementation of CP
- 'Safe' standard IMS Content Packages
- SCORM 1.2
- SCORM 2004
- IMS Common Cartridge

In addition, new versions of Common Cartridge were on the horizon, both as general improvements and to accommodate specific communities. Some of those communities were busy designing their own independent profiles as well.

Clearly, something needed to be able to abstract over this variety in order to give users a chance of finding packages that would actually work in whatever VLEs their institution had. This particularly true of the packages held by the Jorum repositories.

3.2 Cloud computing

At the same time, Cloud Computing emerged as a new paradigm that promises to revolutionise the provision of computing. By concentrating compute cycles and storage in very large, dedicated datacentres rather than on piecemeal, dedicated servers at user

organisations, very large economies of scale can be offered by the cloud vendors on the infrastructure level.

In our estimation at the beginning of the project, these economies of scale could enable two different developments at the level of Software as a Service: one is the outsourcing of the most commoditised and mature IT functions such as file storage and email to the cloud, the other is a wave of innovative services that would not be viable on the scale of an individual organisation, but which could work on a web scale. The transcoder was designed to be a pilot of this type of niche, cloud-based service.

From conversations with stakeholders, we knew that Content Packaging conversion could be valuable to some, but it seemed unlikely that many would find the capability valuable enough to justify risking investment in the necessary software development, dedicated computing resources and manpower. For that audience, a Software as a Service solution would take away that risk and need for investment.

At the same time, the project – as service providers – had no idea what level of usage to expect from the community. While a given amount of resource had to go into development, the investment required for a conventional server solution varied between not justified at all (because the level of usage wouldn't justify the cost of the server) to clusters of servers costing tens of thousands of pounds. Infrastructure and Platform as a Service solutions can scale up and down with demand and as far as resources allow. In other words, the same reasons that make risk manageable for SaaS end users make risks manageable for SaaS providers too.

Also, cloud based Software as a Service lends itself quite readily to a variety of sustainable models. After a pilot phase, ongoing costs become predictable, and the value of the service well established. Depending on the balance between costs and value for the various stakeholders involved (JISC, JISC - CETIS, individual H/FE institutions, content or VLE vendors, other sector representatives such as Becta), a number of different business models can be established:

- If the value (and therefore cost) of running the transcoder proves to be low, one stakeholder can elect to keep it going for a while as a service for what is effectively a minute budget.

6.

- If the value (and therefore cost) of running the transcoder proves to be moderate, an alliance of stakeholders (most effectively sector representatives) can agree to share the cost on an annual basis- either as a fixed percentage, or by relative usage, as evident from the service logs.
- If the value (and therefore cost) of running the transcoder proves to be high, it may be worth developing a 'pay wall' or even advertising to effectively let end-users pay for the use directly or indirectly.

Finally, though the transcoder is a very specific service for a very specific domain, the cloud hosted SaaS model could be a pilot for other specialised services that have potential value to the community as a whole, but unknown levels of usage.

4 The Cloud computing business case

4.1 Aims and Objectives

The main objective behind this element of the project was to investigate the proposition that cloud computing/storage, specifically that provided by Amazon, is a viable approach to providing shared services with low or uncertain demand.

One of the difficulties with pilot services is the high start-up cost of infrastructure provision, either using institutional servers or data centre provision, and the ability to quickly and cheaply start up and scale as necessary. This typically undermines the overall business case for the provision of niche services.

For these reasons the project decided that for this pilot a more agile solution was used, trialling the use of outsourced infrastructure web services from Amazon.

4.2 Methodology

Amazon offers very reasonably priced pay-per-use services for storage and processing capacity that scale with usage.

These services, generally referred to as a "cloud computing platform" enable:

- flexible service scaling,
- high reliability,
- and low entry costs.

Cloud computing builds on established trends for driving the cost out of the delivery of services while increasing the speed and agility with which services are deployed.

JISC projects developing software tools and services are similar to commercial startups with regard to sustaining infrastructures of uncertain size over time, but neither having the time or money to research the merits of cloud computing.

As a project looking to provide a pilot service we were looking for two things: Being able to quickly deploy the transcoder service to the community and keeping costs to a minimum, especially when a key element to the project was in actually ascertaining demand for such a service. Trialling the use of the cloud firstly allowed speedy deployment by eliminating procurement cycles and institutional policy for hardware and software, by outsourcing various management and security functions to Amazon, and the automation of scaling up and down resources as needed. The financial benefits were achieved by not assuming all of the costs that come with setting up the physical infrastructure within the institution (initial purchase of hardware, cooling, maintenance, etc.), only paying for the resources we needed through fluctuating use.

Testing of the server showed that once usage reached the level of around 20-30 simultaneous users, the strain started to show and the server struggled somewhat. If a pilot service such as the Transcoder were to take off and gain widespread and popular usage it is this kind of issue that would need to be able to be quickly addressed and resolved.

The traditional deployment sitting on an institutional server could cause problems here as in a case like the above, a new server would need to be procured, installed and setup. This would, of course, take time and result in a drop in service.

Past experience at the institution has taught us that the procurement and setting up of physical servers in-house can be quite a lengthy process that involves much negotiation around people and policies. Typically this process would take anything between 2 to 3 months.

However, deploying in the cloud offers us quick and easy, flexible scalability. If the server starts to struggle under the weight of heavy use we can simply login to our AWS account through the web and add a new image. This is also the case for any spikes - or surges - in usage. The advantage of cloud computing is that new images can be quickly created to cater for the increased load, then removed when usage drops to its more general level, rather than having to purchase, install and maintain enough capacity to cope with usage at its highest level throughout a year.

Another benefit to the cloud was highlighted during the project when the institution suffered a power cut, knocking down all the IT infrastructure. This, of course, had no impact on the Transcoder service. Two years before a similar power cut had taken a departmental server offline and when power was restored there was an inevitable queue to get various servers back up as quickly as possible. Several things were priorities, including the University software licence server, the VLE and Student Record Service. In that context, the department server is relatively 'low priority' and so there was a delay on getting it restarted and into service again. There can be occasional downtimes in the cloud, as has happened to Amazon EC2 three times in 2007, 2008 and 2009, but they don't tend to last long. Also, given the publicity that such outages generate, and the crucial importance of uptime to its business model, Amazon clearly has a very strong incentive to bring major resources to bear in minimising downtime.

4.3 Costs

As has been outlined above, one of the major benefits of deploying in the cloud is financial. When utilising the cloud, there is no capital investment in equipment and any energy and maintenance costs are covered in the monthly bill for AWS.

Below is a basic outlining of how the costing of AWS compares with hosting in-house. The choice of server is an interesting one as a pilot service must purchase enough capacity to handle the highest level of usage that could occur, if one wishes to avoid disruptions and further costs. This means that a mid to high range server may be procured for a service that turns out to have low usage, with the risk that expensive equipment costs go to waste. However, services can of course take a risk on procuring and setting up a small scale, less expensive server, and "hope for the best". If demand and usage rises, however, a pilot may find themselves having to reinvest in further equipment, involving more cost, negotiation and potential disruption to the service.

The costs in the table below are based on a server of same capacity to that previously procured within the institutional department managing the transcoder project. This is a DELL PowerEdge 1950. Quad Core Intel Xeon L5310 LV, 2x4MB Cache, 1.6GHz, 1066MHZ with 4Gb of memory.

Cost of server	£2,200.00
Yearly energy costs (approx)	£325.00
Total cost over 3 year service*	£3,175.00

* On 'known' costs

When dealing with a physical server, in-house, a service also needs to take into account the depreciation of the equipment. This is based at approximately 20% per year, effectively writing off the cost of the server after 3 years.

Calculating costs of an internally hosted service can be rather difficult as many institutions don't really have a clear picture - and certainly not hard figures - of how much it costs to maintain a standard server in a rack. Much of the time a putative figure is attached by doing a general estimation. How many staff members maintain how many servers, how much time is invested and how much does it cost in salaries?

Given that this data is difficult to ascertain, we looked at the costs that we can be sure of, the minimum - cost of the server itself and the cost of running it. The approximate cost of energy consumption and worked with the figure given for a medium energy profile server (Innovate IT Ltd n.d.)

The costs of AWS is far easier to detail as it is simply a monthly cost based on usage.

Initial setup costs	£0.00
Monthly cost (average)	£45 (at current exchange rate of 1GBP to 1.53USD)
Total cost over 3 years	£1,620.00

Of course, there is some play in this as it isn't a black & white comparison. The AWS deployment has been sufficient to cater for the usage levels of the transcoder so far.

However, if usage increased over time it could be handled by the institutional server scenario outlined above but require the service to add another image on AWS.

We could simply look at a situation like this in its extreme and calculate the cost incurred by adding a 2nd instance, thereby doubling the cost to £3,240 over the 3 years. This is slightly higher than the in-house setup, however, the added costs of maintenance through man hours incurred - that are more difficult to calculate, given those putative estimates IT departments tend to make. And, still, this "doubling up" estimate may not reflect reality.

It's here where doing a direct comparison of costing can be difficult as there are variables at play that simply aren't fixed. When using the cloud for deployment of a service, if usage spikes, requiring the addition of further resources, the added server images are not permanent additions. The extra resource is added to cater for a spike, then removed when usage settles to general levels. So, in viewing the increased cost in a "doubled up" instance scenario as above it really is unlikely to happen. However, it does still - clearly - point to the cost benefits for a pilot service such as the transcoder. The real cost would look more like your home phone bill, being generally around the same price each month but the occasional jump when it's been a particularly talkative month.

The wider picture is that, beyond the numbers for the running of individual servers, there is a limited realistic capacity for an institution to keep procuring, deploying and running servers. At Bolton, physical constraints in the building, and limited energy and staff budgets, means that growing these resources to accommodate specialised functions such as the transcoder requires levels of investment that are pretty unlikely. Even if the investment were forthcoming, it is doubtful whether the return on it would be justifiable.

5 Transcoder feasibility

5.1 Aims and Objectives

5.1.1 Transcoding performance

With regard to the technical feasibility of the transcoder, the basic parameters from the project proposal were the ability to convert to and from:

- IMS Content Packaging 1.1.3
- IMS Content Packaging 1.1.4
- IMS Content Packaging 1.2
- SCORM 1.2
- SCORM 2004
- IMS Common Cartridge 1.0

With the version that is current at the time of writing (the latest is always available via http://purl.oclc.org/NET/transcoder), this has been achieved, albeit with a few provisos and limitations.

First, IMS Content Packaging 1.1.2, 1.1.4 and 1.2 all share the same core element set in the same namespace, which means that an application designed to accept one of these should accept the other two as well. The main difference between 1.1.3 and 1.1.4 is in some rule-clarifications, while 1.2 adds a few new features in a separate namespace. These extended 1.2 features have not been implemented in the transcoder, but they have not, as yet, been implemented by anyone anywhere else either. In short, the transcoder accepts 1.1.3, 1.1.4 and 1.2 packages, but only turns out 1.1.4 packages, which are acceptable to all implementations of these three versions (such as Moodle 1.9).

A slightly greater issue is the fact that there is no SCORM 1.2 output, just the later SCORM 2004 version. The main reason for that is one of resources: SCORM requires the addition of (javascript) code to each page of content in a package. Automatically inserting this code is possible, but labour intensive to develop in a robust way. Doing it in ways that comply with both versions of the profile proved to be too much. It should also be borne in mind that, since the SCORM 2004 output of the transcoder doesn't involve the sophisticated sequencing feature, the SCORM packages should fare reasonably well in a wide variety of tools.

In practice, the ability of the transcoder to convert between plain Content Packaging versions and Common Cartridge 1.0 was best illustrated during the first Common Cartridge Alliance public interoperability demonstration in Long Beach, California in 2009. Several authoring tools were in evidence there, but no VLE capable of accepting cartridges. Without prior preparation, the transcoder was used to convert various

Common Cartridges into Content Packages, and run these in a stock Moodle installation. It worked without a hitch.

As anticipated, though, the transcodings are lossy under some combinations. Features such as SCORM's sequencing or Common Cartridge's Question and Test Interoperability (QTI) tests do not make it across when converting from these formats, nor can they be invented easily when converting into these formats. Further development could lead to workarounds and solutions in some of these cases, however (see section 5.3) .

5.1.2 Usability

The other technology developmental aim of the transcoder lies more in the area of usability. For FE, the intent was to enable the use of materials from the NLN, BBC Jam and other collections in contexts with little expertise or support infrastructure to aid the process. In practice, while NLN materials worked well in the transcoder, we did not have volunteers to run through the scenario in that environment to validate the usability. It should be noted that NLN materials continue to work well in popular current VLEs such as Moodle 1.9 without transcoding, and that the materials of BBC Jam have never been released.

For HE, the main scenario we had hoped to be able to support was the bulk transcoding of proprietary packaging varieties into standard ones. This we did not succeed in, mostly as a consequence of the architecture of the overall service, and the design of the central transcoding. This will be expanded on in subsequent sections.

5.2 The architecture and implementation

The pilot service was designed to operate in a very simple, easy-to-use fashion, using a design based on existing online content services such as Zamzar.com (see Figure 1.) Users upload a package file, which is stored as a temporary object in a backend store. The user then chooses a target format for the package from a list of supported output formats, and enters their email address. The service then queues the conversion job, and executes it with one of a collection of programmed conversion routines. The converted output is then stored with a new address for downloading. When conversion is completed, the service notifies the user by email with the URL where they can download it.

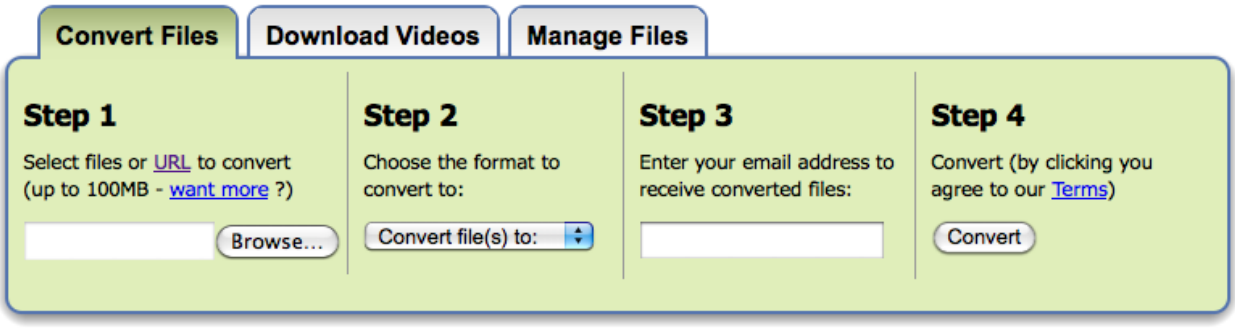

Figure 1: The upload interface of http://zamzar.com/

Since the transcoding process in the service rarely takes that long, users requested that the email alert be made optional in the transcoder, which we did. Converted packages can now be downloaded immediately after uploading, as illustrated in Figure 2. Other than that, the transcoder followed the pattern outlined in the plan.

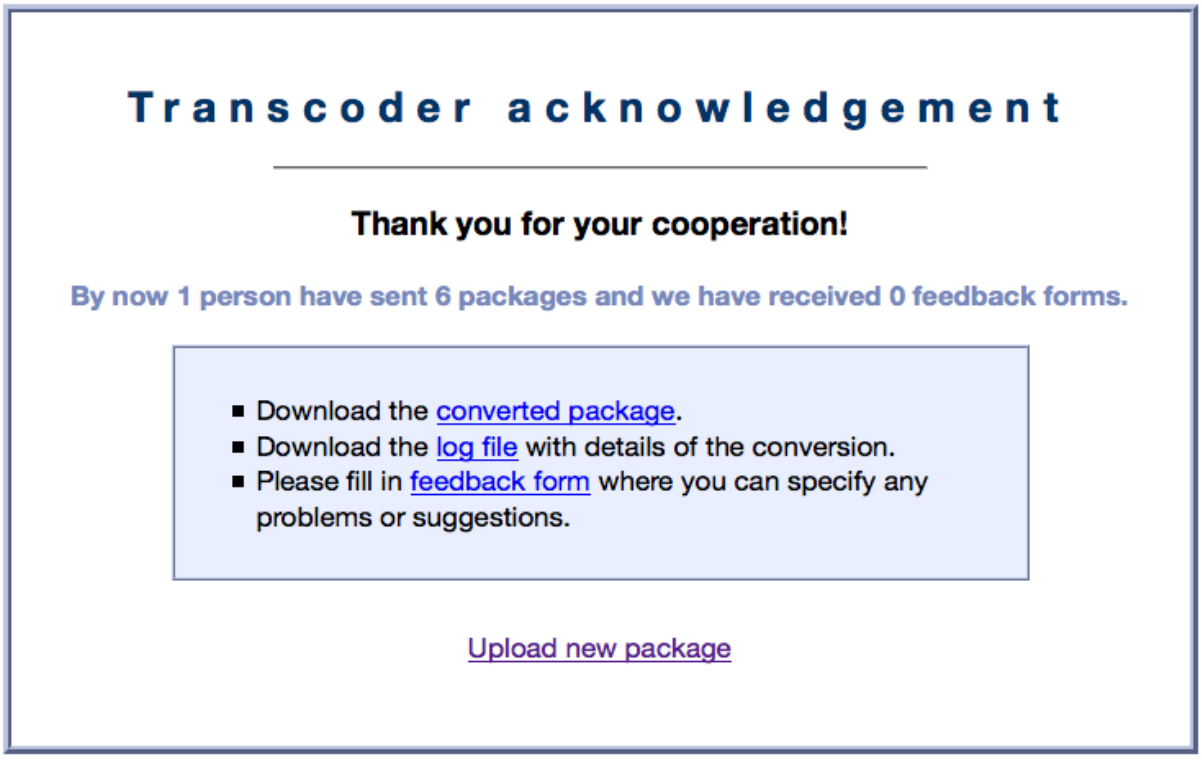

Figure 2: Transcoder upload acknowledgement screen

In order to support such a pattern, the architecture in Figure 3 was drawn up, and then implemented:

Schematic of eLearning Content Transformation Service.

1. A user submits a package through an upload handler, and selects a target format.

2. The service uploads the package to S3, and adds the reference to it to an SQS queue.

3. The SQS queue passes it onto an EC2 processor for translation.

4. On completion the EC2 processor notifies the service that translation is completed.

5. The service notifies the user (e.g. by email) that their translated package is now available for download.

6. The user access the download handler of the service, which downloads it from S3.

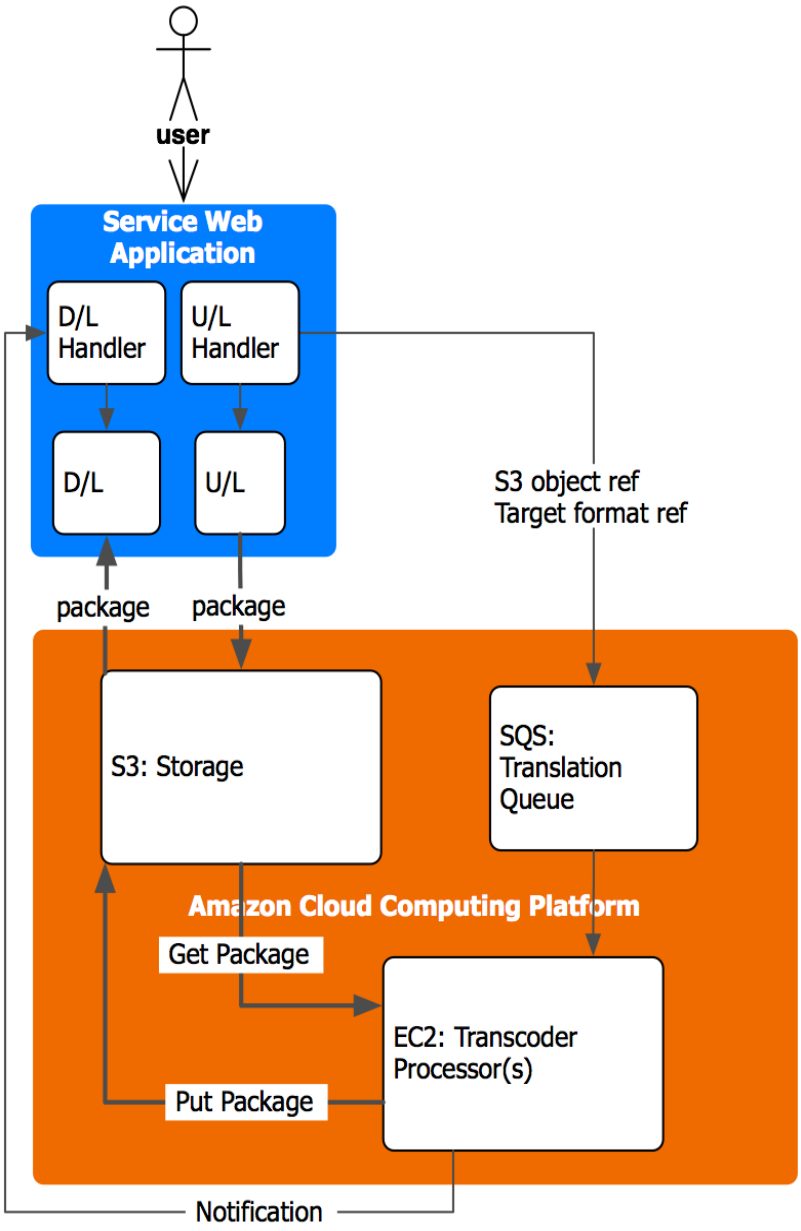

Figure 3: the transcoder architecture

Two things are worth noting about this design. One is that it relies heavily on the Amazon web services for all major aspects of the architecture. The services that were to be used by the pilot are the Elastic Compute Cloud (EC2) service, the Simple Storage Solution (S3) service, and the Simple Queue service (SQS) . The idea was that these would provide all the major infrastructure pieces of the service. The actual development work could then focus on the transformation logic and the user interface.

In practice, the latest version of the transcoder is much more selfcontained as a standalone EC2 machine image. No use is made of the S3 and SQS services, mostly

because neither service load nor contention or a need to persist data prompted the need for them. An important effect of this change is on the portability of the service, which is a topic that will be explored in greater depth in Section 6

The other thing to note about the overall architecture of the transcoder is the extent to which it has been designed around resilience and scalability. Each package needs to be uploaded separately, and the results of the transcoding are not necessarily returned immediately, but when the service is ready.

This choice appears to have lead to an admirable robustness; the service only needs to deal with one package at a time, and can take as long as it needs to transcode it. Given that transcoding is computationally a potentially very 'expensive' (i.e. CPU cycles and memory intensive) operation. As noted, load testing using JMeter (Apache Software Foundation n.d.) indicated that the transcoder could sustain about twenty concurrent users before performance dropped to unacceptable levels.

The feature that provides such robustness is also the source of one of the most frequently requested changes in the design of the service, however. For one, the set-up only allows one package to be transcoded at a time (at least from a single user perspective), rather than in bulk. While the great variability in package size and complexity may make bulk transcoding potentially prone to time-outs or outright failure, a version of the transcoder could be trialled that would allow batch transcoding, but redesigned to rely on the input and output queues envisioned in the plan. That way, the service can still take as long as it needs to, even if the load could increase.

The other frequently requested improvement is an API on the service. This would make it possible to both convert large numbers of packages – piecemeal if necessary – and it would allow developers of VLEs and repositories to make the whole transcoding process entirely transparent to the end-user.

For this aspect, the performance of the transcoder seems good enough in practice to allow the future development of a REST API without needing to rely on potentially complex asynchronous message behaviours. Within a package size based treshold, and perhaps an API key based limit, service consumers should be able to deposit and receive packages using simple http POST and GET operations.

5.3 Possible future developments

Several avenues of possible further development exist, each of which will be discussed in turn:

5.3.1 Supporting proprietary packaging varieties

Adding both Blackboard and WebCT flavours of Content Packaging to the transcoder was in the plan, is a clear priority for HE stakeholders and was the most frequently requested feature. The problem with adding them lies in the design of the central transcoding processor illustrated in Figure 3. This design relies on point-to-point XSL transforms between each supported format. That was a well-founded decision, since XSLT was designed specifically to deal with XML transform cases such as the transcoder. It is also widely supported, the XSLTs can be re-used in many other contexts and technology stacks, and they are relatively high performance.

The point-to-point nature of the design also means, however, that supporting two new packaging formats requires sixteen new transforms, and that's before re-developing SCORM 1.2 export (that'd mean five more). That sixteen also doesn't count the seven existing separate metadata XSLTs. Even if all of these XSLTs are successfully added to the nine existing ones, further development will become exponentially more intractable, and maintenance that much harder.

A pragmatic solution could be to limit support for further formats to import or export only, depending on requirements. Feedback indicates that most people would be interested in the transcoding of standard packages into proprietary versions. This would also make sense from the point of view of the OER programme, as it would mean that a single standard resource can reach many different VLEs.

On the other hand, the ability of VLEs to consume standard content has always been markedly better than their ability to export it. A transcoder that is limited to the export of proprietary formats would therefore risk adding a declining amount of value while not doing anything about the ongoing vendor lock-in issue.

A yet more pragmatic solution suggested by Scott Leslie (personal communication) is to add an export version without an XML control file - effectively a static website in a zip archive. This would not require any new transforms, and the end-result would work in

nearly every conceivable webplatform. This practice has also been found frequently in the JISC OER programme (Robertson 2010).

The definitive solution to the combinatorial explosion of point-to-point transforms is to go for an architecture with a lossless, all-encompassing union of formats as the intermediary format. Adding each new format to such a set-up would only require two more transforms: to and from the intermediate format. Fortunately, such a model exists: IEEE RAMLET. The first half of a proof of concept implementation of that standard has been done, the architecture of which is outlined in (Kraan 2009). The remainder of the implementation could re-use some of the service logic and transform infrastructure of the existing transcoder, but its very different, triple store based architecture would still require significant development.

It would also have two significant benefits, however: one is that RAMLET is designed to deal with many more packaging formats such as MPEG 21 DID, METS, IETF Atom and OAI-ORE. The other is that a RAMLET implementation could turn every package it would deal with into Linked Data 'for free'.

5.3.2 Adding QTI functionality

As noted previously, some transcoding combinations lead to the loss of features from one format to another. The most prominent of these are the loss of sequencing when going from SCORM 2004 to any other format, and the loss of QTI based assessments when going from Common Cartridge or QTI packages to any other format.

Of these, sequencing is perhaps the most difficult to recommend as a target for further development. It seems to be only sparingly implemented in the core SCORM communities, and little or at all in UK FE or HE. Moreover, existing commercial tools such as Icodeon's SCORM player already do a very good job of playing sequenced SCORM 2004 packages inside various VLEs for those who need that functionality.

The ability to preserve QTI through a transcoding is another matter, though. For one, the feature is widely used in Cartridges (IMS GLC n.d.), particularly those from commercial publishers. Online tests are also used rather extensively in the JISC community, for another.

Fortunately, various JISC projects have developed the basic components to add QTI to the transcoder. Since CP or SCORM compliant VLEs don't have the ability to deal with QTI content, the logical workaround would be to keep the QTI content on the server, and insert references to those items in the converted output packages. Following a link in such an output package would then load the test, and play it in place. Communication of scores back to the VLE would be difficult, but for basic formative use, such a set-up should be fine. A return of scores to the VLE could be added later using IMS Basic LTI or similar web services, if there is demand.

While there is no immediately obvious code that can render the QTI 1.2 that is included in CC 1.0, there a number of libraries that can render QTI 2.1; the JISC toolkits and demonstrator programme sponsored QTIengine and APIS and derivatives, for example. There is also an open source QTIMigration script that is being adapted to allow conversion from the CC 1.0 QTI 1.2 profile into an equivalent profile of QTI 2.1 in CC 1.1/2 (Young n.d.). Integrating a combination of the migration script and a rendering library would be enough to add QTI playback capabilities to Content Packages and SCORM objects.

A nice corollary of a solution that involves QTI 2.1 rendering libraries is that it would make both the transcoder as well as numerous VLEs futureproof against forthcoming versions of Common Cartridge. Once installed, such a capability could also provide support for the full gamut of QTI 2.1 item and test capabilities to an equally large range of VLEs- since that is what the rendering libraries are already capable of.

To be fair, there has not been much evidence of a demand for a QTI rendering feature in the transcoder in the UK, possibly because Common Cartridges have not had much traction, and the e-assessment community appears to be happy with the QTI tools that are already there. It may also be a matter of pointing out the possibility.

5.3.3 Adding integration options

One feature that has been requested the addition of a simple REST API to the transcoder, as noted in the Architecture and Implementation section above. Given the present architecture of the transcoder, there are two implementation options. One is to follow the original pre- and post-processing queue pattern, and introduce batch submission of packages. This keeps loads manageable, but asynchronous webservices

are difficult to design and implement well- both on the provider and consumer end. A human interface doesn't seem practical, though WebDAV based submission might be worth exploring.

A more promising route would be to allow programmatic, one-by-one submission of packages. For one thing, the current user interface could already be used as a REST API by a hack-happy developer. Implementing a 'proper' API with the same characteristics is therefore pretty easy. The other advantage of such an API would be that outsize or gnarly packages can fail individually, rather than fail the whole batch or even the whole service. The potential load on the service could be considerable and quite 'spiky', but coping with such loads is the main strength of a cloud based solution.

There has also been some demand for selfhosted or desktop variants of the transcoder for those who do not want to make use of the SaaS option. The main application would be to build the transcoder directly into ingestion or export routines of repositories or VLEs. Fortunately, with the redesign of the service as a more self-contained Amazon EC2 image, the code has effectively become a servlet that can be deployed straight into ubiquitous servlet containers such as Apache Tomcat. Other than some forthcoming documentation, no further action is required there.

5.3.4 Adding standard packaging varieties

There has been little demand for adding export support for other known packaging versions such as those defined by the NLN or BBC Jam. As noted, that low demand appears to be mostly because the BBC Jam materials will not be available (The Guardian 2008), and the existing NLN materials work well enough in most VLEs. They also work well as input to the transcoder as is. It appears that the forthcoming Becta content packaging profile may stay close enough to existing profiles not to require too much adaptation.

5.4 Outcomes

The main issue when designing a service like the transcoder is reconciling the conflicting demands of usability and robustness. Even if a cloud-computing platform makes the provision of more computing resources relatively quick and easy, there still needs to be some way of managing the latency inherent in the process.

On the other hand, the design of the current version of the transcoder does limit its usability for some core users. When designing a speculative new service, erring on the side of robustness over usability was the best choice. In future, development could split and target either robustness or usability, depending on the use case (see section 9).

The inclusion of proprietary versions of IMS Content Packaging seems a very obvious potential new development. The point-to-point design of the transcoding processor militates against such an extension, though. Adding two new formats could require considerable resource, and any further development beyond that will almost certainly not be feasible or economical. Re-architecting the transcoding processor around a common intermediate format would be sustainable, but would also require a considerable level of investment. Meanwhile, reasonably good means for the main VLE platforms to 'play' existing standard content do exist. There is also a simple and effective hack which could alleviate the issue for those users who don't have access to standard package players on their VLEs. There does not appear to be a lot of demand or need for the inclusion of other standard profiles of Content Packaging.

Of all functional additions to the transcoder, adding server-side QTI support is feasible and has potentially high impact. It would be a particular example of Distributed Learning Environments (MacNeill & Kraan 2010) adding functionality, without requiring major change in local systems.

Implementation experience has already led the transcoder to be much easier to integrate in a variety of different systems, so beyond an API and/or bulk upload facility, there appears to be little need for further development here.

Neither the performance and usability issue, nor missing functionality seem quite able to explain the low usage of the service. One factor may have been a lack of visibility, but recent exposure doesn't seem to have made much of a difference. A likely explanation could be that the JISC community at least simply doesn't make much use of packaged content. In the Open Educational Resources (OER) programme – where you might expect some use –, (Robertson 2010) found very little uptake of the technology, and if there was any, it was usually because it was a feature that happened to be implemented in a tool.

6 Amazon webservices as a deployment platform

6.1 AWS and other cloud providers

There are considerable differences between cloud providers, particularly with regard to the delivery levels (Kraan & Li 2009). There's a general distinction between cloud services that deliver as Software as a Service (SaaS), of which the transcoder itself is an example. In terms of what it could run on, there is a choice between two more broad levels: Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Amazon is generally seen as IaaS, whereas competitors such a Microsoft's Azure and Google's App Engine are PaaS. AWS is rather high level compared to other IaaS providers such as Rackspace or Bluelock, however.

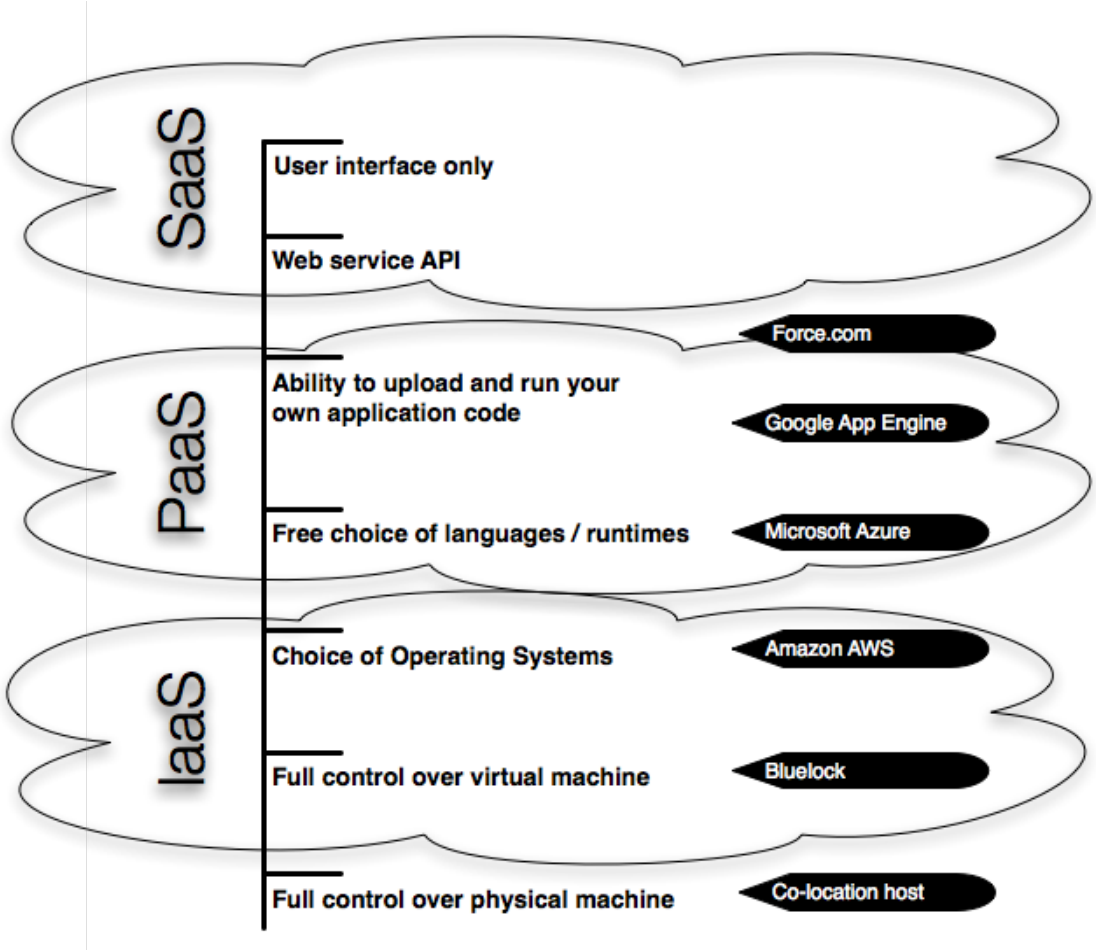

Figure 4: A hierarchy of cloud products

In principle, the transcoder could have been developed to work at any of these levels, but there are some advantages and disadvantages to each level that guided our choice to Amazon's EC2.

Taking it from the top, a very high level service such as Force.com is very easy to develop on and takes away the need to worry about scalability or maintenance. But it can only do that by restricting what can be developed on the platform very severely. The transcoder is unlikely to be in its scope.

Google's App Engine is a rather more likely candidate. Its' most immediate attraction for a speculative service such as the transcoder is the fact that its use is free before certain levels of resource usage have been reached. You pay after that, but costs are quite low. Other advantages include the fact that it scales seamlessly, which means that no technology level interventions are necessary to allow the application to cope with rapid increases in load. There is no need to worry about updating or maintaining the software infrastructure the transcoder would rely on either; Google does that.

To enable these advantages, there are some considerable restrictions. For one, when the transcoder project started, the only language available was Python. This restricts the choice of developers, and – crucially – existing code libraries that can be used for things such as XML transformation or data persistence. Since Google started to support Java as well, this has become much less of an issue, and it appears that the current version of the transcoder could be modified relatively easily to run in App Engine [1].

Such a modified version would have limited portability, though. While alternative implementations do exist (Appscale Project n.d.) and could be used on a local machine, they do not appear to be readily available as an alternative cloud offering.

Microsoft's Azure platform is lower level than Google's App engine: it largely allows applications to run as they do in other Windows environments, but with a number of services available to make use of the cloud borne nature of the product.

Compared to Google App Engine, pricing starts at the first instance (disregarding time-limited special offers), and is organised per instance. If the application uses more bandwidth, CPU or storage resources, new instances need to bought, brought online and the load balanced between them. Unlike AppEngine, scaling is not seamless or automatic. In short, more maintenance is required.

1 The main modification would involve a change in the storage layer. All the other major frameworks the transcoder relies are available in the App Engine. http://groups.google.com/group/google-appengine-java/web/will-it-play-in-app-engine

In return, many languages and libraries that work on Windows, also work on Azure (but not all (Microsoft n.d.)). The associated services focus on easing interoperability with internally hosted applications and migration. For an application of the scope of the transcoder, finding an alternative cloud should be possible.

As it is, the Azure was not available for general consumption at the start of the transcoder project. It could be moved from Amazon AWS to Microsoft Azure with little trouble, though. Pricing is currently identical to AWS, assuming a Windows OS (cf. (Microsoft n.d.), (Amazon n.d.))

Without that assumption, AWS is currently about 16% cheaper. The reason is that it allows a free choice of operating systems, including Linux or other Open Source OSs. That also means that there is practically no limit on the languages or libraries that can be used. The drawback of this liberty is that there is yet more for the cloud customer to look after in terms of code maintenance. Other than that, the advantages and disadvantages of AWS are similar to those of Azure.

One notable feature of AWS is that the open source implementation of the various services (Eucalyptus project n.d.) is mature, widely deployed and has an active development community. Re-deploying AWS applications elsewhere is therefore relatively straightforward, even if the app makes use of a variety of AWS specific services.

Compared to traditional hosting (on a dedicated or shared server), cloud infrastructure such as AWS has the disadvantage that certain features such as dedicated IP addresses and, most importantly, persistent and backed-up system level data storage are extras that work in new ways and need to be paid for separately. On the other hand, such hosting solutions cannot scale as rapidly and flexibly, don't tend to have such slick management interfaces and are more expensive.

6.2 Implementation

The transcoder did require some work when the code was handed over from Knowledge Integration Ltd. to JISC – CETIS. None of that was due to either the quality of the code, or any fault in AWS. The work was purely to do with resolving subtle differences in the configuration of the operating system, the Tomcat application server and some other

dependencies. These issues are frequently encountered with any conventional server software development, but are arguably less of an issue in a more tightly controlled platform such as Google App Engine.

In our experience, compared to conventional hosting solutions, AWS is notable for its ease of deployment and management, especially since the release of an instance management web interface. While the initial set-up of an instance does take some doing – mostly for security reasons – other tasks are straightforward and quick, and a wealth of information is readily available.

As noted in an earlier section, the transcoder was originally designed to rely on more Amazon webservices than just the Elastic Compute Cloud (EC2). The use of the Amazon Simple Storage Service (S3) and the Amazon Simple Queue Service (SQS) was not used in the final version because use patterns and the performance of the software allowed the instance storage to be used.

This has two some consequences. On the downside, data does not really persist on the transcoder's EC2 instance. Packages that have been contributed will be lost if the instance is restarted for any reason, for example. For the transcoder, this is not an issue, but for other services, it might well be. On the upside, it made the transcoder even more readily portable- it can be redeployed on virtually any server in minutes (Sherlock 2010)

6.3 Outcomes

With cloud products, there seems to be a fairly linear trade-off between flexibility and portability on the one hand, and ease of management and low cost on the other. It is worth considering what point of the scale is most advantageous for each application.

In the case of the transcoder, the choice for the AWS platform was simple in that it was the most mature available at the time, but even today, it is probably best hosted there. Usage is not so spiky that seamless scaling is important, and the management overhead is easily covered inhouse. The portability offered by AWS is a significant advantage. We don't appear to be the only ones, since a recent survey by Goldman Sachs found that 77% of companies with cloud developments use AWS (Ricadela 2010).

For other speculative services, though, the free initial usage band and low maintenance overhead of Google's App Engine could well be decisive. It only requires some care in the development to maintain portability to other platforms.

7 Conclusions

Whether a niche service such as the content transcoder works best as SaaS or as conventional downloadable software is bound up with usage scenarios on the one hand, and technical architecture considerations on the other. More widely, SaaS seems to have more traction with highly commoditised services rather than with new, speculative ones. Since there is no clear underlying reason for this, the case might still be open.

By contrast, the business case for piloting innovative and speculative pieces of software such as the transcoder on cloud infrastructure is quite convincing. There are considerable potential savings in both hardware procurement and running costs. In combination with much lower investment risk, this means that cloud computing can help lower the barrier to entry significantly for innovative but unproven server software. At least as important is the much higher agility of deployment and decommissioning, and the fact that cloud computing can help place the responsibility for a service closer to those who have the greatest interest in it. It makes taking responsibility for specialised software more manageable for such communities as well.

These advantages were made quite clear when the uptake of the transcoder service turned out to be very low. The most likely cause for the low uptake of the service is a low level of usage of packaged content.

Other factors include the conflicting demands of two main use scenarios and the different service architectures that would favour either. Reconciling these conflicting demands may involve splitting the transcoder into two separate applications.

The addition of features such as adding support for proprietary packages may also help uptake, but such a development would run into significant limitations of the transcoding processor's design. A simple and inexpensive workaround exists.

Of other potential additions, server-side QTI support looks the most promising.

8 Implications

8.1 The cloud computing business case

Our experience of developing and deploying software on the cloud, as compared to hosting it conventionally is strong, but not very well quantified. Though cloud solutions are not directly comparable either with each other or with conventionally hosted servers, a better approximation of relative ecological and economic costs seems both possible and worthwhile.

8.2 Developing the transcoder

We found that the robust design of the transcoder militated against its usability. The answer to reconciling these conflicting demands of usability and robustness could lie in a separation of concerns. In contexts where spiky loads are low, and transparent transcoding very important – regularly importing a few packages from a repository into a VLE by a lecturer, for example –integrating a copy of the transcoder in local systems via a simple API may be the best way to go.

By contrast, in situations where some oversight of the process can be expected, and volume is more important than speed – when converting a collection of existing materials, for example –, a version of the current cloud based transcoder that allows bulk upload may be most effective.

8.3 Picking the right cloud (or the ground)

As the previous section indicates, some scenarios are simply better served with a local copy of the software. The cloud is not a homogenous either / or, however. Cloud products sit on a scale of decreasing flexibility and cost and increasing convenience and simplicity. A practical guide to such products, with plenty of examples, could help make implementers take informed decisions.

Whether SaaS is best suited for offering commodity functions such as email and file storage, or for offering innovative, niche functions is still an open question that is worth exploring further.

9 Recommendations

9.1 Develop guidance for the use of cloud computing services in agile research & development projects, and include it in JISC ITTs

We have found that there are clear benefits to the use of cloud computing services for the hosting of speculative or innovative new services. Easy of deployment, low costs and manageable risk mean that it can bridge the gap between a good idea and working code significantly. It could therefore be recommended as a route for JISC software development projects.

The current variety of products and services, and their relative merits, do bear some practical guidance, however. Different purposes will require different cloud services at different levels

9.2 Further development of the transcoder service

Further development of the transcoder has the following options, in order of priority:

- Add a control-file less package with it's own frameset for navigation (zipped website) as an output option
- Add a bulk processing version and a version with a REST API
- Add QTI hosting and playback
- Add proprietary package import and export

10 References

Amazon, Amazon EC2 Pricing. Available at: http://aws.amazon.com/ec2/pricing/ [Accessed April 30, 2010].

Anderson, T. & McKell, M., 2001. IMS Content Packaging Information Model. Available at: http://www.imsglobal.org/content/packaging/cpv1p1p2/imscp_infov1p1p2.html#1146150 [Accessed April 30, 2010].

Apache Software Foundation, JMeter - Apache JMeter. Available at: http://jakarta.apache.org/jmeter/ [Accessed April 30, 2010].

Appscale Project, appscale - open source Google App Engine. *University of California at Santa Barbara*. Available at: http://appscale.cs.ucsb.edu/ [Accessed April 30, 2010].

Bailey, W., 2005. What is ADL SCORM? Available at: http://zope.cetis.ac.uk/lib/media/WhatIsScorm2_web.pdf.

Eucalyptus project, Eucalyptus Community. Available at: http://open.eucalyptus.com/ [Accessed April 30, 2010].

IMS GLC, Learning and Educational Technology Product Directory. *IMS GLC*. Available at: http://www.imsglobal.org/ProductDirectory/directory.cfm?show_all_nav_listCCAlliance6=1 [Accessed April 30, 2010].

Innovate IT Ltd, Green IT application - Virtualisation Savings Calculator. *Innovate IT*. Available at: http://www.letsinnovateit.com/greenit-app.html [Accessed April 30, 2010].

Kraan, W., 2005. CETIS-Third CETIS/LIFE codebash goes public. Available at: http://zope.cetis.ac.uk/content2/20050329191619/ [Accessed April 30, 2010].

Kraan, W., 2009. RAMLET implementation study report. Available at: http://www.ovod.net/wilbert/ramlet/ramletImplementationReport2.pdf.

Kraan, W. & Li, Y., 2009. Cloud Computing Briefing Paper. Available at: http://wiki.cetis.ac.uk/images/1/11/Cloud_computing_web.pdf.

MacNeill, S. & Kraan, W., 2010. Distributed Learning Environments briefing paper. Available at: http://wiki.cetis.ac.uk/images/6/6c/Distributed_Learning.pdf.

Microsoft, Windows Azure Interoperability | Developers | Windows Azure Platform. Available at: http://www.microsoft.com/windowsazure/interop/ [Accessed April 30, 2010].

Microsoft, Windows Azure Platform Consumption. Available at: http://www.microsoft.com/windowsazure/offers/popup.aspx?lang=en&locale=en-US&offer=MS-AZR-0003P [Accessed April 30, 2010].

Ricadela, A., 2010. Amazon Looks to Widen Lead in Cloud Computing. *BusinessWeek.com- msnbc.com*. Available at: http://www.msnbc.msn.com/id/36849177/ns/business-businessweekcom/ [Accessed April 30, 2010].

Robertson, J., 2010. The use of IMS CP in the UKOER programme. *John's JISC CETIS blog*. Available at: http://blogs.cetis.ac.uk/johnr/2010/03/08/the-use-of-ims-cp-in-the-ukoer-programme/ [Accessed April 30, 2010].

Sherlock, D., 2010. Deploying Content Transcoder. *David's Blog*. Available at: http://blogs.cetis.ac.uk/david/2010/04/28/deploying-content-transcoder/ [Accessed April 30, 2010].

The Guardian, 2008. No relaunch for £150m BBC Jam | Media | The Guardian. Available at: http://www.guardian.co.uk/media/2008/feb/28/bbc.digitalmedia [Accessed April 30, 2010].

Wiley, D.A. & others, 2002. *The instructional use of learning objects*, Agency for Instructional Technology. Association for Educational Communications & Technology.

Young, R., Assessment tools, projects and resources. *CETISwiki*. Available at: http://wiki.cetis.ac.uk/Assessment_tools%2C_projects_and_resources [Accessed April 30, 2010].