OpenKnowledge
FP6-027253

OpenKnowledge Multimedia Applications

David Dupplaw, Antonis Loizou, Madalina Croitoru, Srinandan Dasmahapatra, Bo Hu, Paul Lewis, Kostantinos Papalias, Nigel Shadbolt, Liang Xiao

1 IAM Group, School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK.
OpenKnowledge Multimedia Applications

David Dupplaw  Antonis Loizou  Madalina Croitoru  
Srinandan Dasmahapatra  Bo Hu  Paul Lewis  
Kostantinos Papalias  Nigel Shadbolt  Liang Xiao  

October 22, 2008

Abstract

The direction of development of the OpenKnowledge kernel has meant that the provision of multimedia data within OpenKnowledge applications can be accommodated effectively without the overhead of extra multimedia-specific functionality in the kernel itself. In this deliverable we present two multimedia applications that have been built upon the OpenKnowledge kernel that show how multimedia data can be manipulated in the context of the OpenKnowledge system while also providing case studies demonstrating the ease with which legacy applications can be made peer-to-peer using the OpenKnowledge kernel.

1 Introduction

OpenKnowledge provides a peer-to-peer knowledge sharing environment in which disparate parties can participate in interactions which execute well-defined protocols. The implementation of the OpenKnowledge idea, the ‘OpenKnowledge kernel’, gives developers a fast route to providing peer-to-peer realisations of their own applications. In this paper we demonstrate the kernel in the context of two very different multimedia applications to show that it is not limited to text-based data, whilst also showing that the introduction of a peer-to-peer backbone on web-based applications can provide advantages both for the developer and for general users.

The multimedia applications both build towards the idea of low-level image descriptors being automatically mapped to high-level semantic descriptors. Furthermore, applications deployed on our open system will improve in accuracy as techniques for multimedia analysis become more accurate.

The applications that we have used to describe the advantages of the peer-to-peer approach have very different audiences.

- The Semantic Logger application utilises the co-occurrence of data within a user’s life-log to provide semantic annotations for objects for which these annotations may not be easily available, rendering their retrieval and/or further processing impossible. Frequently occurring co-occurrences can be
shared to provide such annotations to the whole community of users of the Semantic Logger. To this end, the logger not only provides a personal logging service, but also provides a community-based annotation service. By making the Semantic Logger peer-to-peer the analysers that could be used upon the log data to generate the annotations are not restricted in access and therefore in scope.

- MIAKT is a tool developed to provide knowledge-based distributed decision support of breast cancer diagnoses. It provides a portal through which the various parties involved in gathering data from a patient can store their data. This provides a single-point through which all of the patient’s data can be evaluated, much as is done in person at a Multidisciplinary Meeting (MDM). The audience for the MDM is clearly limited to the doctors involved with a patient’s primary care, although the data which MIAKT gathers could be of interest to more parties, including the patient. A patient-centric model, where patients are responsible for their own data, integrates well with a peer-to-peer paradigm. This involves trusted interactions between peers. It also links well to the Semantic Logger project, as a patient’s healthcare data is just further data about a person that could be stored alongside other personal data in a life-log.

2 Motivation

In recent years the way in which the web is being used has changed. Initially the idea behind the web was to share text-based documents, but the plummeting costs of digital and video cameras, as well as the widespread accessibility of broadband internet, has meant that non-text data is becoming a normal way for users to communicate messages via the web. In November 2007, Flickr, a large photo sharing website, contained over 2 billion photos [4] and YouTube, a large video sharing website, serves over 100 million videos every day [11]. Finding some particular non-text data on the web is getting more difficult; recall may be high but precision is very low.

Although both Flickr and YouTube provide means for tagging photos with simple words or phrases, neither provide automated means for making the tags. This means that the tags are arbitrary; they can be in any language or refer to any part of the given media. YouTube does provide methods for adding annotations of various forms between specific time segments of a video stream but these annotations are for visual augmentation, not for aiding a search. Retrieval of multimedia objects is dealt with in both MIAKT and the Semantic Logger. In this document we describe how we use the OpenKnowledge system to deal with the automatic annotation of images, albeit in very different domains.

One of the important functionalities developed in MIAKT is the difficult task of annotating x-ray mammograms, taken during a patient’s initial screening as well as throughout their care process. Due to the expertise required to initially identify suspicious regions within a mammogram, the MIAKT system provides
a user interface that allows doctors to examine high-resolution digital mammograms and identify suspicious regions. MIAKT then uses image analysis tools to attempt to classify the region highlighted. The classifiers are currently fairly limited, but as research continues in this domain, and by using the OpenKnowledge network, the accuracy and range of functionality of the classifications can greatly increase.

The Semantic Logger covers a broad domain and, as such, we have implemented only very high-level, easy-win analysers and classifiers, such as face detection, EXIF analysis and artificial/natural detection. The Semantic Logger will benefit greatly from an open network where new analysis tools and classifiers can be made available, and because it deals with disparate data and personal data, it appeals to the mash-up community that will often come up with new uses for services that would have otherwise not been considered.

By porting these two applications to the OpenKnowledge system, we show that the OpenKnowledge system is capable software that can be used as a basis for very different applications that utilise multimedia data, while at the same time showing that our engineering of the system has meant that porting legacy applications to a peer-to-peer system can be easily achieved using the kernel.

3 Semantic Logger

The Semantic Logger (SL) is the outcome of previous work looking into automatic, auto-biographical metadata acquisition, published in [13]. The system builds on the ideas brought forward by the Semantic Web vision, with a particular focus on the notion of assembling, and integrating web accessible resources.

The Semantic Logger captures various aspects of its users digital activity, as
presented in Figure 1. Specialised components have been developed to convert each type of raw data into RDF, which is then stored in a private Knowledge Base (KB). The system has been developed to serve as a platform for the deployment of Semantic Web services exploiting the integrated personal metadata. The SL has now been ported to the OpenKnowledge system. The integration of the two systems is seen as illustrative of how legacy systems can be utilised from within the OpenKnowledge framework. The modularised nature of SLs design lends itself well to the p2p paradigm. To this end, the system has been encapsulated in three roles: user, server, and propagator. The server OpenKnowledge Component (OKC) is run on a separate peer, responsible for maintaining SL KBs on behalf of peers running the user OKC. User OKCs are mainly responsible for obtaining input from the user, and displaying the server’s responses. Peers in the user role also periodically subscribe to the propagator one, to upload gathered information to the server. The following interactions (each with an associated interaction model) provide the functionality of the system:

Register

The user is prompted to enter his/her credentials, and is guided through an interactive process to create (or import) a Web accessible FOAF (Friend-of-a-Friend) file. The resulting document will serve as the user’s URI in the SL system.

Configure

The user is presented with the various propagators made available by the server. Upon selection the server returns URLs for the propagator binaries, which are downloaded from the Internet (after the user grants permission) and installed in a predetermined directory. The code for each propagator is accompanied by a configuration file containing invocation details, used by the propagator OKC.

Query

The user is presented with a form to query the KBs they have access to, in SPARQL. Once the server resolves the query, the results are sent back to the user and displayed.

Upload

The user is allowed to select a variety of ways to upload RDF to the server, and receives a notification of the outcome of the import.

Propagate

When the user subscribes to the propagator role, the corresponding OKC performs a search in the SL installation directory for installed propagators. These are then run, and the RDF generated is uploaded to the server. This interaction is ‘silent’ in that it requires no inputs from the user. This is so the interaction can be run periodically, when the user could potentially not be there. As such, the user’s credentials are encrypted and stored in a different directory.
While in the past the user had to explicitly subscribe to each specific interaction to perform its associated function, as the kernel reaches maturity we have been able to provide an overarching interaction model that encapsulates the entire functionality. This is achieved through deferred interaction model execution: the act of subscribing to another interaction in order to satisfy a constraint in a previous one. This process was described previously in Deliverable 5.4. In this recursive interaction model, shown in Model 1, the user is asked to select the interaction they would like to subscribe to.

\[
\begin{align*}
a(\text{semlogger}_\text{server}, ID1) :: &
\quad \text{get options}(O) \Leftrightarrow a(\text{semlogger}_\text{user}, ID2) \text{ then } \\
&\quad \text{options}(O) \Rightarrow a(\text{semlogger}_\text{user}, ID2) \text{ then } \\
&\quad \text{← build options form}(0) \text{ then } \\
&\quad \text{selection}(O) \Leftrightarrow a(\text{semlogger}_\text{user}, ID2) \text{ then } \\
&\quad \text{interaction}(I) \Rightarrow a(\text{semlogger}_\text{user}, ID2) \text{ then } \\
&\quad \text{← getInteraction}(O, I) \text{ then } \\
&\quad a(\text{semlogger}_\text{server}, ID1)
\end{align*}
\]

\[
\begin{align*}
a(\text{semlogger}_\text{user}, ID2) :: &
\quad \text{get options}(O) \Rightarrow a(\text{semlogger}_\text{server}, ID1) \text{ then } \\
&\quad \text{options}(O) \Leftrightarrow a(\text{semlogger}_\text{server}, ID1) \text{ then } \\
&\quad \text{selection}(O) \Rightarrow a(\text{semlogger}_\text{server}, ID1) \\
&\quad \text{← getSelection}(0) \text{ then } \\
&\quad \text{interaction}(I) \Leftrightarrow a(\text{semlogger}_\text{server}, ID1) \text{ then } \\
&\quad \text{← runInteraction}(I) \text{ then } \\
&\quad a(\text{semlogger}_\text{user}, ID2)
\end{align*}
\]

By porting the SL to the OpenKnowledge platform the task of including additional propagators, developed by third parties becomes easy. Once a propagator has been developed and binaries made available on the web, all the creator would have to do is subscribe to the server role in the configuration interaction model and provide a modified list of propagators. The new list may include the original propagators or choose to replace them. The trust layer of OK will then eventually guide the choice of peers.

For similar reasons the deployment of services that exploit the knowledge gathered in a Semantic Log is also made significantly easier. The following section details Photocopain, one such service in the context of image annotation.

### 3.1 Photocopain

The rapid growth in digital visual information has made image annotation an active research area. The automatic assignment of keywords, tags or other annotations is seen as a key step in making such items more widely accessible and in Southampton we have continuing research on the topic [6, 7, 12, 5].
Photocopain [14] is an image annotation system that uses a selection of global and local image analysis techniques and classifiers, in combination with the knowledge gathered by the Semantic Logger to produce high–level annotations for images. Originally, the various classifiers were integrated in a single script, that first combined the features extracted to infer annotations, and uploaded RDF representations of them to the KB. Then, provided the camera recorded the time the picture was taken, the script would query the KB for events that occurred at the same time. Specific types of event (such as calendar entries, see Section 3.2 for a list) can be processed to provide annotations for the image. Photocopain stands much to gain from being deployed on a platform such as OK. In the same manner as explained earlier for propagators, different annotators can be implemented by various peers. In this way Photocopain becomes a dynamically extendable system. However, some annotators may depend on information created by others and have to be run in a specific order. Others will require additional information to be provided by the user. As such, a generic annotator role could not be defined. Instead, the Interaction Model that defines the annotation process contains a role for each type of annotation. For this reason, peers who would like to add annotators to the system would have to modify the interaction model, in addition to creating the corresponding OKC. In time, interaction models with annotators that produce accurate annotations will be selected over others, as a consequence of propagating trust values. A fourth role, annotator, has been developed to accommodate Photocopain in OK,
along with six interaction models:

Login and select action
The user is prompted for their credentials by the server, and asked to select one of the two following interaction models. The user’s login information is saved for use in future interactions. This model showcases another new feature of the kernel, the ability to save variables inside the peer’s state.

Upload photo
As annotators access the image data through the Web, the image must first be uploaded to a public URL. This interaction model describes the process of publishing a photo, and receiving its new URL in return.

Manage photos
This interaction model allows the user to choose further interactions using photos they have already uploaded. They are described below.

View photo
The user selects a photo which is then visualised, along with any associated annotations.

Manually add annotation
The user is allowed to add an arbitrary annotation using the <rdfs:label> or <tagora:tag> properties.

Annotate automatically
This interaction defines the order in which the individual annotators will be contacted. Annotators send proposed annotations back to the user, which is then offered a choice as to whether to upload them to the server. Figure 3 presents an overview of this interaction, as shown by the OK diagnostics panel after its successful execution.

3.1.1 Annotators
This section details the various annotators provided by the Photocopain system.

EXIF information extraction
EXIF data from an image provides useful values, such as the time and date of capture, details about the camera and capture conditions such as whether the flash went off, and the type of lens the photo was taken with. The EXIF annotator can provide the following annotations where appropriate:

- <image_url> <photocopain:timeOfCreation> <UTC_date>:
The time the picture was taken, in milliseconds since the epoch.
Figure 3: Screenshot of the OK diagnostics visualiser after the automatic annotation interaction has been completed.
Images with very high hyper–focal distance are considered to be in the bokeh style, as most of the image is out of focus.

Images with very low hyper–focal distance have high depth of focus and most of the image is in focus.

The focal length.

The size of the object in focus.

The object in focus is very far from the lens.

The object in focus is far from the lens.

The object in focus is at an average distance from the lens.

The object in focus is close to the lens.

The object in focus is very close to the lens.

The aperture.

Face detection

The face detection algorithm uses colour coherence to detect skin–coloured oval regions in the image. It can provide the following types of annotations:

A single face has been detected, covering at least 10% of the image.

Two faces have been detected, each covering at least 10% of the image.

Three or more faces have been detected, each covering at least 10% of the image.

A single face that covers at least 30% of the image has been detected, and the focal length is at least 12 times greater than the aperture. This annotation depends on the EXIF annotator having extracted the focal length from the image.
Calendar information

The calendar annotator queries the user’s KB for calendar entries that span over the time the picture was created. As such it must be run after the EXIF annotator, who will provide the picture’s time of creation. Once an entry is obtained, its summary and location fields are used to provide the following annotations, respectively:

- `<image_url> <photocopain:event> <ical:summary>`
- `<image_url> <photocopain:location> <ical:location>`

GPS information

Similar to the previous one, the GPS annotator queries the KB for any GPS track point recorder within 15 minutes of when the picture was taken. If one is found, the picture is annotated using the following triples:

- `<image_url> <geo:long> <FLOAT>`
- `<image_url> <geo:lat> <FLOAT>`
- `<image_url> <geo:alt> <FLOAT>`

3.2 Semantic Logger Summary

By porting the Semantic Logger and the Photocopain system to the OpenKnowledge network, we have shown a number of things about both our legacy applications and the OpenKnowledge system:

- The interaction deferral mechanism had not, until the development of this application, been demonstrated. This application makes use of the interaction deferral mechanism for modularising the functionality of the logger.

- The visualisation mechanism has been used extensively for producing forms and small windows that are specific to the interactions. This avoids the development cost of creating a specific, monolithic user interface for the semantic logger. Of course, this has the disadvantage that there could be a lack of consistency between user interactions, but it shows that visualisations can be created simply and that application prototypes can be completed quickly.

- Photocopain is shown as an exemplar of how multimedia annotation can be carried out within the OpenKnowledge network. It is clear that as the community of developers creating annotators increases, the accuracy and range of possible annotations that can be created for multimedia objects will grow. Indeed, as the base of annotators grows, new annotators can be created simply through chaining interactions, and the logic that chains these annotation processes can easily be formulated in LCC without the need for large development overheads.
The OpenKnowledge kernel provides all the tools necessary for creating a peer-to-peer system from a legacy system. The use of the various tools for subscribing to, executing and deferring interactions have all been shown with our example, creating an application that would otherwise have required development of a complex communication module. The application was successfully ported with approximately two weeks of development time.

4 MIAKT

The Medical Images and Advanced Knowledge Technologies project, that ran between 2002 and 2004, developed a general knowledge browsing interface with specific plugins that provided for the manipulation of medical images [3]. The project was focussed on the support of a meeting called the Multidisciplinary Meeting (MDM) where medical experts from different fields of medicine would come together in a collaborative meeting to make decisions about the necessary actions to take upon breast cancer patients [2]. The different fields of medicine represented in the meeting have different data needs and expertise; for example, the radiologists are experts in analysing x-ray mammograms, pathologists are experts in the understanding of microscopic images of biopsied tissue, clinicians interact with the patient and handle their patient records, oncologists determine the best course of treatment to maximise a patient’s life span, while surgeons need to understand location and extent of the recommended excision, given all of the above. All this data is stored against a patient’s record in the MIAKT knowledge-base. MIAKT enables access to this data to all the medical personnel involved while also providing supporting classification and analysis tools on top of the data access mechanisms.

Porting MIAKT to the OpenKnowledge system provides a way to keep the innovations in the MIAKT project current while improving the performance and ease of use. Such a port also provides a well established test-bed into which OpenKnowledge can be evaluated for ease-of-integration and performance. Figure 4 shows the original MIAKT architecture. Based around a centralised Java Enterprise server, medical staff would connect via the browser-based interface. The central server provided access to a set of web-services that implemented application-specific functionality on behalf of the software client, depicted in the presentation layer in Figure 4 [10]. A set of abstract concepts was used to mediate service implementations (SOAP, Grid, etc.) and also enabled flexible configuration of both service and presentation layer. A knowledge-base controlled the configuration of the application server and the client accessed the server with simple (informally defined) commands through a servlet interface.

The problem with the MIAKT system became clear when trying to re-install the system in other contexts. The use of a heavy-weight application server that uses SOAP web-services means a great deal of work needs to be put into the installation and configuration of such a system. If that is successfully achieved
Figure 4: The original MIAKT architecture based around a centralised J2EE server

it is clear that the application server will always be a bottle-neck in the process. Although the underlying functionality of MIAKT was already distributed in various web-services, access to that functionality was necessarily through the central server.

Porting the MIAKT system to OpenKnowledge allows us to investigate how the MIAKT system can be improved by removing the centralised server and making the application truly distributed. Using the OpenKnowledge kernel as a means for contacting the services that provide the MIAKT functionality means a central configuration no longer has to be updated when new services become available or services in use become unavailable; the peer-to-peer network appropriates the responsibility of finding the services. Figure 5 shows the updated architecture for the OpenKnowledge port of the MIAKT architecture: the Java enterprise server is replaced with the OpenKnowledge kernel.

The interactions with the services are, of course, controlled by the OpenKnowledge kernel through the LCC interaction modelling language. The interface between the MIAKT user interface (the application as it is seen by the user) and the OpenKnowledge kernel is through the Peer class that is provided
Figure 5: The MIAKT architecture based around the OpenKnowledge kernel with the kernel to ease integration to applications. Figure 6 shows the general overview of interface between an application and the kernel using the Peer class. Conceptually, this class represents the state of a peer and it provides a set of tools for creating applications with the OpenKnowledge system. We designed it to be extended by the developer of an application. It provides methods for subscribing to interactions and for dealing with the requests for action from components that are playing within interactions (shown by the dotted lines in Figure 6).

During the execution of the MIAKT application, the OpenKnowledge network is used to solve the following problems:

- **Patient Record Retrieval**: Retrieval of a patient record from a patient record knowledge-base. In the original MIAKT system the patient record knowledge base was integrated into a SOAP web-service running on a Tomcat server. The code for this webservice is ported to an OpenKnowledge component that has methods that allow querying of the knowledge-base. The knowledge-base is an RDF triple-store (the AKT 3Store [8]).

- **Image Analysis Services**: Analysis of features created from annotations made on a patient’s medical images. The image analysis services were originally run as a SOAP web-service running on Tomcat. Like the
patient database service, it had no formal specifications other than the basic description that was contained within its public WSDL. The MIAKT centralised enterprise server provided a way to annotate these services with concepts from an ontology such that they could be used in the same way as services that used semantic service descriptions. However, this required a large overhead in the configuration of the central server. The advantage of the OpenKnowledge system is that any available image services (available on the OpenKnowledge network as OKCs) can be utilised for image analysis of the image annotations. The interaction is only subscribed to once the feature needs to be analysed, so the system is more dynamic than using a fixed set of web-services, because they will be sought only when they are required.

We will describe these interactions in more detail in the remainder of this section.

### 4.1 Patient Record Retrieval

A patient’s record is stored in the MIAKT knowledge-base as RDF-triples and is made available through an interface that provides querying through standard ontology query languages (SPARQL and RDQL). Generally, SPARQL endpoints (services that provide SPARQL querying to a knowledgebase) are made available over the web such that they can be queried remotely, however when sensitive
data is stored in the knowledge-base access to the endpoint must be controlled. Using OpenKnowledge to access the endpoint through an OpenKnowledge Component, greater access control of the endpoint can be more strictly enforced which means that, in turn, it can be made more generally available.

<table>
<thead>
<tr>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>#00101_patient</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>has_age</td>
</tr>
<tr>
<td>involved_in_ta</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ta-soton-1076933061831</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>consist_of_subproc</td>
</tr>
<tr>
<td>Involve_patient</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>#00101_mammography</td>
</tr>
<tr>
<td>produce_result</td>
</tr>
<tr>
<td>produce_result</td>
</tr>
<tr>
<td>produce_result</td>
</tr>
<tr>
<td>produce_result</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>carried_out_on</td>
</tr>
<tr>
<td>has_date</td>
</tr>
</tbody>
</table>

Figure 7: The presentation of part of a patient's knowledge-base

The process of retrieving patient records is abstracted to generic queries that will execute in any domain. This is because MIAKT provides a general knowledge management interface that we are using for patient record management. Initially, in the MIAKT user interface, the user is presented with a patient search box, that they may use to locate specific patients in the system. The instance browsing interface then allows the user to browse through the case notes associated with that patient in the knowledge-base as shown in Figure 7; each link that the user follows involves the retrieval of the properties of the instance that the user clicked on. This retrieval process is internally handled by the Peer class subscribing to a role in an interaction model that acts as a knowledge retriever.

The MIAKT API for knowledge retrieval is recoded into an LCC interaction model, where the two roles are the knowledge provider and the knowledge retriever. The retriever role actually takes many forms as the retriever will sub-
scribe to the role which best matches the type of query they wish to execute. For example, should an application wish to get all instances of a particular concept (for example, all instances of the \texttt{Patient} concept to find all patients), it would subscribe to the \texttt{instanceListCollector} role that is a simple query-response pattern, as shown in Model 2.

\[
\text{a} \text{(instanceListCollector, C4) ::}
\begin{align*}
& \text{getList(Concept)} \Rightarrow \text{a(instanceDataProvider, P)} \\
& \leftarrow \text{getConcept(Concept) then}
\end{align*}
\]

\[
\text{instanceList(Concept, List) } \leftarrow \text{a(instanceDataProvider, P)}
\]

The knowledge provider subscribes to a role that can respond to all the various forms of query that the model defines. Model 3 shows the query-response pattern is duplicated for each of the possible responses. For the \texttt{instanceListCollector}‘s message, the last choice will be executed causing the constraint \texttt{getInstanceList(Concept, List)} to be executed. This will be an implementation of the SPARQL query interface to access the SPARQL endpoint and retrieve the data. The resulting data will be encoded into the \texttt{List} variable which is presented through the MIAKT interface to the user.

\[
\text{a} \text{(instanceDataProvider, P) ::}
\begin{align*}
& \left( \text{getType(Instance) } \Leftarrow \text{a(instanceTypeCollector, C1) then} \\
& \text{instanceType(Instance, Type) } \Rightarrow \text{a(instanceTypeCollector, C1)} \\
& \leftarrow \text{getInstanceConcept(Instance, Type)} \right) \\
\text{or} \\
& \left( \text{getData(Instance) } \Leftarrow \text{a(instanceDataCollector, C2) then} \\
& \text{instanceData(Instance, Data) } \Rightarrow \text{a(instanceDataCollector, C2)} \\
& \leftarrow \text{getInstanceData(Instance, Data)} \right) \\
\text{or} \\
& \left( \text{getUseList(Instance) } \Leftarrow \text{a(instanceUseListCollector, C3) then} \\
& \text{instanceUseList(Instance, UseList) } \Rightarrow \text{a(instanceUseListCollector, C3)} \\
& \leftarrow \text{getInstanceUseList(Instance, UseList)} \right) \\
\text{or} \\
& \left( \text{getList(Concept) } \Leftarrow \text{a(instanceListCollector, C4) then} \\
& \text{instanceList(Concept, List) } \Rightarrow \text{a(instanceListCollector, C4)} \\
& \leftarrow \text{getInstanceList(Concept, List)} \right)
\end{align*}
\]

Marshalling of the variables from the interaction model run to the local peer is handled by the end-of-interaction event that is fired to the peer by the coordinator for the particular interaction run.
4.2 Image Analysis

During the examination of a patient’s data, a radiologist will be responsible for checking the x-ray mammograms for suspicious looking areas in the image and suggesting a diagnosis for those areas. X-ray mammograms were traditionally film that was viewed on a light box and radiologists may have highlighted suspicious areas with a pen. These manual annotations were then taken to an MDM and considered in collaboration with other evidence to agree on a diagnosis and treatment. The identification and classification of suspicious regions is a complex art and it will be some time before computerised image analysis is able to compete with humans. However, as a decision support tool, classifications of suspicious regions is of use for radiologists to confirm their suspected diagnosis.

![Image](image.png)

Figure 8: The radiology interface in the MIAKT UI that provides annotation of mammograms

The MIAKT radiologist interface, shown in Figure 8, allowed a radiologist to examine a mammogram and create non-destructive annotations on that mammogram. The annotations initially start with the radiologist highlighting a region which they consider to be suspicious using the lasso tool in the interface. The user interface object fires an event to modules that generate low-level descriptors directly from the user-annotation. In Figure 8 the user has delineated a region and added a text annotation; these user-annotations will resolve to three low-level features: a text feature, a shape feature and an image feature. The text feature will contain the value of the text annotation, while the shape feature will contain the shape of the delineation. The image feature will contain
the pixels of the image that are within the delineated shape. The MIAKT user interface has a defined API for these low-level feature generators and more can be plugged-in should new analysis methods become available that require new low-level features.

Once some low-level features have been generated, they are passed to analysis modules within the MIAKT framework that may generate metrics (mid-level annotations) for the given low-level features. It is from these mid-level annotations, or metrics, that we go on to classify to high-level features in a domain ontology using classifiers.

The annotator in the MIAKT API is still independent of the OpenKnowledge system and there may be many analysis modules which are built-in to the framework itself. However, to open the system up to the wider network we implemented a single analysis module that connects MIAKT with the OpenKnowledge network. This module subscribes (via the Peer instance) to an interaction model in an imageAnnotatorCollector role; that is a role that collects mid- or high-level annotations from image annotators (see Model 4).

The role, to which the MIAKT annotator subscribes, is quite simple (see Model 4); it finds peers on the network that are image annotators and asks each one for an annotation for the given feature. It builds a list of the peers it asks and then falls into another loop that waits for each reply. This method is in contrast to asking each one in turn which would make the execution time \( O(n) \), whereas asking in parallel reduces the execution time to \( O(1) \) in theory, although the overall time is practically determined by the interpreter execution time and the peer that is slowest to respond.
The low-level features are a form of multimedia data and the ease with which we define a multimedia-based interaction model shows that OpenKnowledge requires no special modifications for unusual data. This is because of the way we have designed the annotations such that the data format is formalised in LCC and can be transformed, to a limited extent, using the mapping.

In Model 4, the data is organised at a high level using the model annotations that state that low-level features are made from three parts (the feature type, an image description and the low-level descriptor data itself) while images also
have three data-parts (the type of the image, the size of the image and the
URL of the image). This data organisation provides a standard way for OKCs
to access specific parts of the data that are passed around as well as providing
hooks (via the annotations) into which the peer’s subscription-time mapping
can map OKCs to the model. However, as we discuss in section 4.3, there
must be a commonly understood format for the underlying data of the low-level
feature descriptors as, currently, OpenKnowledge has no data transformation
processing.

The image annotator role is a simple role that accepts a feature and, if the
annotator can understand the provided feature and provide an annotation (mid-
or high-level) for the feature, will return the annotation to the collector. If it is
unable to understand or process the feature it will return an error message as
the annotation value. Any peer that can provide annotations (for some feature-
type) can utilise this interaction model for providing their own functionality
to the network. This means the basis on which the analysis of the images in
MIAKT (or any other application that requires image analysis) can grow with
the community.

@annotation(@role(imageAnnotator),
  @annotation(@variable(Feature), feature(type, image(type, size, url), data)))
@annotation(@role(imageAnnotator),
  @annotation(@variable(Annotation), annotation(type, value)))

a(imageAnnotator, I) ::
  annotationRequest(Feature) ⇐ a(imageAnnotationCollector, C) then
    \[
      \begin{align*}
      \text{null} \leftarrow & \ \{ \understandFeature(Feature) \text{ and } \\
                        & \getAnnotation(Feature, \text{Annotation}) \\
                        \text{or } & \text{ Annotation = “Cannot process feature”} \\
      \} \text{ then } \\
      \end{align*}
    \]
  \Rightarrow a(imageAnnotationCollector, C)

(5)

We have ported a small set of simple analysis modules from the original
MIAKT system into OKCs which we run on separate peers. The peers subscribe
to the imageAnnotator role in Model 5 for providing their functionality to the
MIAKT peer. We have created peers for measuring the perimeter length of
a region (given in units based on the scale of the image), the area within the
region (which is currently provided as a pixel count), the average grey-level value
within the region, and a module that gives other shape-based metrics, like the
ratio between the area and perimeter. We are still porting other features from
the MIAKT system for creating histograms of the region of the image within
the boundary and for creating string matchers that match general annotations
into a domain ontology.

The MIAKT framework receives all the results from the annotators (local
and network-based) and displays them to the user. In parallel to providing this user feedback, the MIAKT framework activates another API which takes mid-level annotations and attempts to classify them into a domain ontology. In the case of the MIAKT application, the domain ontology was developed specifically to represent concepts associated with breast-cancer and its treatment. The Breast Cancer Imaging Ontology (BCIO) is described in [1] and [9]. The classifiers we have in MIAKT are also currently quite simple but the extensibility of the MIAKT framework is ensured by well-formed APIs. This helps us with extending the system into a community on the OpenKnowledge network, in the same way we have for the image analysis modules.

The model for classification of mid-level features is very similar to that of the low-to-mid level annotation model. For completeness, the full model is given in Models 6 and 7.

\[
\begin{align*}
\text{@annotation}(\text{@role(classificationRequester)}, \\
\text{@annotation}(\text{@variable(Feature)}, \text{feature(type, image(type, size, url), data)}))
\end{align*}
\]

\[a(\text{classificationRequester(Feature)}, CR) ::
\]
\[null \leftarrow \text{getPeers(‘classificationProvider’, CPs) then}
\]
\[a(\text{classificationRequesterAux(Feature,Classes,CPs,CP2), CR})
\]

\[
\begin{align*}
\text{@annotation}(\text{@role(classificationRequesterAux)}, \\
\text{@annotation}(\text{@variable(F)}, \text{feature(type, image(type, size, url), data)}))
\end{align*}
\]

\[a(\text{classificationRequesterAux(F,Classes,CPs,CPR), CRA}) ::
\]
\[
\begin{cases}
\text{null} \leftarrow \text{CPs} = [] & \text{then}
\text{classificationResult(F,C) } \leftarrow a(\text{classificationProvider}, P) & \text{then}
\text{null} \leftarrow \begin{cases}
\text{Classes} = [C|\text{Classes}] & \text{and}
\text{CPR} = [P2|\text{T}] & \text{then}
\text{null} \leftarrow T = [] & \text{or}
\end{cases}
\end{cases}
\]
\[
\begin{cases}
\text{null} \leftarrow \text{CPs} = [P|\text{T}] & \text{then}
\text{classificationRequest(F) } \Rightarrow a(\text{classificationProvider}, P) & \text{then}
\text{null} \leftarrow CPR = [P|\text{CPR}] & \text{then}
\end{cases}
\]
\[a(\text{classificationRequesterAux(F,Classes,T,CPR), CRA})
\]

(6)
Although we hope to create more classifiers and utilise other classifiers from other projects by porting those to OpenKnowledge, we only have a simple classifier for our current domain ontology. This utilises the area-perimeter ratio of the shape to determine the morphologic descriptor for the shape; shapes with a large perimeter but small area have very unstable edges which is symptomatic of certain tumours, however, the morphology of these tumours have specific medical terminology which we are able to classify into. Figure 8 shows the region of interest semantics, which are generated by the classifications that are automatically executed. The controlled list that is presented is generated from the ontology and allows a radiologist to correct mis-classifications.

4.3 Discussion of the OpenKnowledge Approach

The MIAKT system was initially implemented using distributed web-services controlled through a central server. The idea of extending MIAKT to an open peer-to-peer network has two big advantages; firstly it allows better control of the patient data. Patient data could be controlled exclusively by the patient who can make their data available when they wish to who they wish, wherever they may be. The use of web-services constrains the execution of the service to a web-server that can provide web-service functionality, like TomCat. These enterprise-size applications are not trivial to install. The small footprint of the OpenKnowledge kernel provides an excellent alternative. Secondly, the openness of the OpenKnowledge network means that as new techniques for analysis become available on the network, the applications which are running on the network automatically gain these upgrades.

However, these ideas also have disadvantages. Allowing a patient to control their own data can be both a blessing and a curse, unless data access and authentication methods are made easy for the patient to control. The trust module within the kernel will certainly deal with building trusted relationships between, for example, analysis modules and users invoking the annotators, but the patient needs to know that those wishing to view their data are not only
trusted but also legitimate. Currently this is not possible in the OpenKnowledge network. However, OpenKnowledge will be made available as an open source system; the hope is that a community will build around OpenKnowledge and such advantageous updates will become available.

The openness of using the OpenKnowledge network for bringing in new and better functionalities into an application is obvious. However, unless common data formats are converged upon, incompatible peers may still find themselves participating in an interaction. Although the mapping at subscription time may find a mapping between the functionality of an OKC in a peer and an interaction model, the data that is passed through the methods and variables in the interaction model must also be compatible to ensure a sensible execution or answer. It would be interesting to investigate how data transformation processes can be used to provide a runtime mapping element to interactions. However, this will raise interesting challenges in how to deal with certain data (such as financial data) and where these transformers should execute. If they execute on the peer which is receiving an incompatible message, it at least provides some peace-of-mind to the user that the peer will only provide the transformations that the user has specifically introduced into the peer; however, this also limits a peer to joining new interactions that may be compatible through a transformation process that the peer does not have installed. A powerful way to introduce the transformation processes into the network could be to publish them as OKCs on the network, like any other functionality. A model for data transformation can be subscribed to in order to initiate the conversion of data prior to continuing with the initial model. This is an example of the interaction deferral process that was described in previous deliverables. This process would require some extra functionality on the peer to detect incompatible media types and invoke the appropriate transformation interaction. This raises the question on how to detect incompatible media types and therefore the issue of how to describe the media types without imposing a vocabulary or ontology on the description process which would undermine the openness of the system. This is still an issue that we are investigating.

5 Conclusions

In this paper we have presented the Semantic Logger and the MIAKT applications as they have been ported to the OpenKnowledge system. We have shown that for both projects, an open network like the OpenKnowledge network, provides an ideal basis on which image annotation functionality can be drawn from as as the community grows, the applications will get ‘smarter’. We have shown that we have engineered the OpenKnowledge kernel such that the porting of applications to a peer-to-peer setting requires very little development cost or training overhead, while the application gains the usual advantages of peer-to-peer systems. We have shown specific functionalities of the OpenKnowledge system, such as the dynamic subscription and interaction deferral functions, and the visualisation mechanisms. In particular, this paper has focussed on
showing that the OpenKnowledge system is not limited to text-based data, but can be used in applications where multimedia forms a major part of the data; the image annotation functionality of both the semantic logger and the MIAKT tool show that image annotation can be easily accommodated on the OpenKnowledge network.

References


