Micro-services based distributable workflow for digital archives

Heikki Kurhinen and Mikko Lampi, Mikkeli University of Applied Sciences, Mikkeli, Finland

Abstract

Managing the costs and the workloads in digital preservation requires automation and supportive tools. Micro-services is a well-tested and widely adapted architecture, proven in archive systems [1] and operating systems like Linux and UNIX. The automation level can be increased by combining the services into workflows and making the processing distributed. Resource consuming tasks such as ingest, migration and format conversion can be streamlined with this kind of approach.

This paper is about the development of a workflow engine prototype for micro-services based distributed processes in digital archives. The prototype is demonstrated with a simple use case of digital content ingest workflow. The design goal was to support work done in the digital archive developments and to provide a simple and extendable tool for processing the digital content.

Background

In digital archives, there is a need to process the data in different phases of its lifecycle. For example, ingesting requires certain steps to normalize and prepare the data for long-term preservation and access. There are other requirements for migration and file fixity processes. Most of the processes are able to be automated in order to increase the cost-efficiency and minimize the manual work. Processes can be modeled as actions and combined into workflows which support decision making and the results are collected into logs or other audit records. The workflows can be automated and controlled by a workflow engine. The engine is usually either an existing software or developed specifically for the purpose.

There are plenty of existing workflow engines available, both open source and proprietary. Later in this paper, there is a brief description of the research on the available tools and reasons for developing a new engine. In brief, the majority of workflow engines are complex, multipurpose modeling tools which include a moderate learning curve and specific modeling languages. As a result, we found that building a micro-service supporting engine would also benefit the whole digital archiving community and other projects requiring an agile workflow solution. It was found important that the engine is completely open source. More about the benefits can be found later on this paper.

This development project was carried out as a bachelor thesis by Heikki Kurhinen in Department of Electrical Engineering and Information Technology at the Mikkeli University of Applied Sciences (MAMK) as part of the ongoing Open Source Archive (OSA) project. OSA is carried out by MAMK and is funded by the European Regional Development Fund (ERDF). The project started in the summer of 2012 and will continue until the end of 2014. The goals of the project are to develop a service archive system and to search for and test a dark archive solution for long-term preservation. The focus and key values are open source, sustainable solutions and user centered development. After the project is completed, the results will be migrated with MAMK's digital archive services and the current archive platforms.

Research and evaluation of available tools

Research

From the beginning of the project, it was acknowledged there was no point in re-inventing existing and suitable products. A research was conducted to find out current workflow engines and tools. The requirements were collected and analyzed. The most important were ease of use, being open source, support for micro-services, easy integration via a well-documented API, extendibility and clustering features.

Some of the well-known products evaluated were Apache Camel, Taverna, jBPM and Activiti. All of the mentioned are good workflow engines for their designed purpose, but did not exactly match our needs. They could be used to build the workflow system, as a framework with some markup languages which are needed to model the jobs and workflows. More or less, the tools would also dictate the implementation languages of micro-services. More about the encountered problems is described below.

Problems

All of the evaluated workflow solutions were of mid to high complexity and required lots of effort to be efficiently utilized and deployed even in the development phase. Proprietary engines could not be modified or contained components which were not extendable. Even the open source products had much of the functionality programmed into the core of the engine making them harder to adapt and integrate into the existing projects. Due to the reasons described earlier, open source was the preferred solution and the proprietary engines were ruled out.

There were some promising micro-services based workflows, such as in Archivematica and Islandora, but they were either tightly integrated into their respective environment and scope or the development tools were not optimal considering our available resources. As an example, Perl or Ruby developers are not that widely available due to education and academic traditions in software development. The most other open source tools were heavy and clearly targeted for large enterprises and complex software. The configuration of these tools would probably require more resources than developing a simple and better sustainable engine. Other common problems encountered with the existing software included a significant learning curve in form of the specific modeling languages or methodologies, complexity in the clustering and integration or required the logging features. Collecting log data during the execution of micro-services is of high importance in digital archives. Audit trails and provenance information has to be reliable and traceable to the point an object is ingested the first time.

From these results, it was concluded that developing a new workflow engine was a sustainable solution. Considering the

available resources such as developers, time and budget it was the most feasible option.

Micro-service architecture

Micro-services as an architecture or a software development concept is not a new invention. The origin goes back to UNIX design philosophy and the concept of modular software. There is no explicit definition what is a micro-service. It can be seen that a micro-service is an independent program that does only one thing but it does it very well. The architecture aims to decouple the services in a way that they can be reused, changed and combined independently from each other. It can, however, be hard to define, what the core functionality of a service is. As an example, file name normalization would be one function that could be implemented as a micro-service. The implementation can be as small as just a few lines of code, as name micro suggests. Micro-services rely on external operations to add features such as monitoring, clustering, controlling and advanced user interfaces. The services itself are small, lightweight and independent. They can be built or wrapped with programming frameworks, and usually they are called by workflow engines or software systems rather than the users. The architecture itself was a major driver for choosing open source licensing for the project deliverables.

Micro-service design philosophy

The UNIX design philosophy defines the following core rules for micro-services [2]. Each is discussed in more detail below and it is evaluated how they will benefit the digital archive development.

1. *Each program or service should do its task, and only its task, while doing it very well.*
2. *For new features, create new services. Do not create complexity by adding new features to existing services.*
3. *Expect the output of each program to become the input for another program. Even if yet unknown.*
4. *Design and build software to be tested and used early.*
5. *Do not hesitate to rebuild or remove clumsy or unusable services*.

Each program should perform its function very well and have no other functions. There are two immediate benefits: the amount of work and other resources is much easier to estimate and manage if the scope is kept at minimum; and when solving only one problem at a time, much greater effort can be put on developing a quality solution.

If new features are required, it should be a very careful decision if it should be made a new service or added to an existing one. When each service is kept small, it is easy to maintain and keep them up to date with rapidly changing technology. The quality is better when small amount of features can be put on maximum effort without danger of breaking the other component. Lots of bugs in software are generated because of feature overload.

Micro-services are designed to be chained and combined into more advanced workflows. When the programs' input and output are designed to be generic, they can be linked together in any way required regardless of how the developer designed it. This principle is especially useful when developing services in community and sharing them.

Because micro-services are by nature very small programs, they can be built fast and agile. Any software should be tested early and improved based on the test results and user feedback. The micro-service architecture helps splitting the development in smaller tasks and releasing them often without the need to update the whole product. It helps to manage risks and schedules.

Due to the previous principles, it is easy to refactor and improve a micro-service over time. It is highly encouraged to rebuild or even remove the old services in order to improve the quality and keep the system up to date. With open source development, the community can participate in improving the services and maintaining workflows.

With the above mentioned five basic principles, it can be concluded that micro-service architecture can improve both the quality and the features of software systems. It is especially suitable for industries that have limited funding or form co-operative communities.

Implementation

The workflow engine was implemented utilizing the above mentioned design philosophy. It should be mentioned that the thesis work produced an early prototype which will be further developed in the OSA project.

Java was chosen as the core technology because it is the most popular programming language and the developers are easy to find, which in turn helps to maximize the impact and future use. There was also good in-house knowledge in MAMK development team. Java is also independent of the operating system environment, even though Linux is our preferred environment.

The workflows' job persistence, scheduling and distribution were based on Quartz library. It is the de facto software for Java based scheduling and job management. JSPF library was used to create a plug-in system for introducing new micro-services to the workflow engine. The RESTful API was built with Jersey, which is the JAX-RS reference implementation. It also adds helpful features to speed up the development. The workflow system doesn't require any external services such as web servers. There is a pre-configured and embedded server called Grizzly integrated into the software.

The implementation policy was to use existing open source tools as much as possible. The development focus could be put on the identified problems instead of redeveloping something already done. The approach also made it easier to manage time and budget.

Design goals

The design goals were concluded from the problems identified during the research and evaluation phase. The workflow engine should be easy to use and deployable with minimum configurations. The learning curve should be low for new developers. No specific modeling language should be required. It should scale well to any size of software environment and workload. Clustering and distributed computing should be built-in from the beginning, not as an add-on feature. The engine should be extendible and modifiable with basic programming knowledge and it should be able to be integrated with any existing software with standard a RESTful web interface. Finally, it should be industry agnostic and suitable for any kind of content.

The results were from the beginning designed to be released as open source. Because micro-service architecture supports open

source and community driven development, it was crucial to support running services developed with any language or tools. It was achieved by designing a lightweight wrapper for local services and RESTful interface for remote services.

Extensibility

The workflow engine can be contacted, monitored and managed via a RESTful web service based API. The engine itself doesn't require user interface or manual management once started. It enables complete automation, but also management from any external software. In the OSA project, it is added to the archivist's workspace which is a web based portal for archive management and ingest tools. Also, the engine can be extended with Java by directly modifying it. The program code is well documented and commented.

Because Java was our preferred tool, the project created a small client library to be embedded in any Java based software. It provides a convenient access to API features. The OSA project will, once completed, provide a complete use case on how to exploit the engine. The API was built with Java's reference REST implementation which provides the best possible compatibility.

Benefits of open source

In the archiving industry, transparency provides better trust and independence over the continuously changing software companies. With limited funding, building the features and functions together is the only sustainable way. Micro-service architecture supports this approach very well. As an example, a software developer programs a new service to be used for a specific digital archive function. After completion, the service is released as open source. Any other digital archive can pick it up and modify it to suit their needs as well. With coordination, the services could be built based on common dialogue and the community. It would help reducing the redundancy of developments and decrease the dependency of commercial software.

Using open source licensing enforces that the solutions are available and usable even if the original developer would not be active at that time. It has been confirmed that open source improves the code quality due to various reasons such as the amount of people developing the software. There are more eyes looking for the bugs, more use cases to consider and more knowledge to apply. That is, if the community is active and the project scope matches the ecosystem. [4]

Clustering and distributed processing

Easy clustering and distributed processing were a major design goal of the workflow engine. It means that creating and managing a cluster of micro-services should be possible with very little configuration or extensive knowledge. It is an ambitious goal and it was clear that it was out of scope for the thesis based project. The development is continued during the OSA project.

Clustering and scalability

When defining a cluster, the only information needed is an identifying name for it. The engine then automatically discovers every node belonging to the named cluster. The engine then communicates with the found nodes. At this point, it will only keep track on the health status of the cluster (e.g. if there are servers offline).

The distribution and load balancing of work and services is done with Quartz. The cluster has a shared database which is used to store all the currently allocated jobs. A workflow request can be sent via the API to any of the nodes, which then stores it into the database. Any node available for performing the task will pick it up based on its response speed. If a node is slowed down by the heavy workload, it is unable to ask for more tasks. The cluster should automatically balance its performance. However, it is known that the approach may not be the optimal method for distributing heterogeneous jobs that can have varying requirements and payload. However, the current solution works for the scope of this project and it can be upgraded later. Because of the architectural choices, it does not affect the other parts of the engine and should be a good basis for improvements.

Fault tolerance and availability

Since the current version is only a prototype and not intended to be used in production environment, the maximum availability was not the top priority during development. There is a framework for creating actions, for example, if a health check between any nodes should fail. There could be added automatic actions like reporting, alerts and backup nodes.

Still, because this workflow engine is completely distributed and each node is a standalone unit, the failing of a single node does not bring offline the cluster. New nodes can be created and started any time and the size and capabilities of the cluster can be modified runtime. There is no need to pre-define the cluster in a configuration. The critical task is keeping the job store available. Because standard database is used, in our case MariaDB, it can be replicated with its own set of tools and there are no special tools required.

Future development

In the prototype distributing work is possible only at workflow level, not yet at micro-service level. Future development could add parallel processing for heavy micro-services like checksum calculations or preview generation. Micro-service level distributed processing would allow calculating the most efficient nodes for processing and achieve better throughput for high volume digital archives. Management of the cluster could be improved by exploiting tools like Apache ZooKeeper and nodes could be deployed on Apache Hadoop.

Building more sophisticated user interfaces for monitoring the workflow status and output is prioritized in OSA project. The development target is something similar to a very well done user interface in the Archivematica system. Some of the pilot cases require full automation, so the interface should support either just to monitor the activity or manage the workflows.

The groundwork is already done in the workflow engine and design choices have been made to enable future development. The timeframe in thesis works is limited and some features had to be prioritized.

Use case: simple ingest workflow

The workflow engine was tested and demonstrated with a simple, proof of a concept type, ingest workflow. The ingest

process used was for born digital and digitized materials. It is designed to be further developed in the OSA project and to be pilot tested during spring and summer 2014. As part of the thesis, the workflow mas modeled based on input from private archives such as Central Archives for Finnish Business Records (ELKA) and some private companies. The simplified model is presented in figure 1 and 2.

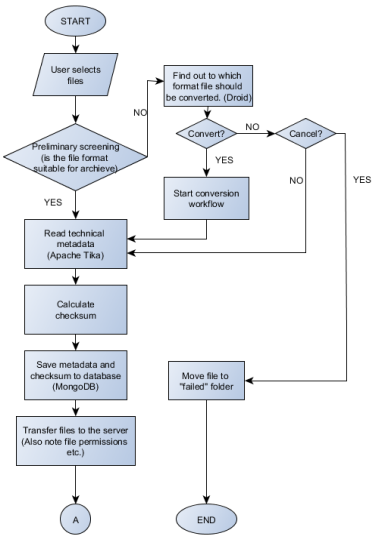

Figure 1. *A simplified presentation of the pre-ingest workflow.*

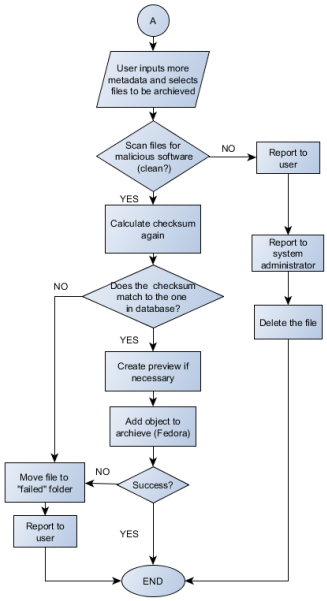

Figure 2. *A simplified presentation of the ingest workflow.*

Next, the workflow was configured for the engine. Basically, the required steps were to add a very short XML description, which included an identification, a human readable name and a description for the workflow. In addition to that, each micro-service was introduced in order that they would be executed. Defining the services has a few other parameters describing what happens in case of failure and if the service would require additional resources such as a database connection. As seen in figure 3, the configuration is very simple and straightforward. It is expected that the developer of a micro-service provided the information for the required configuration or a premade configuration fragment. In the sample configuration, a basic MongoDB document database connection is configured.

```
<workflow name="Batch-ingest" defaultAction="yes" description="Moving files to workspace" >
    <options>
        <mongoHost option="localhost" /><!-- If some of your micro-services require configurable parameters -->
        <mongoPort option="27017" /> <!-- you can provide them in XML configuration. -->
        <mongoDBName option="documents" /> <!-- In this example there are parameters for database -->
    </options>
    <task name="CleanCheck" important="true" />
    <task name="ReadMetadata" important="true" />
    <task name="SaveToMongo" important="true" />
</workflow>
```

Figure 3. *Example of the workflow and micro-service configuration.*

For each micro-service the engine requires a small handler, a wrapper class in technical terms, which enables using services implemented in any language or tools. They can be locally installed or accessed remotely via a network or Internet. The handler manages monitoring and input-output activities of the service. Adding services and their handlers does not require modifying the engine itself or restarting it. A sample of a handler is presented below in figure 4.

```
@PluginImplementation
public class NewService extends Microservice {

    @Capabilities
    public String[] caps() {
        return new String[] {"name:NewService"}; //Return the name of handler class.
    }

    @Override
    public boolean execute(String input, HashMap<String, Object> options)
            throws Exception {
        super.setState("running");  //Tell software that micro-service has started running.
        /*
         * Do what this micro-service is supposed to do, start an external program or call web service etc.
         * and wait for output.
         */
        boolean success = true;
        log(); //Perform logging function specified in super-class.

        return success; //Return whether micro-service was successful or not.
    }
}
```

Figure 4. *Example of a micro-service handler code.*

Results

The project was considered successful though still being a prototype. It has proven that all features specified in the beginning are achievable and that they were feasible decisions. The resulting software is used in OSA project pilots during spring and summer 2014. The engine will be released as open source and is used as a part of the OSA archive platform. Other deliverables are the handler code samples and the configuration samples for implementing custom workflows and services and the developer level documentation.

The thesis report about the work done in the project will be available at http://theseus.fi/handle/10024/2088 by the end of spring 2014. The source code and the complete documentation will be released by the end of the 2014 on GitHub at https://github.com/Belvain/simple-workflow-engine.

Conclusions

The prototype of the workflow engine and micro-services met the design goals set at the beginning of the project. For a production use, more development should be made. It was discovered that the extendibility and integration readiness were

already at good level. A standard based RESTful API is universal solution which supports the community driven development and digital archive systems.

Digital archives are a fine example where micro-services have proven to be useful [1]. The architecture also contributes to the risk management, budgeting and other development activities when building new digital services and tools for archives.

References

[1] Peter Van Garderen, Archivematica: using micro-services and open-source software to deliver a comprehensive digital curation solution (Artefactual Systems, New Westminster, Canada, 2010) http://www.ifs.tuwien.ac.at/dp/ipres2010%29/papers/vanGarderen28.pdf.

[2] Stephen Abrams, Patricia Hswe, Delphine Khanna, Katherine Kott, Micro-Services "It's a Series of Tubes". http://www.diglib.org/wp-content/uploads/2011/01/06micro-services.pdf

[3] James Hughes, Micro Service Architecture http://yobriefca.se/blog/2013/04/29/micro-service-architecture/

[4] Coverity Scan: 2012 Open Source Report (Coverity Inc, 2012) http://wpcme.coverity.com/wp-content/uploads/2012-Coverity-Scan-Report.pdf

[5] Kirsta Stapelfeldt, Islandora documentation https://wiki.duraspace.org/display/ISLANDORA712/Islandora

[6] Diomidis Spinellis, Code Quality: The Open Source Perspective. (Addison Wesley, 2006.)

Author Biography

Heikki Kurhinen is a software developer at Otavan Opisto. Heikki is currently studying his 4th year for BEng in information technology at Mikkeli University of Applied Sciences. He is interested in programming and all kinds of software development.

Mikko Lampi is the project lead for Open Source Archive at Mikkeli University of Applied Sciences. He has a BEng in information technology. Mikko is interested in open source, agile development and involving the community and users with the software development and digital archives.