Using Linked Data and Web APIs for Automating the Pre-processing of Medical Images

Philipp Gemmeke[1], Maria Maleshkova[1], Patrick Philipp[1], Michael Götz[2], Christian Weber[2], Benedikt Kämpgen[1], Marco Nolden[2], Klaus Maier-Hein[2], and Achim Rettinger[1]

[1] Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany
philipp.gemmeke@student.kit.edu, maria.maleshkova@kit.edu, patrick.philipp@kit.edu, kaempgen@kit.edu, rettinger@kit.edu

[2] German Cancer Research Center, Heidelberg, Germany
m.goetz@dkfz.de, ch.weber@dkfz.de, m.nolden@dkfz-heidelberg.de, k.maier-hein@dkfz-heidelberg.de

Abstract. Current developments in the health care sector are marked by the increased digitalisation of patient records, the use of electronic devices as supporting tools in patient care, and the employment of sensors (e.g. monitoring devices and surgery recording devices), which contribute directly to the abundance of medical data. However, before any significant benefits can be derived based on these growing data volumes and sources, challenges such as data format heterogeneity, distribution of the data sets, interoperability issues and basic pre-processing have to be addressed. In this paper we present an approach and a concrete architecture that support data consolidation and integration based on Linked Data principles. Furthermore, our solution enables the flexible composition and execution of data processing pipelines, based on individual processing steps, exposed through semantically described Web APIs. We demonstrate the applicability of our approach by implementing a specific scenario – Brain Tumour Progression Maps, evaluating the performance of the distributed Web API-based solution in comparison to a local execution, and determining the coverage of derived requirements.

1 Introduction

Current developments in industries and areas, which are directly influenced by IT developments, are marked by the increasing importance of data, both in terms of the growing data volumes as well as in terms of the potential in the context of analysis, trend detection, predictions, decisions support, recommendations and automated processing. Especially in the health care sector the digitalisation of patient records, the use of electronic devices as supporting tools in patient care, and the employment of sensors contribute directly to the abundance of medical data. However, before any approaches or solutions can be develop based on using the data for the grounding of decisions, it is necessary to collect, integrate and pre-process the individual datasets, which often come from distributed sources such as MRT scans, laboratory findings or results of diagnostic analysis.

The development of data-based supporting systems is hindered by a number of existing challenges. First, the used diagnostic devices and medical records software usually store data in different formats and in different locations. As a result, building solutions that employ querying data distributed over many different databases with different data formats and database schemas is very difficult. Second, the overall increase in the volume of data needs to be taken into consideration and be addressed by high-performance and scalable solutions. Furthermore, more and more applications in health care focus on distributed solutions and provide their functionalities over remotely accessible Web APIs[3]. Web APIs are based on a Web-compatible technology stack – relying on URIs for endpoint and resource identification and HTTP for communication and message exchange. Therefore, individual functionalities for identification and organisation of the available patients' information (perception), for analysing the existing information (interpretation) and for deriving treatment decisions (actions), are exposed over individual programmable interfaces. In this context, of particular interest is the composition of several Web APIs as part of new applications, which generates an increased value by enabling the composition of data and hence the extraction of new information and insights.

In order to address the problems described above, we advocate a solution based on Linked Data principles. In general, semantic technologies and Linked Data have proven to have a number of advantages in the context of data integration, interoperability and interpretation, and have already been evaluated in other medical projects (overview: [1]). Therefore, this papers presents a complete architecture based on consuming and producing Linked Data and at the same time, facilitating the direct integration of different semantic and non-semantic data sources. Furthermore, our approach is based on benefiting from the synergies of Linked Data and medical applications available through Web APIs, whose advantages have not been evaluated yet.

To this end, we describe a concrete scenario in a medical setting, depicting the challenges described above (Section 2). In the following, we present our cognitive approach based on using Web APIs and Linked Data, making the following contributions (Section 3):

1. **We use Linked Data for describing images and Web APIs, thus enabling the automatic composition of pre-processing pipelines based on distributed data sources.**
2. **We use Web APIs for remote and distributed execution of pre-processing pipelines.**
3. **We provide a prototypical implement of our Linked Data and Web API-based architecture for the pre-processing of images in brain surgery.**

[3] In 2005, the Web API repository programmableweb.com counted less than 100 registered Web APIs. However, in 2013 there were over 10 000 Web APIs registered. `http://blog.programmableweb.com/wp-content/pw-9k-growth-600x349.png`

Our evaluation is based on validating against the requirements derived from the scenario, on an implementation of the scenario and on performance tests comparing a local execution against a distributed one, based on Web APIs and Linked Data (Section 4). A short discussion and lessons learned are given in Section 5. We describe related work in Section 6 and finally conclude in Section 7.

2 Scenario – Brain Tumour Progression Maps

In this Section we present a concrete scenario in the context of the Transregional Collaborative Research Centre (TCRC) project "Cognition-Guided-Surgery"[4].The aim of this project is to create a technical system to support surgeons by the preparation, execution and postprocessing of surgeries, such as surgeries of brain cancer patients. During the diagnostic analysis and treatment of brain cancer patients a large number of different types of image data are produced. For instance Magnetic Resonance Tomography (MRT) or Computed Tomography (CT) scans are used to monitor the development of a brain tumour during treatment. In order to be able to handle this information overload effectively, more and more automatic procedures have been developed, targeted at supporting surgeons in their daily work. However, before the actual data can be used by the surgeon via automatic or interactive analysis tools, often some pre-processing steps need to be completed. These steps may take a long time and should be done automatically and in advance.

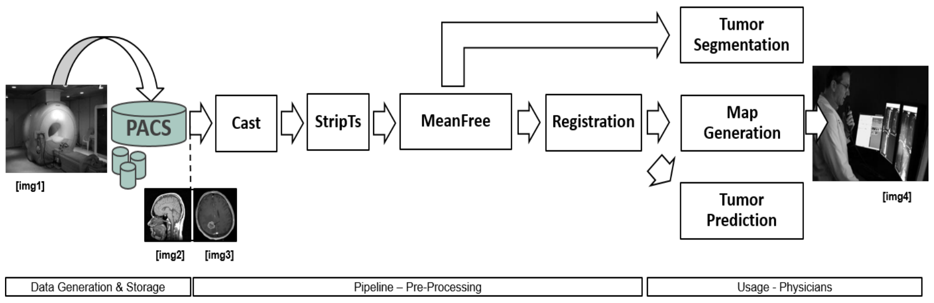


Fig. 1. Scenario – Tumour Progressions Maps

In this paper we describe exemplarily the scenario of "Brain Tumour Progression Maps" (BTPM). It is a new presentation method, which helps to monitor and evaluate the treatment progress of cancer patients by presenting images of head scans from different time intervals. The generation of BTPM consists of several steps (see Figure 1[5]) – first, the raw input images, coming from MRT and CT units, stored in a Picture Archiving and Communication Systems (PACS), are processed individually to transfer them into a common representation format ("**Cast**" step). This is necessary in order to eliminate differences in their

[4] `http://www.cognitionguidedsurgery.de/`

[5] Pictures used from wikipedia - authors: [img1] KasugaHuang, [img2] TheBrain, [img3] Marvin_101, [img4] Zackstarr

respective data types. Following is the automatic segmentation of the brain ("**StripTs**"). This is necessary in order to ensure that all the following steps are executed only on the brain data and are not influenced by other tissues, such as for example bones.

The intensities in MRT scans do not represent a physical unit and, therefore, are relative. To be able to compare different MRT scans over time a normalisation of the image has to be carried out. This is achieved during the "**MeanFree**" step. In order to be able to compare different images they need to overlay in space. Therefore in the last step of pre-processing, they are registered in relation to the same image so that a corresponding voxel represents the same area in all images ("**Registration**"). Finally, the BTPM are generated and displayed to the surgeon.

Based on the scenario and the current situation we can derive the following requirements for the development of a supporting system. The pre-processing of MRT and CT images should be executable from any workstation (**R1**). Given a centralised file storage repository, new images should be processed automatically at regular intervals (**R2**). Given a register of pre-processing interpretation algorithms, available images should automatically be pre-processed with new algorithms (**R3**).

3 Developing Medical Cognitive Applications via Linked Data and Web APIs

In this section, after we describe our approach and the architecture of the system, we demonstrate how the combination of Linked Data and Web APIs can generate increased value. Furthermore, we show how the problems pointed out in the introduction section and the requirements derived from the scenario can be adequately addressed by this combination.

3.1 Architecture

The basis of our cognitive system is a semantic knowledge base, which contains all data and information needed for supporting the previously described scenario, as well as additional practical knowledge generated through its use. In this way, the information available in the knowledge base is generated by different users or systems, and by querying this data everybody benefits from the collaboratively generated knowledge.

The semantic knowledge base consists of the following main components: 1) a central smart file storage (**XNAT**); 2) a semantic Wiki (**Surgipedia**). The interaction and the structure of our system for developing composite data-processing apps and the relationship to the presented scenario are shown in Figure 2. Data in the knowledge base is published using the Linked Data principles[6]. In this way, the interoperability between the components of the system, the integration of new data into the knowledge base and the ability to interpret this data is realised and guaranteed.

[6] `http://www.w3.org/DesignIssues/LinkedData.html`

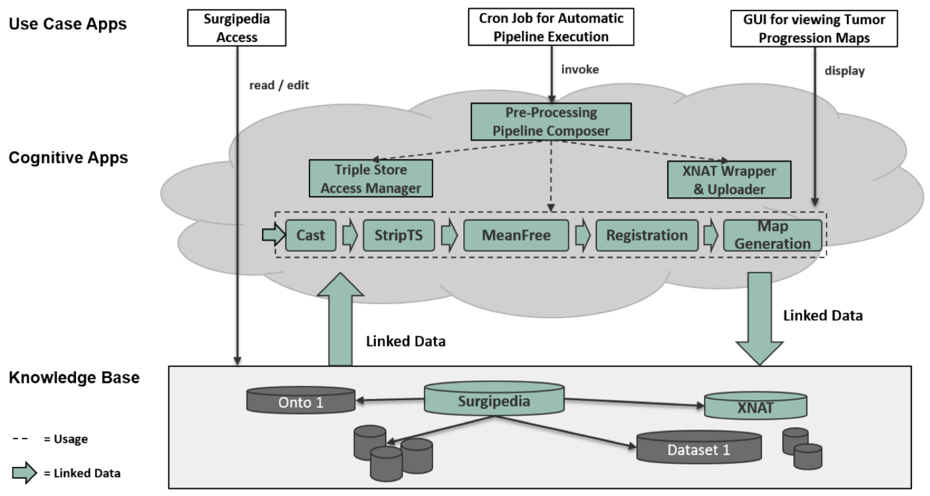


Fig. 2. Architecture – Developing Composite Cognitive Apps

XNAT as central file storage plays an important role. It is used in the knowledge base to store data generated by users and other systems, and make this data accessible within the knowledge base. One of the most important aspects of XNAT is the possibility to store files and data, which are not only Linked Data, but can be described with a free text field, which is used to describe the respective image semantically. Therefore, XNAT acts as the first point of contact to integrate data into the knowledge base. Furthermore, XNAT provides an API for getting information about the stored files and uploading new files.

Surgipedia is the data hub in the system and allows modelling metainformation and linking it to all knowledge base relevant data instances. For example, Surgipedia contains links to files stored in XNAT or other external data sources. Furthermore, it provides support for defining and managing taxonomies and instances, which are relevant for the whole scope of the system. Surgipedia supports the exporting of all the stored data as Linked Data automatically. Therefore, the interaction with the other components is facilitated.

In addition to the knowledge base, there are further components, which belong to our system. These components are called **Cognitive Apps** and **Use Case Apps**. The communication and interaction between the knowledge base and Cognitive Apps is based on Linked Data. Therefore, a Cognitive App is a Linked Data consuming and producing web application, which benefit from the knowledge base and implements a data-processing application, which feeds its results back into the knowledge base. Use Case Apps are applications designed for end-user interaction or simple execution without application logic.

3.2 Cognitive Apps

Cognitive Apps are Web APIs, which also have a semantic description based on Linked Data principles. In general, Web APIs provide a uniform interface for remote access, independently from their underlying software architecture. By adding a semantic machine-readable service description to each API, it is possible to discover each API within the knowledge base and to compose the APIs to complex pipelines or workflows. Furthermore, a semantic description allows specifying exactly the relationship between inputs and outputs for each API invocation. Technically Cognitive Apps are mainly wrappers of medical interpretation algorithms, as part of the technical system within the Cognition-Guided-Surgery project.

By applying our system to the scenario – creating Cognitive Apps for all steps of the scenario and using the Pre-Processing Pipeline Composer, we solve all problems mentioned in the introduction section and fulfil all requirements derived from the scenario. Furthermore, in order to guarantee the integration and interoperability with the knowledge base, every step has a Linked Data description and consumes and produces Linked Data.

3.3 The Image Pre-processing Pipeline

The processing pipeline consists of the following apps: Cast, StripTs, MeanFree, Registration (Section 2). After Registration, the pre-processed data is used to generate Brain Tumour Progression Maps. We show how the Cognitive Apps participate in the execution of the pipeline based on the MeanFree app. For all other steps/apps, the approaches and results are applicable.

In order to be able to use a cognitive app, it is important to make it discoverable and to provide all information needed for its invocation. This is realised by registering a new Cognitive App in Surgipedia. Each Cognitive App comes along with a semantic and machine-interpretable Linked Data description based on the LAPIS/LIDS approach [7, 5, 6]. Every description contains information about the API – for instance what the service does and how the input and output look like. An excerpt from the description of the MeanFree Cognitive App is shown in Figure 3.

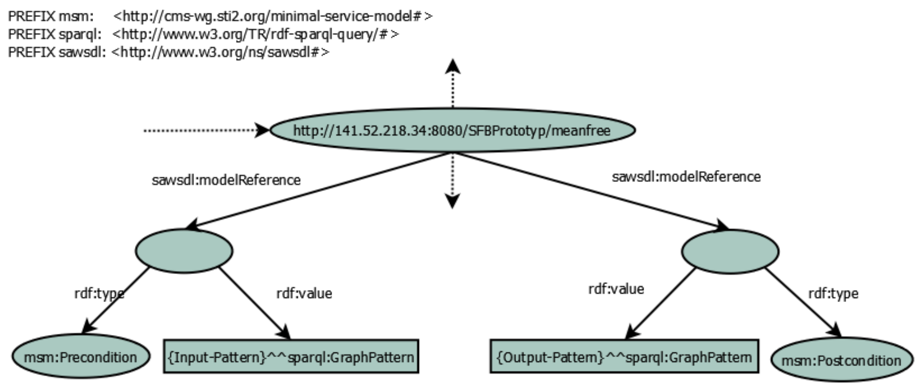
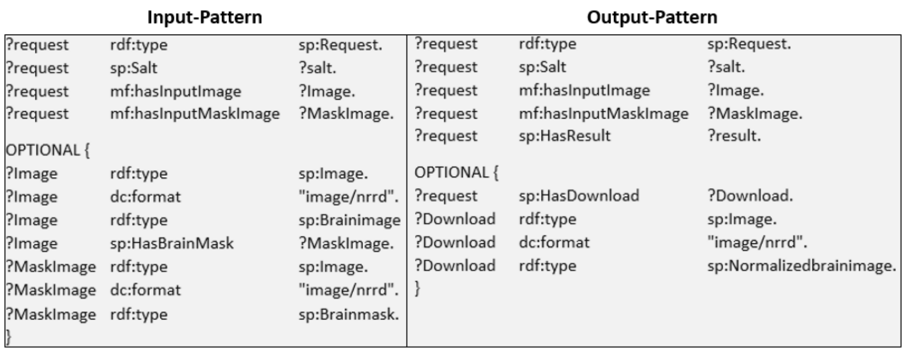

PREFIX msm: <http://cms-wg.sti2.org/minimal-service-model#>
PREFIX sparql: <http://www.w3.org/TR/rdf-sparql-query/#>
PREFIX sawsdl: <http://www.w3.org/ns/sawsdl#>
http://141.52.218.34:8080/SFBPrototyp/meanfree
sawsdl:modelReference
sawsdl:modelReference
rdf:type
rdf:value
rdf:value
rdf:type
msm:Precondition
{Input-Pattern}^^sparql:GraphPattern
{Output-Pattern}^^sparql:GraphPattern
msm:Postcondition

Fig. 3. Description of MeanFree Cognitive App as RDF graph

Through the specification of resources of the type msm:Precondition and msm:Postcondition the service contains SPARQL graph patterns, which describe the input and output of the service, based on the input RDF that it expects and the output RDF that it produces (see pattern in Figure 4). With this information, it is possible to execute the input SPARQL pattern on given Linked Data and verify whether the service is runnable on the available data.



Input-Pattern

```
?request     rdf:type                 sp:Request.
?request     sp:Salt                  ?salt.
?request     mf:hasInputImage         ?Image.
?request     mf:hasInputMaskImage     ?MaskImage.
OPTIONAL {
?Image       rdf:type                 sp:Image.
?Image       dc:format                "image/nrrd".
?Image       rdf:type                 sp:Brainimage
?Image       sp:HasBrainMask          ?MaskImage.
?MaskImage   rdf:type                 sp:Image.
?MaskImage   dc:format                "image/nrrd".
?MaskImage   rdf:type                 sp:Brainmask.
}
```

Output-Pattern

```
?request     rdf:type                 sp:Request.
?request     sp:Salt                  ?salt.
?request     mf:hasInputImage         ?Image.
?request     mf:hasInputMaskImage     ?MaskImage.
?request     sp:HasResult             ?result.
OPTIONAL {
?request     sp:HasDownload           ?Download.
?Download    rdf:type                 sp:Image.
?Download    dc:format                "image/nrrd".
?Download    rdf:type                 sp:Normalizedbrainimage.
}
```

Fig. 4. SPARQL Input/Output Pattern for Invocation in Turtle

As visualised in the Input-Pattern in Figure 4, a service request to MeanFree must contain several parameters. It is required that the request contains as input a request (sp:Request). Furthermore, a request must contain a random string called “salt”, used by the requester to identify his/her request and download the results after the service has finished its processing. Additionally, it is required to specify the URI of the input image (a head scan image of a patient), on which the service performs its processing operations. The last required parameter is a URI of a brain mask used to recognise the shape of the brain. A service request results in a service response (according to the Output-Pattern). The response contains all information of the request and further information about the result – either “Success” or “Failure”. Additionally the response also contains a download link of the result file, and specifies that the result file has the format “image/nrrd” and is a normalised tissue colour head scan.

The **Pre-Processing Pipeline Composer** is also a Web API and uses the Scenario and general Cognitive Apps to automatically pre-process available MRT and CT scans of brain cancer patients within the knowledge base by generating and executing HTTP requests. To realise this, the Composer consists of three phases and is started by performing a HTTP GET on its base URI[7]. In the first phase (Ramp Up), all information needed for identifying matches are collected. Via the XNAT-Wrapper, all data regarding the available files in XNAT and their semantic description in the free text field is crawled and stored in an own triple store. Additionally, the Composers queries the Triple Store Access Manager to get a list of Cognitive Apps registered in Surgipedia, and each retrieved Cognitive App for its respective semantic description – especially the SPARQL pattern. In the second phase (Test Phase), the Composer tests for each identified image and Cognitive App, whether the Cognitive App is applicable for

[7] `http://host:8080/Composer/check/`

the image. To this end, the Composer generates a SPARQL Construct Query using the input pattern of a given Cognitive App and executes this on the semantic description of the given image, stored in the triple store. If a semantic description of a head scan matches a SPARQL input pattern and this match has not been processed already, the Composer will generate a RDF+XML request file and a HTTP request for the identified matches automatically. In the last phase (Execution), the Composer executes automatically all identified matches by executing the respective HTTP requests. Furthermore in case of a successfully executed request, additional commands are executed to upload the result file(s) to XNAT and describe it semantically, thus enriching the knowledge base. As a consequence, newly uploaded files will be processed automatically in the next execution round of the Composer as long as matches are found.

4 Implementation and Evaluation

We successfully implemented the presented knowledge base, the Composer and all steps of the pre-processing pipeline. The scenario steps are wrapped as Web APIs, based on the Java API for RESTful Web Services (JAX-RS) implementation Jersey[8], extended by a semantic service description and run on an Apache Tomcat 6 web server. In detail, the wrapping of a step means that a via HTTP POST submitted RDF+XML request file is processed using the respective SPARQL input pattern. Afterwards, the underlying MITK command line tool is executed locally on the web server using the parameters extracted from the request file. In the end, a RDF+XML response file according to the respective SPARQL output pattern is generated, and the result file is copied to the download folder on the web server, which is accessible by using the specified "salt". By using the user provided identifier "salt" it is easy to identify the result file on the web server, given a RDF+XML request file. In all steps, the open source framework Jena[9] is used to process Linked Data and execute SPARQL pattern.

The Composer Cognitive App is also a Web API based on Jersey. By defining a cron job for performing a HTTP GET on its base URI as Use Case App, the Composer is executed automatically and at regular intervals. As mentioned in the approach section, the composer creates and executes HTTP requests for identified matches of MRT/CT images and Cognitive Apps.

Performance Evaluation

Now we describe the performance difference between a manually composed local execution of a pipeline, with the scenario steps as command line tools (experiment 1: **LOCAL**), a manually composed remote pipeline, with the steps as Cognitive Apps (experiment 2: **REMOTE**) and the automatic usage of the Cognitive Apps by the Composer (experiment 3: **COMPOSER**). To evaluate the performance of each experiment, both the respective command line tools and the realised Cognitive Apps run on the same virtual machine (following: Cognitive VM) – with QEMU Virtual CPU version 0.9.1 with 2266.796 CPU

[8] `https://jersey.java.net/`
[9] `http://jena.apache.org/`

MHz and 4GB memory. The Pre-Processing Pipeline Composer runs on a different machine (following: Composer VM) – Intel Core i7 860 with 2800 Mhz and 8GB memory – and communicates with all components (knowledge base, General Cognitive Apps, Scenario Cognitive Apps) over the Web.

For every experiment we used real MRT images of a head of a patient, a brain atlas mask and a brain atlas image (needed for stripping the brain). As mentioned before, we only implemented the pre-processing steps of the scenario. So the final result of this four step pipeline are stripped and normalised MRT head scans. At the beginning of each experiment, the test images are stored on the client's local disc. This is also the end criteria for each experiment, the overall result images have to be stored on the client's local disc, too. To verify that the results of each test case are the same, we checked for each result file the size and additionally opened it with MITK to check it visually.

Experiment 1: All steps were performed by a hard-coded java script of their respective command line calls and executed on the Cognitive VM. All files are loaded and stored from local disk of the Cognitive VM.

Experiment 2: At first, a hard-coded java script uploaded the test images from a client's local disc to XNAT and afterwards the same script executed requests of the Cognitive Apps remotely on the client. All temporay generated files were loaded directly from the web server of a Cognitive App, since the result files are identificable via "salt" without parsing. Finally, the result files were downloaded from the web server of the Cognitive VM to the client's local disc.

Experiment 3: Again, at first the test images were uploaded to XNAT from a client by a java script. Afterwards the script performs a HTTP get on the Composer's base URI to start processing the test images. The test images were processed automatically step by step in each execution round by the Composer. At the end of execution, the client downloaded the final results from XNAT to the local disc.

Experiment	Duration of Pipeline Execution (mean) in s	Difference of Duration in %	Valid result
LOCAL	585.3	-	Yes
REMOTE	689.5	+17.8	Yes
COMPOSER	900.06	+53.78	Yes

Table 1. Evaluation results of each experiment.

As it can be seen (Table 1), all experiments and their respective execution style, produce the same output files. However, the motivation of the scenario was also to perform the pre-processing of files in advance (not on demand), because the pre-processing of files is time-consuming. By looking at the durations, the combination of Cognitive Apps and knowledge base for pre-processing the files takes much longer than the old execution style with just command line tools. The main difference is caused by uploading and downloading the needed files to XNAT. Therefore, Table 2 shows the recalculated durations, considering the uploading and downloading of files with an average upload speed of 255 KB/s and download speed of 1891 KB/s[10].

[10] Duration of file transfer = File size [MB] / Average Speed [KB/s]

Experiment	Duration of Pipeline Execution (mean) in s	Duration of file Upload in s	Duration of File Download in s	Recalculated Duration in s	Difference of Duration in %
LOCAL	585.3	-	-	585.3	-
REMOTE	689.5	71.08	2.49	615.93	+5.23
COMPOSER	900.06	192.35	18.84	688.87	+17.70

Table 2. Evaluation results of each test case considering file upload and download.

5 Discussion and Lessons Learned

In this section, we discuss whether or not all derived requirements of the scenario have been fulfilled. As shown before, by comparing the respective result files of each of the experiments, providing the command line tools of each scenario step as Cognitive App allows for pre-processing MRT and CT images from any workstation (**R1**). In addition to that **R2** is fulfilled by defining a Cron entry, which executes the Composer regularly and automatically for images stored in the central file storage XNAT. After equipping a new Cognitive App with a semantic service description (especially with SPARQL graph pattern for input and output) and registering it in Surgipedia, the Composer will automatically notice this new Cognitive App and use the service description to check whether or not images in XNAT match, therefore the **R3** is also fulfilled.

As a consequence, wrapping the command line tools as Web APIs allows for providing their functionality remotely and causes only a small additional performance overhead. Furthermore, by using the Composer in combination with the knowledge base and the Cognitive Apps for every step of the scenario, we implemented a system that solves all mentioned problems and fulfils all requirements, with only a slight time delay.

As a result of implementing each scenario step as loosely coupled Web API, we provide Cognitive Apps, which have a lightweight and uniform interface, and are scalable to handle the continuously growing data volumes. By using Linked Data in our approach, it is guaranteed that data is accessible in a standardised way. Furthermore, it allows that new data within the knowledge base can be easily integrated by using a Linked Data-based crawler. By using Surgipedia as a data hub and core of the knowledge base, we are able to combine all components and their respective information and query those over the Triple Store Access Manager.

6 Related Work

There are already some approaches for realising medical image processing command line tools as Web APIs/Web services and combining them to workflows or pipelines: the Taverna[11] plugin for GIMIAS[12]. GIMIAS is a software framework

[11] `http://www.taverna.org.uk/`
[12] `http://www.gimias.org`

for building applications in a medical setting, i.e. applications for image processing, analysis of images or simulations. Thereby, GIMIAS is kind of a workflow-oriented environment, which allows composing applications to workflows and extending the functionality of a given GIMIAS installation by adding GIMIAS command line tools [3]. In addition, there is a Web service plugin for GIMIAS, which allows providing these command line tools as SOAP/WSDL (WS-*)[13] Web services or RESTful Web services. Moreover, the Taverna plugin[14] for GIMIAS allows building and executing workflows for distributed runtime environments by composing available web services using drag and drop. Consequently, a user can combine local GIMIAS applications and remote command line tools, which are provided as Web services [3]. In general, the Taverna Workbench allows to create, execute, monitor and share workflows that use local and remote third-party applications. It uses several techniques for service discovery, such as registries and shallow semantic description based on OWL for selecting appropriate web services [4]. In contrast to our approach, Taverna (respectively the Taverna Plugin for GIMIAS) enables the manual creation or modification of existing workflows of medical image processing applications, supported by a graphical user interface. Using our approach with a semantic knowledge base and Linked Data prosuming Web APIs according to SPARQL pattern, pre-processing pipelines of wrapped MITK command line tools are created and executed automatically and stepwise. In addition to these two different creation styles of workflows/pipelines, there are other approaches such as using templates as in the Semantic Workflow Approach [2].

In general, realising applications as loosely coupled Web services allows to build powerful applications, which can use Web services that are not explicitly specified at design time. Thereby, the semantic description of each Web service and a possibility to register and query already registered Web services, are the keys to realising scalable and flexible systems.

7 Conclusions

Current developments in the medical and health care domains are characterised by the increased use and importance of digitalised patient records, monitoring devices and medical decision supporting systems, which lead to an increase in the volumes of available data. However, before any significant benefits can be derived based on this data abundance, challenges such as data format heterogeneity, distribution of the data sets, interoperability issues and basic pre-processing have to be addressed. In this paper, we presented an approach and a concrete architecture that supports data consolidation and integration based on Linked Data and semantic technologies. Furthermore, our solution enables the flexible composition and execution of data processing pipelines, based on individual processing steps in the form of Cognitive Apps exposed through Web APIs, with semantically described interfaces. We demonstrate the applicability of our approach by implementing a specific scenario, evaluating the performance of the

[13] `http://www.w3.org/TR/ws-arch/#technology`
[14] `https://code.google.com/p/taverna-plugin-for-gimias/`

distributed components in comparison to a local execution, and determining the coverage of the derived requirements. As part of future work, we will focus on handling command line tools, which hold state changes during their execution. Realising those as Web APIs requires a kind of enclosure and initialisation in a first step and, therefore, additional application logic and approaches. Finally, there is a challenge regarding how to handle data, which matches only partially a given SPARQL graph pattern. This would require using approaches employing some fuzzy logic.

References

1. Cheung, K.H., Prud'hommeaux, E., Wang, Y., Stephens, S.: Semantic web for health care and life sciences: a review of the state of the art. Briefings in Bioinformatics 10(2), 111–113 (2009)
2. Gil, Y., Gonzlez-Calero, P.A., Kim, J., Moody, J., Ratnakar, V.: A semantic framework for automatic generation of computational workflows using distributed data and component catalogues. J. Exp. Theor. Artif. Intell. 23(4), 389–467 (2011), `http://dblp.uni-trier.de/db/journals/jetai/jetai23.html#GilGKMR11`
3. Hunter, P., Bradley, C., Britten(et al.), R.: The vph-physiome project: Standards, tools and databases for multi-scale physiological modelling. In: Ambrosi, D., Quarteroni, A., Rozza, G. (eds.) Modeling of Physiological Flows, MS&A Modeling, Simulation and Applications, vol. 5, pp. 205–250. Springer Milan (2012)
4. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: Lessons in creating a workflow environment for the life sciences: Research articles. Concurr. Comput. : Pract. Exper. 18(10), 1067–1100 (Aug 2006), `http://dx.doi.org/10.1002/cpe.v18:10`
5. Speiser, S., Harth, A.: Taking the lids off data silos. In: Proceedings of the 6th International Conference on Semantic Systems. pp. 44:1–44:4. I-SEMANTICS '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1839707.1839761`
6. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: ESWC 2011. Lecture Notes in Computer Science, vol. 6643, pp. 170–184. Springer (2011)
7. Stadtmüller, S., Norton, B.: Scalable discovery of linked apis. Int. J. of Metadata, Semantics and Ontologies 8(2), 95–105 (2013)