CVML – An XML-based Computer Vision Markup Language

Thor List and Robert B. Fisher
School of Informatics, University of Edinburgh, UK
thor.list@ed.ac.uk

Abstract

We propose an XML-based Computer Vision Markup Language for use in Cognitive Vision, to enable separate research groups to collaborate with each other as well as making their research results more available to other areas of science and industry, without having to reveal any proprietary ideas, algorithms or even software. The Computer Vision Markup Language can communicate any type and amount of information, making unavailable functionality accessible to anyone. In this paper we introduce the language and describe how we have implemented it in a very large cognitive vision project. We provide a free open source library for working with this language, which can easily be implemented into existing code providing seamless network communication abilities and multi-platform support. Last we describe the future of CVML and how it might evolve to include other areas of research.

1. Introduction

The last 20 years have seen a remarkable amount of progress in the abilities and usability of Computer Vision and one is left to wonder why we still only see very individual and proprietary use of this work, both in research and in industry. Admittedly, there is a lot of collaboration between groups working in the field, but even these find it very hard to integrate existing research into new projects. This is partly because most people are concerned about protecting their own ideas and IPR, but equally because each group has their own ways of working, has existing code which is not reused much if at all and finally, because they do not communicate with other groups early enough in the project.

The result is a lot of research which is incompatible with the work of other groups. The algorithms are different, the software architectures are proprietary and the results are only presented in papers as graphs and summaries.

There have been several attempts to unite existing efforts in Computer Vision. There are many commercial packages with proprietary representations, but two of the most widely used or promoted are Intel's OpenCV [2] and DARPA's Image Understanding Environment [3]. They offer large amounts of built-in functionality and library routines, but most of these require that all or most of the basic components be built on their libraries using their data structures. They offer little or no support for networking or distributed programming and very few even try to work on more than one platform.

MPEG 7 [4] is a good candidate for a Computer Vision interface language. It is versatile, works with many types of software and hardware, has the ability to tag specific information in each or multiple frames, and is globally recognized as a standard [4]. However, anyone who has ever worked with MPEG knows that it is usually complicated to work with. There is a need for very large – and usually not free – libraries to be included, the data encoding is highly asymmetrical, and the standard is not open enough for people to add their own data types at will. It takes a lot of time to change existing software to work well with MPEG; time that most people do not wish to spend.

We need to accommodate the way people work in this field where the majority create their own software, not compatible with any of the major libraries and very few agree on which platform to use. We need a language which does not change this, but works along side with it while providing a strong interface to the work of others.

2. Computer Vision Markup Language

With the introduction of a common data interface specifically designed for Computer Vision one would enable compatible projects to work together more easily, if not as a unit then as modules in a larger setting. The unique abilities of one group would be accessible to others without giving away any secrets. We have created a language which is easily combined with existing code, and a library which people can use if they wish which runs on all major platforms.

This interface is simple enough so nobody would have to spend too long implementing it, versatile enough to encompass many of the possible needs of functionality, extendible so each group can add their own additional information sources and lastly is partially parse-able. This means that there might be auxiliary information in the data, which can safely be ignored if not understood or expected.

There is one such language already available which has been widely used, namely the XML standard. This has all of the abilities required and furthermore, already is a worldwide institution. It is human-readable even without tools, but viewing can be enhanced with standard applications such as Internet browsers and editors.

3. What is XML?

The World Wide Web Consortium (W3C) designed a general mark-up language in 1974 called Standard Generalized Mark-up Language (SGML), which was adopted as an ISO standard in 1986. This paved the way for the well-known Hyper Text Mark-up Language (HTML). In 1998 a simplified version of SGML (which has proven too complex to use generically) called Extensible Mark-up Language or XML was proposed, which carried most of the features of SGML and HTML such as the extensibility, the hierarchical structure and the possibility of validating the information (or parts thereof) contained in a document [1].

XML is hierarchical by nature and usually contains nested structures of tagged pieces of information. Each is well described and isolated in such a manner that parsers more easily can extract partial information as needed without caring about the rest. For example, given this XML fragment

```
<mydataset var1=”hello” var2=”world”>
   <mydata seq=”22”>...</mydata>
   <otherdata map=”8”>...</otherdata>
</mydataset>
```

if the parser knows about **mydataset** and **mydata**, but does not expect **otherdata**, the latter can be ignored.

This makes XML ideal as an interface language for Computer Vision where a part of a system or a module in a larger application has vast amounts of information to convey to others, who might be interested in some or all of it. Although XML is not specifically designed to carry binary data such as images it can easily be implemented, either embedded or as separate data packages.

4. The Proposal

The Computer Vision Markup Language (CVML) should have the ability to carry any “vision” information one could think of, but an overall structure is needed, describing the content. Standard data commonly used needs to be specified by an agreed format or range of formats. Some of these might be coordinate systems, time or frame count information, areas or enclosures in images, trajectories, contours and gradients.

We have defined an initial set of tags to illustrate the approach. Below we list all of the tag names (at the time of writing) and then we give a detailed description of some of these, followed by an example of how real information is carried using these structures. A more complete description of the tag grammar plus parsers for the tags can be found at

http://www.mindmakers.org/projects/cvml

4.1 List of Tags

Figure 1 shows the current set of tag names in use. These are ordered from smaller to larger data structures.

Basic Types:	**integer**	**float**	**module**
	string	**color**	**sequence**
	time	**size**	**scene**
	point	**pointfloat**	**image**
	line	**polyline**	**contour**
	box	**circle**	**coordinatesystem**
Collections:	**vector**	**list**	**collection**
	dictionary	**timeseries**	**sortedcollection**
Binary Data:	**xbin**	**bindata**	**raw**
Structures:	**pointfeature**	**linefeature**	**polylinefeature**
	areafeature	**featurevector**	**featurevectorlist**
	group	**grouplist**	**subgrouphierarchy**
	grouptracker	**temporalgroup**	**temporalgrouplist**

Figure 1: List of current tag names

4.2 Tag Implementation

Time has always been a troublesome entity as Computer Vision groups hardly ever agree whether to use real time, frame reference time or frame counters. We propose allowing people to specify any quantity they find useful, along side with one or more standard specifications. The XML for this would be

```
<time sec="1064500814" ms="48" frame="3345" ... />
```

The ‘...’ allows many more voluntary information types to be added, but either a global time or a frame counter should be provided.

Another useful entity is location, whether in images or scenes. It can be given as a coordinate, as an offset, with or without a size. To accommodate this we created a point structure, which can be two- or any-dimensional, optionally with a size specification attached.

```
<point x="0" y="0" z="0" ... >
   <size width="0.0" height="0.0" depth="0.0" ... />
</point>
```

One can choose to represent the coordinates as the much-used *x,y*, as $i_1,i_2,...,i_N$ or as polar coordinates. Other optional parameters might be direction, speed and principal axes.

Lists of information are often used in the form of vectors of numbers or strings. We have tried to generalise these to encompass all types of lists and dictionaries.

```
Collection                    Dictionary
<collection>                  <dictionary>
   <entry>...</entry>            <entry name="...">...</entry>
   <entry>...</entry>            <entry name="...">...</entry>
   ...                           ...
</collection>                 </dictionary>

ObjectCollection              ObjectDictionary
<objectcollection>            <objectdictionary>
   <object>...</object>          <object name="...">...</object>
   <object>...</object>          <object name="...">...</object>
   ...                           ...
</objectcollection>           </objectdictionary>
```

Of course, collections can contain other collections, making good use of this as a fully hierarchical language.

It will always be necessary to work with binary data in a variety of formats, such as images in memory or in files. A binary data descriptor, such as the *Binary Description Language* (BinX) [5] detailing the ordering of information, byte lengths, etc. will enable others to read your binary data fully or semi-automatically. The binary data is held elsewhere, but fully described in the XML.

The *Binary Data Descriptor* is written as

```
<binx>                                           ...
   <dataset>                                     1010 0010
      <integer-32 />              ──────────►    1001 1110
      <float-32 />                ──────────►    1000 0001
      <array type="fixed">        ──────────►    1000 1010
         <double-64 />            ──────────►    1001 1010
         <dimindex to="9" />                     1010 0101
      </array>                                   ...
   </dataset>
</binx>
```

Features are a common tool to symbolically express traits of objects or locations found in images or image spaces. Most are described by having a physical location or boundary, by a time or duration and by a set of values, representing the specific type of feature in question.

A **featurevector**, for example, would be represented by a coordinate location i_1,i_2, a timestamp of the image it was found in, and a vector of double values

```
<featurevector i1="2" i2="3">
   <time sec="1066231619" ms="880" />
   <vector type="double">
      <entry>6384220.9</entry>
      ...
   </vector>
</featurevector>
```

Sets of feature vectors are wrapped in a <featurevectorlist>.

Reporting tracked entities in a scene is done with the <entity> tag, where data such as the bounding box and orientation, but also high-level information such as role and scenario is provided. Groups of entities have their own bounding box, role and scenario, and this is output for each frame in a video sequence.

```
<sequence name="Fight_OneManDown">
      <frame number="192">
            <entitylist>
                  <entity id="1">
                        <orientation>151</orientation>
                        <box x="81" y="101" w="31" h="21" />
                        <appearance>visible</appearance>
                        <movement>walking</movement>
                        <role evaluation="1.0">walker</role>
                        <event evaluation="1.0"></event>
                        <scenario evaluation="1.0">immobile</scenario>
                        <situation evaluation="1.0">moving</situation>
                  </entity>
            </entitylist>
            <grouplist>
                  <group id="0">
                        <orientation>103</orientation>
                        <box x="228" y="110" w="55" h="126" />
                        <entities>4,5</entities>
                        <appearance>appear</appearance>
                        <movement>active</movement>
                        <role evaluation="1.0">fighter</role>
                        <event evaluation="1.0"></event>
                        <scenario evaluation="1.0">fighting</scenario>
                        <situation evaluation="1.0">merge</situation>
                  </group>
            </grouplist>
      </frame>
</sequence>
```

Group hierarchies specify that one group is a member of another group, as a set of $G_{id1} \subset G_{id2}$

```
<subgrouphierarchy id="id">
   <parent id="id1" />
   <child id="id2" />
</subgrouphierarchy>
```

4.3 Example of use

Imagine that we have a system with a number of modules to process images captured from a video camera. We could have two modules ***A*** and ***B***, each finding features in the images, outputting lists of feature vectors, to be read by other modules, one of these a feature grouping module ***C***, whose job it is to group features into spatial groups. Module ***C*** then outputs both the groups of features which it found as well as a list of which groups are contained in other groups as members.

The output from module ***A*** would be a feature vector list with features, coordinates and frame time. Module ***B*** would output the same format, but with some additional information as it was also required to calculate feature spatial densities. Module ***C*** knows nothing about feature spatial density and will safely ignore this while still understanding all the feature vectors. After processing all the vectors from both modules, module ***C*** produces an output containing groups of features and group pairings to show individual group memberships.

The CVML language is also used in the hand-labelled ground truth datasets of more than 80 video sequences – monitoring street scenes and shoppers, made publicly available as part of the CAVIAR project at

http://homepages.inf.ed.ac.uk/rbf/CAVIAR

5. The Free CoreLibrary

Software for directly manipulating the language could easily be written into each module, but the full free version of the CoreLibrary made generating and parsing of the XML much easier, especially with its multi-OS support and seamless network integration. It is free to use by anyone and is available for download at

http://www.cmlabs.com/corelibrary

It provides full support for the CVML, along with many objects and interfaces, making networking, multi-threading and data parsing very easy on multiple platforms, including Linux, Windows, MacOSX and PocketPC. It is distributed both as source and a binary library ready to be linked with existing software.

6. What have we found?

XML has some weaknesses in that it tends to become rather voluminous and therefore can both be hard to read for humans and can take time to parse for computers. Tools exists that assist human-readability of sections at a time and the addition of partial parsing would greatly improve the speed of the parsing.

7. The Future

CVML is intended to be extensible and evolving, promoting collaboration between many different branches of Computer Vision and Image Analysis. It is our hope that it can be extended to bridge the gap to industrial use, such as in entertainment, robotics and medicine. The CVML website will be central to the further development of the language and is found on

http://www.mindmakers.org/projects/cvml

CVML is now part of the growing Mindmakers.org network, designed to promote collaboration between groups working within all areas of artificial intelligence. Here people can discuss issues and ideas, as well as help the development of common principles in A.I.

8. Conclusion

We have proposed a new XML-based computer vision data interface language called CVML to enable research groups to more easily work together. The language provides the ability to interface with information sources, allowing access to both known and unknown types of data, without prior knowledge about the source itself.

We have provided a free software library called the CoreLibrary which will assist people getting started with the language with minimal overhead and little change to their existing software.

Acknowledgments

This research was supported by the CAVIAR project, funded by the EC's Information Society Technology's programme project IST 2001 37540. We would like to thank David Hogg for suggesting that we promote this language as a standard.

References

[1] David Mercer, *XML: A Beginner's Guide*, McGraw-Hill Osborne Media, 2001

[2] Gary Bradski, "The OpenCV Library", *Dr. Dobb's Journal November 2000, Computer Security*, 2000

[3] Charles Kohl and Joe Mundy, "The Development of the Image Understanding Environment", *CVPR 94*, pp. 443—447, http://citeseer.nj.nec.com/95374.html, 1994

[4] Fernando Pereira and Rob Koenen, "MPEG-7: A Standard for Multimedia Content Description", *International Journal of Image and Graphics* **Vol 1(3)**, pp. 527—547, 2001

[5] Martin Westhead, Ted Wen and Robert Carroll, "Describing Data on the Grid", *Fourth International Workshop on Grid Computing*, pp. 134—141, Phoenix Arizona, 2003