TKK Reports in Information and Computer Science
Espoo 2008 TKK-ICS-R12

# SPECIFICATION OF INFORMATION INTERFACES IN PINVIEW

Deliverable D8.1 of FP7 project n° 216529 PinView

Markus Koskela and Jorma Laaksonen

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

TKK Reports in Information and Computer Science
Espoo 2008 TKK-ICS-R12

# SPECIFICATION OF INFORMATION INTERFACES IN PINVIEW

Deliverable D8.1 of FP7 project n° 216529 PinView

**Markus Koskela and Jorma Laaksonen**

Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Distribution:
Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science
P.O.Box 5400
FI-02015 TKK
FINLAND
URL: http://ics.tkk.fi
Tel. +358 9 451 1
Fax +358 9 451 3369
E-mail: series@ics.tkk.fi



ISBN 978-951-22-9673-6 (Print)
ISBN 978-951-22-9674-3 (Online)
ISSN 1797-5034 (Print)
ISSN 1797-5042 (Online)
URL: http://lib.tkk.fi/Reports/2008/isbn9789512296743.pdf

TKK ICS
Espoo 2008

**ABSTRACT:** This report defines the information interfaces for the PinView project to facilitate the planned research of the project. Successful collaborative research between the multiple project sites requires that the individual efforts can directly support each other. The report contains definitions for the used file system structure, for various file formats, and for data transfer between the project sites. The report will be updated regularly during the project.

**KEYWORDS:** information interfaces, file formats, data transfer, content-based image retrieval

**ACKNOWLEDGEMENT:** The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement n° 216529, Personal Information Navigator Adapting Through Viewing, PinView. All deliverables of the PinView project are available at `http://www.pinview.eu/` .

# CONTENTS

# 1 OVERVIEW

This Deliverable D8.1 of the *Personal Information Navigator Adapting Through Viewing*, PinView, project, funded by the European Community's Seventh Framework Programme under Grant Agreement n° 216529, constitutes the output of Task 8.1 *Definition of interfaces for information exchange*. It aims at defining in a sufficient detail the file system structures, file formats and inter-process communication methods needed in implementing the planned research of the project.

One of the first undertakings in the beginning of the PinView project has been to define the ways how all the research efforts in the different work packages can be made to support and progress one another. A practical solution has been to specify the interfaces between all possible software components to be used or developed in the project. Most importantly, those interfaces that facilitate communications between algorithms to be developed at different project sites have been given the strongest emphasis.

The content of this report is an outcome of a joint effort of all the PinView partners. In meetings and discussions between the parties, all necessary information exchange needs have been identified. In addition, a special questionnaire was sent for surveying the needs and suggestions of the other parties. The expertise and individual requirements of all PinView partners have thus been gathered in this report. As such it will serve for the project partners by facilitating the integration of the algorithms and software modules that either already exist or will be developed during the project.

Some of the file formats and other conventions are based on those used in the PicSOM content-based information retrieval system [5], developed at the Helsinki University of Technology (TKK). As the image database of the PASCAL network of excellence Visual Object Classes Challenge 2007 [3] will be used in the experiments, some file structures have been adopted from there too. Finally, lots of eye movement measurement data will be collected and analysed in the PinView project and appropriate data formats will be needed for storing that data. The eye movement measurements will be implemented by using Tobii eye trackers and in the first place the recordings will be stored in Tobii's native file formats. In addition to those device-specific files, the data will also be post-processed to a device-independent XML-based data format specified in the European Community's Sixth Framework Programme Network of Excellence project COGAIN. The COGAIN file format [1, 2] was found to be appropriate for being reused in the PinView project in the earlier Task 1.1 *Study of different forms of enriched feedback* [7].

Despite all efforts, any interface specification for planned pioneering scientific work can never been sufficient and final in its first form. Therefore, it has been agreed that this document will be updated regularly when new needs for communication between separate software components developed in the PinView project occur. The revised versions will then be delivered to the project partners and also to the wider audience at PinView's web site `http://www.pinview.eu/`.

The work presented in this report will be continued next in the PinView Task 1.3 *Definition of transport protocol for enriched feedback*.

## 2 INTRODUCTION

In this report, our main goal is to define the necessary information interfaces for the PinView project at a sufficient level to facilitate the planned research of the project. The definitions are, however, intended to be as extensible as possible. This is due to the fact that any interface specification for a collaborative scientific project is likely to turn out to be limited or insufficient at some point due to the nature of open-ended pioneering research. For this reason, this document is intended to be updated regularly during the course of the project.

The report is organised as follows. In Section 3, we begin by defining the general structure of a media database used in the various experiments within the PinView project. The defined structure closely resembles the database structure used in the PicSOM content-based information retrieval system and has proved to be readily applicable to diverse purposes and experiment settings. In Section 4, the formats of various files used in a PinView database are defined. Most of the format definitions are adopted from earlier projects and other sources, enabling us to capitalise on existing work and to facilitate the commencement of the current project with a robust foundation. In particular, a number of file formats specified are identical to the ones currently in use in the PicSOM system. In Section 5, we specify the methodology used for data transfer between the PinView project partners. This includes both off-line file transfers and on-line data communication needed in distributed real-time applications. Finally in Section 6, we present conclusions and a discussion on the future work in the PinView project.

## 3 FILE SYSTEM STRUCTURE

The media databases used in the PinView project conform to a specification adopted from the generic database structure of the PicSOM content-based information retrieval system, developed since 1998 at the Helsinki University of Technology (TKK).

In PicSOM/PinView, all data files, experiments, and other similar material are always specified related to exactly one database and are located in the subdirectories of the *main database directory*. The name of the main database directory equals the name of the database. Inside the main database directory, the database layout is a multi-level directory structure, which is intended to be easily expandable with both future extensions concerning the whole PinView project and site-specific local additions.

Figure 1 displays the overall structure of the media databases used in the PinView project. The structure consists of a *database level*, which contains common data for whole database, and an *object level*, containing object-specific data. Any additional files or directories may be incorporated into the file system hierarchy provided that they do not overlap with the reserved file names defined in this report. If any files or directories not documented in this report are used in data distributed to other project participants, a description of the additional files and directories should be included in the main database directory.

The primary method for referring to any object in a PinView database is an 8-digit *label* string corresponding to the file name of the primary object, such as an image (see Section 4.1). The label part of a file name is followed by a dot and a three-letter suffix corresponding to the type of the object. The objects are organised into the object level using a three-level directory hierarchy in the `objects` subdirectory. The names of the directories in the hierarchy consist of two digits beginning from the two leftmost digits in the object reference on the first directory level. Similarly, the

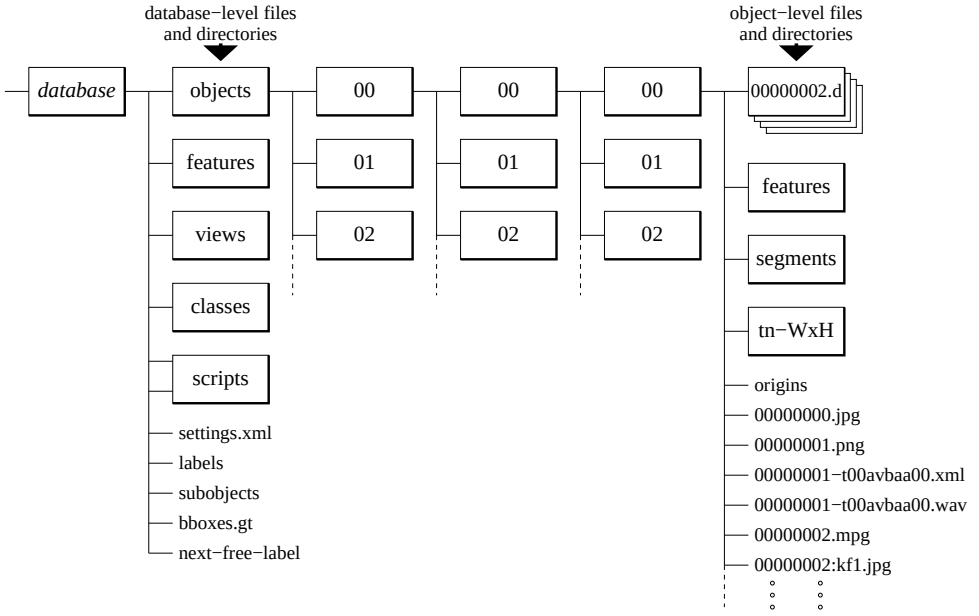


Figure 1: PicSOM/PinView database structure.

directories on the second and third levels are named using the third-fourth and fifth-sixth digits. Each object level directory hence contains at most 100 primary objects. For example, an image in JPEG format having the 8-digit label *12345678* would be stored in the database *mydb* as `mydb/objects/12/34/56/12345678.jpg`.

The database-level and object-level items in the overall structure of Figure 1 are briefly described in Sections 3.1 and 3.2, respectively. The actual content of the files will be detailed in Section 4 and its subsections.

## 3.1 Database-level structure

**objects** A directory containing the three-level directory hierarchy of objects stored in the database.

**features** A directory containing feature vectors extracted from the database objects. On the database level, the feature files in the `features` directory must contain all available feature vectors for the whole database. The feature files do not have to be exhaustive, as objects whose labels are not found in a feature file are considered as having a missing value. Alternatively, the feature vectors may be stored distributed on the object level for the corresponding subsets of the database objects. See Section 4.4 for the description of the format of the feature files.

**views** A directory containing optional *views* to the database. A view is an additional main directory which may contain alternative versions of any files or directories in the database structure. Within a given view, these files and directories replace the original files and directories. For example, an alternative settings file for the view *exp2* of the database *mydb* would be stored as `mydb/views/exp2/settings.xml`.

**classes** A directory containing various subsets of the database objects. See Section 4.3.

**scripts** A directory containing miscellaneous scripts and tools that can be used to work with the database.

**settings.xml** A file specifying various settings for the PicSOM program on how to work with the database. The details of the `settings.xml` file are PicSOM-specific and as such beyond the scope of this report.

**labels** A file containing a list of all valid object labels in the database. See Section 4.1.

**subobjects** A file specifying hierarchical inter-object relations in the database. See Section 4.2.

**bboxes.gt** An optional file specifying rectangular object bounding boxes for the images of the database. See Section 4.6.

**next-free-label** An optional file specifying the next free label that can be given to the next object when adding new objects to the database. See Section 5.3.

## 3.2 Object-level structure

All objects in the database are stored on the leaf directories in the three-level directory hierarchy. The objects having a primary-format label and file name (i.e. an 8-digit string followed by a dot and a suffix) are always stored directly in these object directories. Secondary objects may be stored either in the object directories or in object-specific subdirectories named by the label of the primary object followed by `.d`. See 4.1 for further information on primary and secondary objects in a PinView database.

For example, an MPEG video shot having the 8-digit label *00000002* would be stored as

```
mydb/objects/00/00/00/00000002.mpg
```

and its associated keyframe as

```
mydb/objects/00/00/00/00000002:kf1.jpg
```

or

```
mydb/objects/00/00/00/00000002.d/00000002:kf1.jpg .
```

**features** A directory containing the feature vectors extracted from the database objects in the corresponding object directory. See Section 4.4 for the description of the format of the feature files.

**segments** A directory containing the segmentations for the database objects in the corresponding object directory. See Section 4.5.

**tn-WxH** An optional directory containing thumbnail images of the objects in the corresponding object directory. W and H denote the width and height of the thumbnails in pixels, respectively.

**origins** A file containing information on the source of the objects in the corresponding object directory. It contains a separate line for every object with the following information:

- the object's local file name (label) with extension
- original name of the object
- address of the page or directory where the object was found
- object's content type in MIME notation
- the number of colours in the object if appropriate
- object dimensions (width×height for images)
- object size in bytes
- MD5 checksum of the object
- time when the object was inserted (yyyymmddhhmmss)

The fields are separated with tabs. Each field is mandatory. All missing values thus need to be filled with a dummy value, e.g. a dash ('-').

# 4 FILE FORMATS

The contents and format of various files in a PinView media database are specified in this section.

## 4.1 Object labels

All identifiable objects in a PinView database are referenced to using a unique label. The primary label format is an 8-digit string, e.g. `01020304`. All independent or principal objects generally have this kind of a label. Alternatively, secondary or derived objects which have a clear dependency on a primary-type parent object can be labelled using the label of the parent object with additional prefix and suffix specifiers. The secondary objects may be either real or virtual, i.e. they may or may not have an associated file stored in the file structure.

The general format of a label is:

```
[spec:]plabel[:vsuffix][_isuffix][-esuffix]
```

where `plabel` is the label of the (principal) object, `:vsuffix` and `_isuffix` are suffixes used for video and image objects as the parent object: `:vsuffix` refers to a single frame of a video object, whereas `_isuffix` refers to a subimage segment. Eye movement recording data objects are denoted using `-esuffix`, whose syntax is described in Section 4.9.

The prefix `spec:` is an additional specifier used together with `_isuffix` to differentiate between alternative segmentation-based subobject of the object. For example, let us assume that `01020304` is an image object. Then, it could have the following subobjects `a:01020304_0` and `b:01020304_1` corresponding to two different image segments, *0* and *1*, the former one obtained with a segmentation algorithm *a* and the latter with algorithm *b*. The image segmentation file format is detailed in Section 4.5.

The list of supported object types is shown in Table 1. Each object in a database must have a type specifier denoting the type of the object in question. In addition, Table 2 lists two additional type modifiers which are not independent, but are rather used to modify the corresponding type specifier. The modifier `segment` is used to denote image segment objects, and `file` is used to differentiate real objects that have an associated file in the database structure from virtual objects. The type specifiers and modifiers are combined with the '+' character.

Table 1: The list of supported object types.

| supported object types | specifier |
|---|---|
| images | `image` |
| image collages | `collage` |
| full videos | `video` |
| video shots | `video` |
| audio waveforms | `audio` |
| eye movement measurements | `eye` |
| email / multimedia messages | `message` |
| misc. textual metadata | `text` |

Table 2: The list of object type modifiers.

| object type modifiers | specifier |
|---|---|
| segments | `segment` |
| actual file | `file` |

The list of available objects in a database is stored into a text file named `labels`, which consists of a line-oriented list of all labels (in both the primary and secondary format) followed by the type of the object. The objects are listed on separate lines, and the label and type are separated with a space character. The order of the labels in `labels` is significant and new objects should only be added to the end of the file. The removal of any labels from a `labels` file is not allowed. For example, the objects of the previous example could be represented in a `labels` file as follows:

```
01020304 file+image
a:01020304_0 segment+image
b:01020304_1 segment+image
```

## 4.2 Subobjects

The hierarchical parent–child relations between objects with primary and secondary format labels are evident. Similar hierarchies may, however, also exist between objects which are both represented in the primary label format. Such hierarchies are specified in the file `subobjects`. Common examples of such hierarchies occur when a set of images are combined into a collage, when the individual frames of a video shot are represented in primary format, or when an original video is split into shots.

The format of an entry in the `subobjects` file consists of the parent object, the '+' character, and the corresponding subobjects, all separated by a space character and listed on a single line. Thus, the general format of a line in `subobjects` is

```
parent + subobject1 [subobject2 [subobject3 [...]]]   .
```

The order of the subobjects is significant and subobjects having a natural order (e.g. frames of a video) need to be listed in the correct order.

## 4.3 Classes

All subsets of the database objects are specified as class files that consist of labels of the objects belonging to the class listed on separate lines. Lines starting with a hash sign ('#'), empty lines, and lines containing only whitespace characters are ignored. The name of the class file equals the name of the class. For example, a class file named `classes/cats` could be as follows:

```
# All cats found in the database.
01020304
05060708
22446688
```

In addition, specific reference points for the objects can be stored in the class files. Such reference points are typically obtained by human interaction, e.g. by mouse clicking over an image. The $x$ and $y$ coordinates of the reference points are listed on the same line after the object label according to the pattern:

```
# All cats found in the database with click locations.
01020304 xy=(99,56)
05060708 xy=(45,98)
22446688 xy=(100,123),(300,237)
```

## 4.4 Features

Each type of statistical features extracted from images and other PinView database objects needs to have a fixed dimensionality. Such features are stored in text files that have the file name suffix `.dat`. The definition of the file format originates from that of the LVQ_PAK and SOM_PAK[1] software distributions [4].

The file format is as follows. The first non-comment line must contain only a positive integer value specifying dimensionality of the feature space. The following lines each contain one data vector and its label (see Section 4.1). From the beginning of the line, first are values of the vector components separated by whitespaces, followed by the label also separated by whitespace. Comment lines starting with a hash sign ('#'), empty lines, and lines containing only whitespace characters are ignored. As an example, Figure 2 shows three-dimensional average color features extracted from two images.

```
# The average color feature
3
0.320681 0.230830 0.175782 00000001
0.229737 0.209693 0.214361 00000002
```

Figure 2: An example of a feature file in the SOM_PAK format.

The SOM_PAK data format is also supported by the SOM Toolbox[2] [6] for Matlab. In particular, the SOM Toolbox contains the function `som_read_data`, which can be used to read SOM_PAK format data files into Matlab.

Optionally, the comment lines may contain additional structured information about the feature extraction process in XML format. Figure 3 shows an example of a feature extraction information definition.

[1] `http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml`
[2] `http://www.cis.hut.fi/projects/somtoolbox/`

```
<?xml version="1.0"?>
<feature>
  <name>colorlayout</name>
  <longname>Colour layout</longname>
  <shorttext>
    DCT coefficients of average colour in 20x20 grid.
  </shorttext>
  <target>image</target>
  <vectorlength>12</vectorlength>
  <options/>
  <hasraw>no</hasraw>
  <components/>
  <version>Id: ColorLayout.C,v 1.2 2006/06/16 12:35:19</version>
  <date>Sun Jul  1 12:40:17 2007</date>
  <user>mats</user>
  <host>itl-cl29</host>
  <cwd>/m/fs/home/mats</cwd>
  <cmdline>*picsom-features-internal* -Or colorlayout</cmdline>
</feature>
```

Figure 3: XML file format used for feature extraction information.

## 4.5 Image segments

The format of the image segmentation files used in the PinView project follows the one used in the PicSOM CBIR system. All characteristics of the PicSOM segmentation data format are not likely to be needed in the PinView context and this section addresses only those issues that are currently assumed to be relevant to the PinView project.

Segmentation files record information associated with image objects in a database. Typically segmentation files are used e.g. for recording pixel-level segmentation bitmaps of the corresponding images, merging hierarchies of such bitmapped segments, specifications of geometric subdivisionings of the image area, and manual or automatic annotations of image locations. The image objects and segmentation files are in a one-to-many relationship: one image object may be described by several alternative segmentation files, but one segmentation file always describes just one image object.

Logically, the segmentation files are presented as XML documents. Optionally, a set of bitmaps may also be included in the segmentation file. In the context of segmentation files, the image objects are understood as one-frame image sequences, as the same segmentation file format is used outside the PinView project to describe also genuinely multi-frame video objects.

The segmentation files are thought to result from applying segmentation methods to the image objects. The methods can be chained. The segmentation files record information of the method invocations and the segmentation results that the methods produce. These results can be either specific to a certain frame of the described image sequence or common to all frames of the sequence. In the PinView context, the distinction between sequence-level and frame-specific segmentation results is redundant and we have adopted the convention to regard the results as frame-specific.

In a file system the segmentation files are stored as plain text XML files if no segmentation results in bitmap form are included. When they are, the segmentation files are stored as TIFF files, with the XML document being stored in the textual description field of the possibly multi-frame TIFF sequence. If the result bitmaps

require word lengths exceeding capabilities of common TIFF viewers, the bitmaps may be split into several ones.

The segmentation file names are derived from the described object labels by prefixing the object label with the method specifier followed by a colon, and appending a filename extension `.seg`. An example of a segmentation file name is `rm25:00000765.seg`, which is a segmentation file resulting from the application of segmentation method *rm25* to the image object *00000765*. In the method identifier, chaining of segmentation methods is indicated by a `+` sign, such as for example in `m1+m2+m3:00000765.seg`. In practice, long chained names can be cumbersome, thus the chained name can be replaced by some simple identifier of the chain.

Figure 4 shows a sample XML part of an imaginary segmentation file named as `rm25:00000765.seg`. The *rm25* segmentation files contain two kinds of segmentation results. First one is the segmentation on the image area into 25 segments. The other result describes the process of further pair-wise merging these segments until all the segments have been merged together. The first 25-part segmentation is stored in the bitmap of the segmentation file with the same dimensions as the original image, with indices 0 to 24. The merging process information can be found under XML element `<mergelist>`. In practice, there is only a single `<mergelist>` in the *rm25* segmentation files used in PinView. The entries of the `<mergelist>` specify two source segments that are merged and the identifier of the newly formed composite segment. The source segment identifiers 0–24 refer to the segment-specifying pixel values in the TIFF file bitmap. Figure 4 shows portions of a sample segmentation XML document that are relevant for extracting the `<mergelist>`.

Further details of the PicSOM segmentation format will be described in the later revisions of this report if they turn out to be needed in the implementation of PinView project.

```
<?xml version="1.0"?>
<segmentation xmlns="http://www.cis.hut.fi/picsom/...">
  ...
  <framelist>
    <frame><framenr>0</framenr><processedby>0 1 2</processedby>
      <resultlist>
        <result xmlns="http://www.cis.hut.fi/picsom/..."
                name="regionhierarchy" type="regionhierarchy"
                value="" resultid="0" methodid="2">
          <regionhierarchy>
            <leafregions inbitmap="true"/>
            <mergelist>
              <merge from1="14" from2="15" to="25"/>
              <merge from1="10" from2="13" to="26"/>
              ...
              <merge from1="40" from2="34" to="47"/>
              <merge from1="46" from2="47" to="48"/>
            </mergelist>
          </regionhierarchy>
        </result>
      </resultlist>
    </frame>
  </framelist>
  ...
</segmentation>
```

Figure 4: Portions of a segmentation XML document. Three dots (...) indicate omission of XML elements not relevant for extracting the `<mergelist>`.

## 4.6 Bounding boxes

The image database of the PASCAL network of excellence[3] Visual Object Classes Challenge 2007[4] [3] will be used in the experiments of the PinView projects and the object bounding box file format has been adopted from there.

The VOC2007 image database contains ground truth information for all object classes in the from of rectangular bounding boxes. Figure 5 displays the original XML format bounding box information provided by the organisers of the VOC2007 Challenge. The original VOC2007 filename `000001.jpg` is shown in the file while the PinView database contains the file with name `00000001.jpg`.

The XML-type bounding box information has later been post-processed to a line-oriented format visible in Figure 6 and stored in the file `bboxes.gt` in the root of the VOC2007 database of PinView. The first whitespace-separated column specifies the object label prefixed by `lbl=` and the second column the object class. The remaining entries determine the pose, bounding box (upper-left-$x$ upper-left-$y$ lower-right-$x$ lower-right-$y$), difficulty of recognition and whether the object is truncated or not, similarly to the original VOC2007 Challenge convention.

```
<?xml version="1.0"?>
<annotation>
        <folder>VOC2007</folder>
        <filename>000001.jpg</filename>
        <source>
                <database>The VOC2007 Database</database>
                <annotation>PASCAL VOC2007</annotation>
                <image>flickr</image>
                <flickrid>341012865</flickrid>
        </source>
        <owner>
                <flickrid>Fried Camels</flickrid>
                <name>Jinky the Fruit Bat</name>
        </owner>
        <size>
                <width>353</width>
                <height>500</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>dog</name>
                <pose>Left</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>48</xmin><ymin>240</ymin>
                        <xmax>195</xmax><ymax>371</ymax>
                </bndbox>
        </object>
        ...
</annotation>
```

Figure 5: PASCAL VOC 2007 XML format bounding box information for image `000001.jpg` labeled as `00000001` in PinView.

[3] `http://www.pascal-network.org`

[4] `http://www.pascal-network.org/challenges/VOC/voc2007/`

```
lbl=00000001 dog pose=Left bndbox=(47 239 194 370) difficult=0 ...
lbl=00000001 person pose=Left bndbox=(7 11 351 497) difficult=0 ...
```

Figure 6: Line-oriented bounding box information for image `00000001`.

## 4.7 Manual segmentation bit masks

In addition to bounding boxes, the VOC2007 Challenge organisers have manually annotated some of the database images to generate pixel-wise approximate segmentation masks for the 20 object classes. The segmentation masks are stored as PNG image files. For each database image, there are two segmentation mask files: a file that has a separate label for each object class, and a file that has separate labels for each separate object. These files also have a label for image background and a label for uncertain regions, such as transition areas between objects, severely occluded or small objects and regions difficult to interpret. In total, such segmentation bit masks have been provided for 632 images, out of which 209, 213 and 210 images belong to "train", "val", and "test" sets, respectively [3].

## 4.8 Image collages

For the planned PinView experiments with eye movement recording, special collage images made of smaller or down-scaled subimages are required. The composition of the image collages are specified with special XML files understood by the PicSOM CBIR system.

Figure 7 shows an example of an image collage definition. The `<frameset>` element contains definitions for a set of image collages, each specified by one `<frame>` element. The pixel size and background colour of the image collage is determined from the `width`, `height` and `color` attributes, respectively, of the `<frame>` element. If those do not exist, those of `<frameset>` are used as substitutes. If the `color` attribute is not given, white is used as the default colour.

Each `<image>` element specifies the `label`, `location` (top-left bottom-right $xy$ coordinates) and possible `crop` (top-left bottom-right coordinates) of an image to be inserted in the composite. Optional `border` and `bordercolor` attributes specify the pixel width and colour of the potential border drawn around the image.

The images are inserted into the collage in the order they are specified in the XML file. In the case of overlap the earlier image will be occluded by the later one.

```
<?xml version="1.0"?>
<collage>
  <frameset width="400" height="300">
    <frame background="#ff8080" bordercolor="#000000">
      <image label="00000000" location="10 10 100 100" border="2"/>
      <image label="00000001" location="150 20 350 150" crop=""/>
      <image label="00000002" location="20 160 130 290"
                                  crop="50 50 300 300"/>
      <image label="00000003" location="140 170 390 280"
                                  crop="40 30 250 300"/>
    </frame>
  </frameset>
</collage>
```

Figure 7: XML file format used for image collage definition.

## 4.9 Eye movement data

This section first addresses the question of naming the data files that contain eye movement recordings in PinView. The following two subsections then define the used data file structures.

### Naming of eye movement data files

As already specified in Section 4.1, the PinView data files for storing eye movement recording data are named with two-part file names as `plabel-esuffix.xml`. The primary object label `plabel` specifies the image or image collage label relative to which the gaze recording has been made. The `-esuffix` part specifies the eye movement recording as described in the following.

The `esuffix` is separated from the primary object label with a dash ('-') and is composed of five concatenated character fields of fixed length, which can be symbolically denoted with identifiers `xddtttaann`, where

- `x` is an identifier of the measurement site
  - `t` is TKK
  - `s` is SOTON-ECS
  - `u` is UCL
  - `m` is MUL
  - `x` is XEROX
  - `c` is celum
- `dd` are two digits that specify the measurement setting
  - `00` testing without any specific purpose,
  - `01` experiment with pointer and single click, etc.
- `ttt` are three letters identifying the task within the setting `dd`
  - `01cvd` cat versus dog,
  - `01cvh` cow versus horse, etc.
- `aa` are two letters that specify the test subject inside site `x` and shall not reveal the true identity
  - `aa` first test person,
  - `ab` second test person, etc.
- `nn` are two digits that identify each specific measurement
  - `00` first measurement,
  - `01` second measurement, etc.

In addition to the XML-format eye movement data files, the eye movement recording setup may include also speech-based input from the user. In this case, the corresponding audio recordings are named as `plabel-esuffix.wav`.

### Tobii native data

The textual export function of the Tobii eye tracker consists of multiple data types, viz. *gaze data* (GZD), *fixation data* (FXD), *event data* (EVD), *area of interest data* (AOI), *area of interest list* (AOIL), and *combined data* (CMD). All types of data are exported as tab-delimited text files. The text files are given the name of the recording concatenated with the abbreviation of the type of the file in capital letters. For example, a gaze data recording with the name *my_recording* is given the file name as `my_recordingGZD.txt` and the corresponding fixation data is named as `my_recordingFXD.txt`.

All export files begin with a header, which contains information about the recording date and time, study name, test subject name, recording name, screen resolution and coordinate unit used for gaze data. Some of the export files contain also additional information. For example, the fixation data (FXD) export contains settings information about the filters used to perform data abstraction or to remove malformed data. Figure 8 shows an example of the header format from the Tobii export files. Table 3 and Table 4 list the columns that are created in the raw gaze data (GZD) export and in the fixation data (FXD) export, respectively.

In addition to the textual export, the whole recording process can also be exported as a video file in AVI video format.

```
Data properties:

Recording date: 16.5.2008
Recording time : 15:10:01:234 (corresponds to time 0)
Study: dog-cat-click
Subject: He Zhang
Recording: dog-cat-click
Screen resolution: 1024 x 768
Coordinate unit: Pixels


Filter settings:

Eye: Average
Fixation radius: 30
Min duration: 100
```

Figure 8: An example of header format extracted from Tobii export files.

Table 3: The columns created in the raw gaze data (GZD) export.

| Data column | Description |
|---|---|
| Number | A sequential serial number given to the gaze point |
| Time | The timestamp, in ms, for this gaze point from the start of the recording |
| Screen X (left eye) | The horizontal position of the gaze point, measured in pixels from the left |
| Screen Y (left eye) | The vertical position of the gaze point, measured in pixels from the top |
| Cam X (left eye) | The horizontal location of the pupil in the camera image, on a scale from 0 to 1 |
| Cam Y (left eye) | The vertical location of the pupil in the camera image, on a scale from 0 to 1 |
| Distance (left eye) | The distance from the camera to the left eye, in mm |
| Pupil (left eye) | The size of the pupil, in mm |
| Code (left eye) | The validity of the gaze data |
| ... | The corresponding columns for the right eye |

Table 4: The columns created in the raw fixation data (FXD) export.

| Data column | Description |
|---|---|
| Number | A sequential serial number given to the gaze point |
| Time | The timestamp, in ms, for start of this fixation from the start of the recording |
| Duration | The duration, in ms, of the fixation |
| Screen X | The horizontal position of the fixation centre, measured in pixels from the left |
| Screen Y | The vertical position of the fixation centre, measured in pixels from the top |

### COGAIN

The eye movement data will also be post-processed to a device-independent XML-based data format specified in the European Community's Sixth Framework Programme Network of Excellence project COGAIN[5]. Figure 9 displays the overall structure of the COGAIN data format [1, 2].

```
<?xml version="1.0"?>
<start>
  <ident>1</ident>
  <setup>
    <date>06/06/2008</date><time>11:43:15</time>
    <name>single click on person</name>
    <subject><name>xx</name></subject>
    <tracking_conditions>
      <timescale>ms</timescale>
      <data.scales><x>1</x><y>1</y></data.scales>
      <screen>
        <resolution><x>1600</x><y>1200</y></resolution>
        <size><x>40</x><y>31</y><units>cm</units></size>
      </screen>
      <image><resolution><x>353</x><y>500</y></resolution></image>
      <stimulus>00000001</stimulus>
    </tracking_conditions>
    <device.state>
      <name>Tobii</name>
      <sampling_frequency>50</sampling_frequency>
      <eye>left+right</eye><coordinates>xy</coordinates>
    </device.state>
  </setup>
  <!-- image data -->
  <!-- keyboard data -->
  <!-- audio data -->
</start>
```

Figure 9: The overall structure of COGAIN format eye movement data and an example of header part of data.

[5] http://www.cogain.org

The eye and pointer movement data in PinView are stored relative to image coordinates. An example of this kind of data with four different information types (gaze, fixation, pointer and click coordinates) is shown in Figure 10.

For storing additional information from keyboard events and audio recordings, new data structures have been added to the COGAIN format. An example of keyboard events and audio data are shown in Figure 11.

```
<?xml version="1.0"?>
<start>
  <!-- ident data -->
  <!-- setup data -->
  <image>
    <gaze>
      <samples>
        <sample>
          <timestamp>38</timestamp><number>2</number>
          <eye><x>91</x><y>296</y></eye>
        </sample>
      </samples>
      <fixations>
        <fixation>
          <timestamp>38</timestamp><number>1</number>
          <duration>35</duration><x>91</x><y>296</y>
        </fixation>
      </fixations>
    </gaze>
    <pointer>
      <samples>
        <sample>
          <timestamp>68</timestamp><number>2</number>
          <x>298</x><y>302</y>
          <buttons>left</buttons>
        </sample>
      </samples>
      <clicks>
        <click>
          <timestamp>128</timestamp><number>1</number>
          <x>190</x><y>133</y>
          <buttons>left</buttons>
        </click>
      </clicks>
    </pointer>
  </image>
  <!-- keyboard data -->
  <!-- audio data -->
</start>
```

Figure 10: An example of eye and pointer movement data stored in COGAIN format.

```
<?xml version="1.0"?>
<start>
  <!-- ident data -->
  <!-- setup data -->
  <!-- image data -->
  <keyboard>
    <keys>
      <key>
        <timestamp>78</timestamp><number>1</number>
        <name>x</name><direction>down</direction>
        <modifiers>ctrl</modifiers>
      </key>
    </keys>
  </keyboard>
  <audio>
    <file>00000001-t00aaaxx01.wav</file>
    <nbesthyp>
      <hyp>
        <rank>1</rank>
        <score>0.0012345</score>
        <text>dog</text>
      </hyp>
    </nbesthyp>
  </audio>
</start>
```

Figure 11: An example of keyboard, audio and speech recognition data stored in COGAIN format.

## 4.10 Feature-wise object score value files

In the PicSOM system, it is possible to extract feature-wise score values for each object in the database. Figure 12 displays how this information can be stored in simple line-oriented ASCII format files. The header lines contain free format comments, the first non-comment line specifies the dimensionality of the data vectors and the following lines end with the object label. The score value file format is thus similar to the feature file format described in Section 4.4.

```
# colorlayout dominantcolor edgefourier edgehistogram  ...
11
0.000334672 0.000334672 0.000376506 0.000988804 ... 00000001
0.000133869 0.000239848 0.000143431 0.000119526 ... 00000006
```

Figure 12: An example of feature-wise image score value dumps.

## 4.11 PicSOM retrieval experiment definition scripts

In the PicSOM system, any offline retrieval experiment can be defined using a specific simple scripting language. The details of PicSOM experiment scripting are beyond the scope of this report. However, the same format of scripts will be available to PinView simulations for both online and offline experiments. The exact content of the PinView scripts will vary depending on the type of the experiment and will be specified as the detailed experiment settings are determined.

Figure 13 displays an example of a script for running an offline content-based im-

age retrieval experiment with the VOC2007 database and two features, *colorlayout* and *edgefourier*, with 20 images "shown" to the simulator on each query round. The `cat` object class has been used as the retrieval ground truth target in the experiment.

```
#! picsom
analyse=simulate
database=voc2007-20080403
restriction=train
target=image
features=colorlayout,edgefourier
class=cat_train
maxquestions=20
forcehit=1
initial_set=only_forced_images
result=cat
```

Figure 13: An example PicSOM retrieval experiment specification script.

## 4.12 Retrieval result files

The PicSOM system uses a special XML-type file format for storing the results of offline retrieval experiments. After the experiment, the retrieval performance can be observed from the result file either as displayed *receiver operating characteristics* (ROC) and *recall–precision* (RP) curves or as the scalar values of *equal error rate* (roc_eer), *area under curve* (roc_auc) and *average precision* (avgprec). Figure 14 displays an example of results of a retrieval experiment.

```
<?xml version="1.0"?>
<picsom:analyse_result
  xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
  <picsom:corr>166</picsom:corr>
  <picsom:tot>2501</picsom:tot>
  <picsom:rounds>125</picsom:rounds>
  <picsom:seen>2501</picsom:seen>
  <picsom:found>166</picsom:found> ...
  <picsom:recall size="125">
    6.060605869e-03 2.424242347e-02 ... 1.000000000e+00
  </picsom:recall>
  <picsom:precision size="125">
    5.000000075e-02 1.000000015e-01 ... 6.599999964e-02
  </picsom:precision>
  <picsom:avgprecision size="125">
    3.012048081e-03 4.566353280e-03 ... 1.138411239e-01
  </picsom:avgprecision> ...
  <picsom:false_pos size="2501">
    1.595413385e-11 4.282655427e-04 ... 1.000000000e+00
  </picsom:false_pos>
  <picsom:true_pos size="2501">
    6.024096161e-03 6.024096161e-03 ... 1.000000000e+00
  </picsom:true_pos> ...
</picsom:analyse_result>
```

Figure 14: An example of an XML format retrieval result file, ... indicate omissions.

# 5 COMMUNICATION

In this section, the methodology used for data transfer between the PinView project partners is specified. It consists of off-line file transfers (Section 5.1), eye movement data streaming (Section 5.2) and generic on-line data communication needed in distributed real-time applications (Section 5.3).

## 5.1 File transfers

The file transfers in the PinView project are mainly performed using the ZIP data compression and archival format. The directory structure within the ZIP files should conform to the file system structure specified in Section 3.

For enabling centralised data transfer, a data repository is set up using the Web-based Distributed Authoring and Versioning, or WebDAV[6] protocol. WebDAV is a set of extensions to the Hypertext Transfer Protocol (HTTP) which allows users to collaboratively edit and manage files on remote World Wide Web servers. The WebDAV transfers will use secure socket connections and password protection and be available only for the partners of the PinView project.

Data that is made also publicly available will be placed in PinView's public web pages at `http://www.pinview.eu/`.

## 5.2 Eye movement data streaming

In the communication between the search engine and the gaze tracker equipped image retrieval client there will be a need for asynchronous communication from the client side to the server. The COGAIN project has defined a special streaming data structure that can be useful for this purpose.

Figure 15 shows an example of streamed eye movement data in COGAIN format. One can see that the gaze direction and fixation measurements are encapsulated inside small `<data>` elements. The process receiving the streamed data can later reorder the packets to follow the COGAIN archive format discussed in Section 4.9.

The details of the streaming interfaces will be specified later as a part of Tasks 1.3 *Definition of transport protocol for enriched feedback*, 1.4 *Implementation of pointer track feedback* and 1.5 *Implementation of point-and-speak feedback*.

[6] `http://webdav.org/`

```
<stream.start> ...
  <data>
    <screen>
      <gaze>
        <sample>
          <timestamp>0</timestamp>
          <eye><x>200</x><y>150</y></eye>
        </sample>
      </gaze>
    </screen>
  </data>
  <data>
    <screen>
      <gaze>
        <fixation>
          <timestamp>50</timestamp>
          <duration>35</duration><x>205</x><y>156</y>
        </fixation>
      </gaze>
    </screen>
  </data> ...
</stream.start>
```

Figure 15: COGAIN streaming data, where ... indicate omissions.

## 5.3 SOAP remote procedure calls

When two interacting software components in the PinView project need to change data in a real-time application, the data interchange will be implemented by using SOAP procedure calls[7]. SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.

The general SOAP functionality can be easily implemented in most of the currently used operating systems and programming languages by using available open source interface libraries, for example the CSOAP library[8] in C. The SOAP server–client communication is embedded in XML documents that follow the pattern presented in Figure 16. Only the content inside the `<SOAP-ENV:Body>` tag will depend on the application.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 16: The general structure of SOAP data interchange packets.

[7] http://www.w3.org/2000/xp/Group/

[8] http://csoap.sourceforge.net

The following sections define the basic SOAP calls needed for executing remote procedures on the PicSOM system. All PicSOM SOAP calls use the following parameters:

| | |
|---|---|
| URL: | `http://picsom.pinview.eu:$port/picsom-soap` |
| $port: | 10000 |
| URN: | `urn:picsom` |
| method: | `execute` |

### List of databases

Figure 17 displays an example of the request and response formats of a SOAP remote procedure call for obtaining a list of existing databases.

```
<m:execute xmlns:m="urn:picsom">
  <m:databaselist xsi:type="xsd:string"/>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result
       xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
       <picsom:databaselist>
         <picsom:database>
           <picsom:name>voc2007-20080403</picsom:name>
           <picsom:longname>VOC 2007 snapshot 20080403
           </picsom:longname>
           <picsom:shorttext>Pascal VOC challenge 2007
           </picsom:shorttext>
           <longtext content="xml">
              Database for Pascal VOC Challenge 2007
           </longtext>
           <picsom:size>9963</picsom:size>
           <picsom:tnsizes>120x90</picsom:tnsizes>
           <picsom:views>0</picsom:views>
           <picsom:objecttypes>
              <picsom:file>9963</picsom:file>
              <picsom:image>9963</picsom:image>
          </picsom:objecttypes>
        </picsom:database>
        ...
      </picsom:databaselist>
    </picsom:result>
  </m:result>
</m:executeResponse>
```

Figure 17: SOAP request and response for obtaining the database list.

### Adding an object

Figure 18 displays an example of the request and response formats of a SOAP remote procedure call for adding new objects in an existing or newly created database. In the context of PinView, the objects are typically either image files or files containing eye movement recordings in the COGAIN format.

In the example sketched in Figure 18, two images `boy.jpg` and `dog.jpg` are being added. The first one is referenced to with a filename path relative to the PicSOM server, so the file has to reside in the same file system as the server before issuing the SOAP call. The second file is passed from the SOAP client to the server

side as a binary attachment of the request.

When the object addition request finishes, the SOAP server returns information on the added objects in an `<objectlist>` element containing a list of elements named `<objectinfo>`. Most importantly, the `<label>` element tells the label given to the added object within the database.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">insert</m:analyse>
  <m:database xsi:type="xsd:string">testdb</m:database>
  <m:updatediv xsi:type="xsd:string">yes</m:updatediv>
  <m:args xsi:type="xsd:string">boy.jpg</m:args>
  <m:upload href="cid:0055123458945832">dog.jpg</m:upload>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result
      xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
      <picsom:objectlist>
        <picsom:objectinfo>
          <picsom:label>00000109</picsom:label>
          <picsom:index>108</picsom:index>
          <picsom:targettype>file+image</picsom:targettype>
          <picsom:database>testdb</picsom:database>
          <picsom:segmentationinfolist/>
          <picsom:objectfileinfo>
            <picsom:date>20080613160957</picsom:date>
            <picsom:filename>00000109.jpg</picsom:filename>
            <picsom:format>image/jpeg</picsom:format>
            <picsom:name>00000109.jpg</picsom:name>
            <picsom:original-name>boy.jpg</picsom:original-name>
            <picsom:size>19120</picsom:size>
            <picsom:url>boy.jpg</picsom:url>
            <picsom:available>1</picsom:available>
          </picsom:objectfileinfo>
          <picsom:frameinfo>
            <picsom:colors>&gt;256</picsom:colors>
            <picsom:dimensions>256x384</picsom:dimensions>
          </picsom:frameinfo>
          <picsom:keywordlist/>
          <picsom:subobjects/>
        </picsom:objectinfo>
        <picsom:objectinfo>
          ...
        </picsom:objectinfo>
      </picsom:objectlist>
    </picsom:result>
  </m:result>
</m:executeResponse>
```

Figure 18: SOAP request and response for adding objects in a database.

### Removing an object

Figure 19 displays an example of the request and response formats of a SOAP remote procedure call for removing a set of objects from an existing database. The objects to be removed are specified by their labels. The same labels will never be reused in the context of that database.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">delete</m:analyse>
  <m:database xsi:type="xsd:string">testdb</m:database>
  <m:args xsi:type="xsd:string">00000001 01020304</m:args>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result xmlns:picsom="http://www.cis.hut.fi/picsom/ns"
                   picsom:status="OK"/>
  </m:result>
</m:executeResponse>
```

Figure 19: SOAP request and response for removing objects from a database.

### Recreating the content-based retrieval indices

Figure 20 displays an example of the request and response formats of a SOAP remote procedure call for recreating the indices used for content-based retrieval of objects in the specified database. This time-consuming operation does not need to be performed after each added object. Instead, it will suffice to recreate the indices only after a substantial number of new objects have been inserted in the database.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">create</m:analyse>
  <m:database xsi:type="xsd:string">testdb</m:database>
  <m:features xsi:type="xsd:string">*</m:features>
  <m:createparams xsi:type="xsd:string">tssom-4-100
  </m:createparams>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result xmlns:picsom="http://www.cis.hut.fi/picsom/ns"
                   picsom:status="OK"/>
  </m:result>
</m:executeResponse>
```

Figure 20: SOAP request and response for updating object indices.

### List of available features in a database

Figure 21 displays an example of the request and response formats of a SOAP remote procedure call for obtaining a list of available features in an existing database.

```
<m:execute xmlns:m="urn:picsom">
  <m:database xsi:type="xsd:string">testdb</m:database>
  <m:featurelist xsi:type="xsd:string"/>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result
      xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
      <picsom:featurelist>
        <picsom:feature>
          <picsom:name>zo5:texture</picsom:name>
          <picsom:longname>zo5:texture</picsom:longname>
          <picsom:shorttext>zo5:texture</picsom:shorttext>
          <picsom:longtext>zo5:texture</picsom:longtext>
          <picsom:size>40</picsom:size>
          <picsom:levels>2</picsom:levels>
          <picsom:objecttype>image</picsom:objecttype>
          <picsom:counts>109/110: 109 x file+image
          </picsom:counts>
          <picsom:type>normal</picsom:type>
          <picsom:index>TS-SOM:3x3,9x9</picsom:index>
          <picsom:contains>Mlab+vec</picsom:contains>
          <picsom:selected>1</picsom:selected>
          <picsom:showlevels>1</picsom:showlevels>
        </picsom:feature>
        <picsom:feature>
           ...
        </picsom:feature>
      </picsom:featurelist>
    </picsom:result>
  </m:result>
</m:executeResponse>
```

Figure 21: SOAP request and response for obtaining the list of features.

### Feature values

Figure 22 displays an example of the request and response formats of a SOAP remote procedure call for obtaining the requested feature vector values for given database objects, typically images.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">featurevalues</m:analyse>
  <m:database xsi:type="xsd:string">testdb</m:database>
  <m:features xsi:type="xsd:string">zo5:texture,dominantcolors
  </m:features>
  <m:args xsi:type="xsd:string">00000001 01020304</m:args>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result
      xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
      <picsom:objectlist>
        <picsom:objectinfo>
          <picsom:label>00000001</picsom:label>
          ...
          <picsom:featurelist>
            <picsom:feature>
              <picsom:name>zo5:texture</picsom:name>
              ...
              <picsom:featurevalues>0.45 0.23 0.78 ...
              </picsom:featurevalues>
            </picsom:feature>
            ...
          </picsom:featurelist>
        </picsom:objectinfo>
        ...
      </picsom:objectlist>
    </picsom:result>
  </m:result>
</m:executeResponse>
```

Figure 22: SOAP request and response for obtaining feature vector values.

### Retrieval of the most similar images

Figure 23 displays the request and response formats of a SOAP remote procedure call for retrieving a set of the most similar images for a given example image. The labels of the image objects and their related information are returned in the order of decreasing similarity. The feature-wise similarity scores are normalised so that their overall sum for the most similar image (typically the example image itself if it is included in the database) equals to one. The relative scores are then smaller than one for the rest of the objects returned.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">best</m:analyse>
  <m:database xsi:type="xsd:string">voc2007-20080403</m:database>
  <m:target xsi:type="xsd:string">image</m:target>
  <m:maxquestions xsi:type="xsd:string">10</m:maxquestions>
  <m:features xsi:type="xsd:string">*</m:features>
  <m:positive xsi:type="xsd:string">00000100</m:positive>
  <m:relativescores xsi:type="xsd:string">yes</m:relativescores>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result
      xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
      <picsom:objectlist>
        <picsom:objectinfo>
          <picsom:label>00000100</picsom:label>
          <picsom:index>99</picsom:index>
          <picsom:targettype>file+image</picsom:targettype>
          <picsom:database>voc2007-20080403</picsom:database>
          <picsom:stageinfolist
            xmlns:picsom="http://www.cis.hut.fi/picsom/ns">
            <picsom:stageinfo stage="2" feature="colorlayout"
                              score="0.0862069"/>
            <picsom:stageinfo stage="2" feature="dominantcolor"
                              score="0.0862069"/>
            <picsom:stageinfo stage="2" feature="edgefourier"
                              score="0.137931"/>
            ...
          </picsom:stageinfolist>
          ...
        </picsom:objectinfo>
        ...
      </picsom:objectlist>
    </picsom:result>
  </m:result>
</m:executeResponse>
```

Figure 23: SOAP request and response for obtaining the list of the most similar images to the given example.

### Database deletion

Figure 24 displays the request and response formats of a SOAP remote procedure call for deleting an existing database.

```
<m:execute xmlns:m="urn:picsom">
  <m:analyse xsi:type="xsd:string">deletedatabase</m:analyse>
  <m:database xsi:type="xsd:string">testdb</m:database>
</m:execute>

<m:executeResponse xmlns:m="urn:picsom">
  <m:result xsi:type="picsom:result">
    <picsom:result xmlns:picsom="http://www.cis.hut.fi/picsom/ns"
                   picsom:status="OK"/>
  </m:result>
</m:executeResponse>
```

Figure 24: SOAP request and response for deleting an existing database.

## 6 CONCLUSIONS

In this report, we have documented a number of issues required for the forthcoming collaborative research among the project partners in the PinView project. The report defines the file system structures, file formats and inter-process communication methods needed in implementing the planned research. A large portion of the file formats specified in this report originate from the PicSOM system. This report, however, describes only a certain subset of the conventions used in PicSOM that have been deemed relevant for the PinView project. Additional specifications may thus be needed to be included in later revisions of this report during the course of the PinView project. Furthermore, as any interface specification for open-ended pioneering scientific work will unlikely be sufficient in its initial form, it is the intention that this report as a whole will also be updated during the course of the project. In particular, the eye movement data streaming interfaces will be specified later and included in future revisions of this report.

## ACKNOWLEDGEMENTS

We wish to thank all our collaborators in the PinView project, especially He Zhang, Ville Viitaniemi, and Mats Sjöberg of TKK, Zakria Hussain of UCL, Teofilo de Campos of XEROX and Michael Kumar of celum, for their valuable comments and contributions.

## References

[1] Richard Bates, Howell Istance, and Oleg Spakov. Requirements for the common format of eye movement data. Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 2.2., October 2005.

[2] Richard Bates and Oleg Spakov. Implementation of COGAIN gaze tracking standards. Communication by Gaze Interaction (COGAIN), IST-2003-511598: Deliverable 2.3., March 2006.

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[4] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen. SOM_PAK: The Self-Organizing Map program package. Technical Report A31, Helsinki University of Technology, Espoo, Finland, 1996.

[5] Jorma Laaksonen, Markus Koskela, and Erkki Oja. PicSOM—Self-organizing image retrieval with MPEG-7 content descriptions. *IEEE Transactions on Neural Networks, Special Issue on Intelligent Multimedia Processing,* 13(4):841–853, July 2002.

[6] Juha Vesanto, Johan Himberg, Esa Alhoniemi, and Juha Parhankangas. SOM Toolbox for Matlab 5. Technical Report A57, Helsinki University of Technology, Finland, April 2000.

[7] He Zhang, Markus Koskela, and Jorma Laaksonen. Forms of enriched relevance feedback. PinView FP7-216529 Project Deliverable Report D1.1, April 2008. Available online at `http://www.pinview.eu/deliverables.php`.

TKK REPORTS IN INFORMATION AND COMPUTER SCIENCE

| | |
|---|---|
| TKK-ICS-R2 | Ville Viitaniemi, Jorma Laaksonen<br>Evaluation of Techniques for Image Classification, Object Detection and Object Segmentation. June 2008. |
| TKK-ICS-R3 | Jussi Lahtinen<br>Model Checking Timed Safety Instrumented Systems. June 2008. |
| TKK-ICS-R4 | Jani Lampinen<br>Interface Specification Methods for Software Components. June 2008. |
| TKK-ICS-R5 | Matti Koskimies<br>Applying Model Checking to Analysing Safety Instrumented Systems. June 2008. |
| TKK-ICS-R6 | Alexander Ilin, Tapani Raiko<br>Practical Approaches to Principal Component Analysis in the Presence of Missing Values. June 2008. |
| TKK-ICS-R7 | Kai Puolamäki, Samuel Kaski<br>Bayesian Solutions to the Label Switching Problem. June 2008. |
| TKK-ICS-R8 | Abhishek Tripathi, Arto Klami, Samuel Kaski<br>Using Dependencies to Pair Samples for Multi-View Learning. October 2008. |
| TKK-ICS-R9 | Elia Liitiäinen, Francesco Corona, Amaury Lendasse<br>A Boundary Corrected Expansion of the Moments of Nearest Neighbor Distributions. October 2008. |
| TKK-ICS-R10 | He Zhang, Markus Koskela, Jorma Laaksonen<br>Report on forms of enriched relevance feedback. November 2008. |
| TKK-ICS-R11 | Ville Viitaniemi, Jorma Laaksonen<br>Evaluation of pointer click relevance feedback in PicSOM. November 2008. |

ISBN 978-951-22-9673-6 (Print)
ISBN 978-951-22-9674-3 (Online)
ISSN 1797-5034 (Print)
ISSN 1797-5042 (Online)