Project name: Identification of Common Development Environments
Project no.: TA 2.10
Partners: Bielefeld University and Vienna University of Technology

Technical Report for Special Action 13-2
Overview of Software Frameworks for Use in Cognitive Vision Approaches

Wolfgang Ponweiser and Markus Vincze
Automation and Control Institute,
Vienna University of Technology,
Gusshausstrasse 27-29 / E376, 1040 Wien, Austria
{ponweiser,vincze}@acin.tuwien.ac.at

Sebastian Wrede and Christian Bauckhage
Applied Computer Science,
Faculty of Technology, Bielefeld University,
P.O. Box 100131, 33501 Bielefeld, Germany
{swrede,cbauckha}@techfak.uni-bielefeld.de

January, 2005

Abstract

Owing to the ever growing complexity of present day computer vision systems, system architecture has become an emerging topic in vision research. Systems that integrate numerous modules and algorithms of different I/O and time scale behavior require sound and reliable concepts for interprocess communication. Consequently, topics and methods known from software and systems engineering are becoming increasingly important. Especially framework technologies for system integration are required. This report presents the results of the special action 13-2 on evaluating software frameworks suitable for the creation of Cognitive Vision Systems. It discusses functional and non-functional requirements in cognitive vision and compares and assesses existing solutions.

Contents

1 Motivation and Background

Apart from traditional research on algorithms for low and high level computer vision, the last 20 years have seen considerable efforts on integrating such algorithms into larger systems. During the last decade, vision system research further advanced to the ambition of developing *cognitive computer vision systems* (CVS) [5, 8]. These are systems which not only involve computer vision but also employ machine learning and reasoning in order to extend prior knowledge or to verify the consistency of results as well as to manage several computational modules [12].

Obviously, if fast online capabilities for cognitive vision are desired, the complexity of the involved subtasks requires to distribute computations over several machines. Looking at the design of distributed systems, however, often reveals additional degrees of complexity. Distributed architectures usually consist of many components, connectors, patterns and various rules for the connections among building blocks [29]. In order to ensure architectural soundness, integration frameworks thus are mandatory.

Throughout the past decade, several frameworks for vision systems have been proposed [32, 13, 24, 11, 19, 20]. All these frameworks were tailored to certain project specific requirements and thus are of limited generality. However, there are common needs in traditional as well as in cognitive computer vision that can easily be identified (e.g., efficient handling of image data). These of course provide general criteria that can guide a comparison of frameworks.

Practical experience shows that if large scale vision systems are being implemented (in the worst but nowadays somewhat usual case by teams of researchers from different institutes located in different countries), one has to consider not only domain specific requirements but always will face problems of *programming in the large*. Therefore, non-functional requirements have to be taken into account, too. Common examples encountered in practice are reusability, scalability, or transparency. As a consequence, techniques and approaches from software engineering should be cared for when designing vision systems.

Due to the increasing importance of this topic and since only few, less comparative and not very recent reports [6, 25] are available, this evaluative study is requested by a european research network. As it is carried out in collaboration among several research groups with a background in integrated vision systems, experience with various frameworks gathered from different projects and applications is being accounted for. Methodologies and results will be reported here. In order to explain our approach, the next section will outline functional and non functional requirements for (cognitive) vision integration frameworks and presents a resulting classification scheme. The evaluation of several frameworks selected from different origin categories will be presented in section 3 with a comparison of these results in section 4. Finally, a conclusion will close this contribution.

2 Identification of Requirements

Based on experience with integrated systems (cf. e.g. [4, 15, 24]) this section reviews functional and non-functional requirements that have been identified as the most important for the construction of such systems. The functional requirements are further separated into core requirements from CVS and distributed systems engineering attributes.

2.1 Core Requirements from CVS

Cognitive vision systems have to process image data and abstract representations derived therefrom. Hence, *Support for User Defined Datatypes* and *Suitability for Binary Data Transfer* are required. Among the core features of CVS presented in the previous section are reasoning and learning. The reasoning property is a pure functional one. There is no specific requirement for the framework resulting thereof.

Learning and adaption in a CVS requires some kind of memory so *Data Management Facilities* must be provided. Since adaption and attention control induce dynamics into the system, *Dynamic (Re-)Configuration* of architectural building blocks is desired. According to Christensen [5], CVS should be embodied and consequently requires at least a hybrid architecture (*Independence of Architectural Styles*) to enable sensory bottom-up as well as actuatory top-down processing. In terms of *Programmatic Coordination*, a CVS can thus be viewed as a reactive system whose behaviour is constrained by the task.

Following the argument of Christensen and Crowley [6] we believe that *Evaluation Support* is essential, e.g. the possibility to experiment with recorded data flow.

2.2 Distributed Systems Engineering Attributes

As motivated in the previous section, features for the construction of parallel distributed software systems are especially important. Distributed engineering attributes capture this requirements. First of all this includes the *Level of Transparency* [30] provided by the integration framework. As far as possible we require *(Explicit) Interface Specifications* because experience shows that this reduces the risk of interface mismatches. Possible sources for errors are further reduced if the framework allows for *(Active) System Introspection* and *Error / Exception Handling*. These features directly support the *Robustness* of the framework.

Furthermore, the *Framework Performance* and *Scalability* are of course important integration framework features but since they always depend on specific application scenarios, they cannot be rated generally.

2.3 Non-Functional Requirements

In system integration, non-functional requirements are often neglected and considered to be less important. However, experience reveals that they are crucial for project success. The following demands either represent best practices resulting from previous projects of our groups or describe insights gathered from software engineering research.

As integration is a complex task and requires profound knowledge of middleware like, e.g. CORBA, many vision researchers admittedly concentrate on the development of single modules. Consequently, the goal of frequent system integrations could be achieved more often if module developers were able to easily integrate their components. Thus, *Usability* and simplicity in terms of a flat *Learning Curve* are basic essentials of an integration framework.

In long-term research projects aiming at integrated demonstrators, specifications or datatypes may change frequently. Since the resulting modules should be flexible in terms of changeability, *Ease of Modification* is also important. For example, the impact of interface changes on an existing system architecture should be minimal to avoid a versioning problem as known from CORBA [28].

Another requirement directly related to usability and ease of modification is *Rapid Prototyping*. The ability to use a framework not only for prototyping of single modules but also of a whole system supports iterative development. Thus wrong directions in system evolution can more easily be identified, especially if integration is performed on a regular basis starting at an early project stage.

As reusability of existing software modules increases productivity, the *Integration of Legacy Code* or legacy modules, e.g. for object recognition, must be enabled. Ways of integrating existing modules broadly range from dynamically loadable plugins or object-oriented wrapper facades for existing C libraries to a closed framework that does not allow any external dependencies due to constraints enforcing a parallel control architecture.

Framework Sustainability and *Framework Maturity* provide strategic criteria if one has to choose between different framework alternatives. Sustainability characterises the current and expected project activity. In either case, closed or open source framework development, an active development community will ensure support and further bug fixing. The maturity level of a framework characterises in how many projects an integration software has been used and which level of stability it has reached.

Finally, the *Available Documentation* is of invaluable importance. Especially in frameworks resulting from research projects this feature often is neglected. By documentation we do not mean only an appendix in a corresponding PhD thesis. It rather necessitates an up-to-date reference manual and a complete API documentation. For successful reuse of an existing technology this is mission critical.

2.4 Additional Features

So far, we only considered criteria for which a qualitative rating is possible. Some features, however, will improve the framework's applicability in system integration but cannot be benchmarked. These are called additional features.

The *License Type*, *Supported Architectures* and *Operating System(s)* as well as available *Language Bindings* are criteria that help assessing the suitability of a framework for a specific purpose and environment.

The assessment of a communication framework is completed by considering which *Communication Patterns* (e.g. streams, event channels, etc.) it provides and by regarding its *External Dependencies* and *Standards Compliance*.

2.5 Evaluation Scheme

Dealing with framework assessment, a common evaluation scheme did not exist so far. We thus propose a scheme based on the requirements identified above. For each criterion, we apply a five step scale abstractly denoted as {- -, -, o, +, ++}. As an example for the meaning of these units, consider the assessment of the feature *Support for User Defined Datatypes*: In a framework with only a fixed set of data types that can be exchanged between architectural building blocks, support for user defined data structures is almost *impossible* (- -). A *difficult* (-) rating would be assigned, if for communication purposes artificial data types must be decomposed into the native datatypes provided by the framework. If the required coding effort is equivalent to a definition of a class in OOP including serialisation, this would be a reason to rate the framework as being *neutral* (o) regarding the data structure support criterion. If standard serialisation methods are provided, the feature is *supported* (+). An *automatic* (++) optimisation case would require no explicit communication related coding at all.

Details on how the other functional and non-functional criteria are mapped to the five step scale will be described at Annex A. A caveat is that especially for the non-functional criteria the assessment is based on the subjective experiences of the authors. Thus, although our assessment applies software engineering principles as far as possible, we would very much appreciate discussions on these important group of criteria for integration frameworks.

3 Integration Software for Cognitive Vision Systems

3.1 Selection of Candidate Frameworks

Starting with the listed vision processing environments at CV-Online[14], the approaches the authors tried for their own integration work and a survey among all EC-Vision members by email,we compiled a collection of interesting frameworks for evaluation. Of course, to evaluate all of them would have been beyond the scope of this report. Thus, the authors decided on a selection among those candidates on the basis of the following criteria.

First of all, the candidate frameworks have to be usable for integration of computer vision systems. Regarding the visual processing software environments this means that the frameworks should at least be able to integrate several different image processing steps in a single application or in a runtime environment. Secondly, as distribution of processes and parallel processing is often mandatory for building large-scale vision systems that perform in (soft) realtime, frameworks that provide mechanisms for building distributed vision systems were also considered as important.

A third important aspect was the availability of the software. Within this report only software available to the public community, either by request or internet download, was incorporated into the evaluation. An additional criteria for our selection has been that at least one successful integrated system on the basis of the candidate framework should have been published.

In the bottom line, we selected ten frameworks that should be representative for their corresponding category. Those five categories are classical *Vision Processing Libraries*, *Integrated Vision Environments*, *Robotic Frameworks* focused on integration, generic *Middleware Approaches* and *Domain Specific Frameworks* that were developed recently for cognitive vision projects. Next we describe the five categories and the respective examples.

3.2 Image Processing Libraries

Libraries for computer vision purposes often provide a set of low-level image manipulation functions as well as supportive methods and algorithms from related domains like geometry, optics, and mathematics. These functionality can then be used directly from the corresponding programming language for the construction of monolithic image processing components as there is often only weak support for the construction of large-scale distributed systems. Although there are several other libraries, e.g. OpenCV [23], CImg [7] or the MATLAB Image Processing Library [22] we will now focus on RAVL and ImaLab as representatives for this software category.

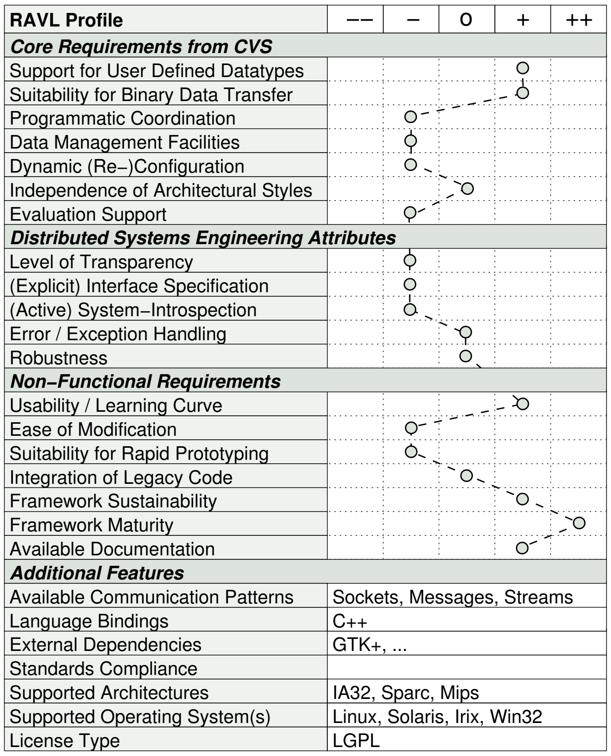
3.2.1 RAVL

RAVL, the **R**ecognition **A**nd **V**ision **L**ibrary, developed within the Centre for Vision, Speech and Signal Processing, CVSSP at the University of Surrey, England is a typical representative of programming libraries for image processing purposes. Our evaluation of RAVL is based on the open-source version that is freely available at Sourceforge [9]. It is being used at least by CVSSP itself, partly in the EU project VAMPIRE[31] and some commercial companies, namely Omniperception Ltd. and Advanced Technology Laboratories, Lockheed Martin.

RAVL consists of a C++ class library together with a range of computer vision, pattern recognition and supporting tools. According to the developers of RAVL some of the features that set RAVL apart from other C++ libraries are:

- SMP/thread-safe reference counting, allowing easy construction of large programs that takes full advantage of multiprocessor servers.
- Powerful IO mechanism, allowing issues for file formats and type conversion to be handled separately from the main code.
- JAVA-like class interfaces which largely avoid the direct use of pointers, allowing code to be written in a clear, readable style.

Those features already address some of our criteria directly which is also reflected in our results. As it should be the case when using a library, artificial datatypes that profit e.g. from reference counting or de-/serialization can be introduced. With regard to typical vision related use-cases classes for e.g. images, matrices, points and corresponding algorithms are widely available.



RAVL Profile	––	–	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				○	
Suitability for Binary Data Transfer				○	
Programmatic Coordination		○			
Data Management Facilities		○			
Dynamic (Re–)Configuration		○			
Independence of Architectural Styles			○		
Evaluation Support		○			
Distributed Systems Engineering Attributes					
Level of Transparency		○			
(Explicit) Interface Specification		○			
(Active) System–Introspection		○			
Error / Exception Handling			○		
Robustness			○		
Non–Functional Requirements					
Usability / Learning Curve				○	
Ease of Modification		○			
Suitability for Rapid Prototyping		○			
Integration of Legacy Code			○		
Framework Sustainability				○	
Framework Maturity					○
Available Documentation				○	
Additional Features					
Available Communication Patterns	Sockets, Messages, Streams				
Language Bindings	C++				
External Dependencies	GTK+, ...				
Standards Compliance					
Supported Architectures	IA32, Sparc, Mips				
Supported Operating System(s)	Linux, Solaris, Irix, Win32				
License Type	LGPL				

Figure 1: Integration suitability assessment for RAVL.

On the one hand, RAVL does not constrain developers and resulting systems to specific architectural styles, on the other hand it gives no or only minimal support on the architecture level, yielding in low assessments for coordination, data management and dynamic re-/configuration.

RAVL was not designed with the aim to provide means for building distributed system architectures. Nevertheless, minimal support for accessing other network nodes is included but limited to low-level socket or stream access with message passing semantics. Besides usual features of the C++ programming language, no additional methods for interface specification, introspection or error handling are provided.

Asking RAVL users about their learning curve most of them answered that it is really easy-to-use, with the exception that RAVL is built completely from scratch and does not make use of e.g. the C++-Standard Template Library (STL). Ease of modification is as complex as in usual C++ programming and thus highly application dependent. Additional libraries can easily be integrated in the qmake build system. Last but not least RAVL can score especially in sustainability and maturity aspects as it is developed for over ten years now and still being further developed at CVSSP. Correspondingly, available documentation is very good at least for the RAVL core classes. RAVL is available under the Lesser GNU Public License (LGPL) and can thus even be used freely in commercial applications across several different architectures and OS platforms while being easily installable with only a few number of dependencies.

Concluding, RAVL seems to be an easy-to-use, well documented programming framework with strong support for image processing and parallel processing on SMP machines, but only minimal support for distributed processing and system engineering.

3.2.2 ImaLab

Imalab is developed by the PRIMA project team [21] of the Laboratoire GRAVIR/IMAG at the Institut National Polytechnique de Grenoble. It is an interactive shell for image processing. It is shipped with a large set (>1000) of vision and vision related functions. Imalab has been used at XRCE - Xerox European Research Centre, Meylan France, Johanneum Research Forschungsgesellschaft mbH, HS-ART Digital Service GmbH, Austria Videocation

Fernseh-Systeme GmbH, Germany, Univ. of Edinburgh, Edinburgh, UK Instituto Superior Tecnico, Lisbon, Portugal, Neural Networks Research Centre, Helsinki University of Technology (HUT), Jaakko Pöyry Consulting, Helsinki, Finland and the Universite de Liege, Belgium.

Imalab mainly benefits of two underlying tools:

- The Prima Vision library containing the C++ based implementations of many fundamental computer vision algorithms.
- The Ravi system, which is an extensible system kernel providing an interactive programming language shell (similar to C and Schema).

Since Imalab has no direct support for any kind of distributed processing it is not well suited for the generation of a large CVS. Hence some of the evaluation criteria are of no relevance because of the missing distribution capability (e.g. Level of Transparency). Additionally there are no data management facilities or programmatic coordination functionalities implemented. Imalab provides only plug-ins like library method calls which can be loaded during runtime.

The advantage of the straightforward component structure are robustness and good usability with a very small learning curve. As long as the system doesn't cross the one computer border Imalab is well suited for rapid prototyping. Especially a lot of vision processing GUI features are available, which supports the system development process. Due to the Ravi system kernel with the focus on dynamic loading and automatic program generation within an interactive shell the integration of legacy code is simple. Imalab is the outcome of several man years of development. As mentioned at the beginning there is a reasonable community. Therefore documentation is easily availabe and sufficient to get started using Imalab.

Imalab Profile	––	–	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes			○		
Suitability for Binary Data Transfer			○		
Programmatic Coordination		○			
Data Management Facilities		○			
Dynamic (Re–)Configuration				○	
Independence of Architectural Styles	○				
Evaluation Support			○		
Distributed Systems Engineering Attributes					
Level of Transparency	○				
(Explicit) Interface Specification				○	
(Active) System–Introspection		○			
Error / Exception Handling		○			
Robustness				○	
Non–Functional Requirements					
Usability / Learning Curve					○
Ease of Modification			○		
Suitability for Rapid Prototyping				○	
Integration of Legacy Code					○
Framework Sustainability				○	
Framework Maturity				○	
Available Documentation				○	
Additional Features					
Available Communication Patterns	(local) Procedure Call				
Language Bindings	C, C++, Schema (Lisp), Prolog				
External Dependencies	Ravi, PrimaVision, svideotools				
Standards Compliance					
Supported Architectures	IA32				
Supported Operating System(s)	Linux				
License Type	GNU GPL				

Figure 2: Integration suitability assessment for Imalab.

Imalab is a workbench to develop a vision algorithm based on image processing functions. Related to the integration frameworks the outcome of an Imalab system corresponds to a single component of the integration framework. Consider that Imalab provides no distributed processing. Hence 'communication' is shortened to a method call.

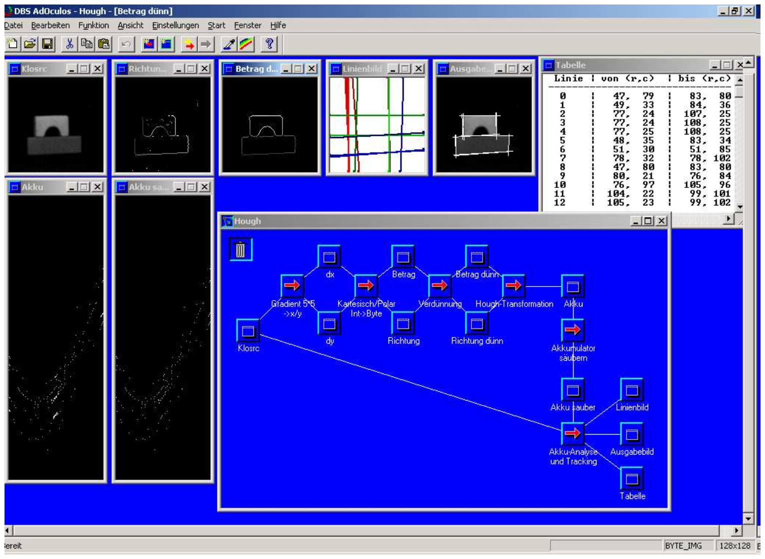
3.3 Interactive Image Processing Environments

Primary goal of graphical environments for image processing is to facilitate rapid development of machine vision and image analysis applications. This task is usually carried out within a graphical workbench, where processing operators for e.g. image analysis, morphology or pattern matching can be dragged around and easily linked by connection lines. Where the self explanatory behavior of this category of software tools leads to very good usability most of them provide only little support to create large-scale distributed vision systems. In the following we will focus on Ad Oculos as a rather simple tool also used for education purposes and on VisiQuest as a commercial high-end software for a broad range of computer vision applications similar to HALCON [17]. Concluding this section, a brief description of IceWing as a tool for high-performance image and video analysis is given.

3.3.1 Ad Oculos

Ad Oculos is a commercial tool distributed by 'The Imaging Source Europe GmbH' [1]. Ad Oculos serves as a platform for image processing algorithms. It is based on a graphical user interface where images, histograms and text windows can be connected by function items.

[1]http://www.theimagingsource.com/



```
Linie ¦ von (r,c)  ¦ bis (r,c)
--------------------------------
  0   ¦   47,  79  ¦   83,  80
  1   ¦   49,  33  ¦   84,  36
  2   ¦   77,  24  ¦  107,  25
  3   ¦   77,  24  ¦  108,  25
  4   ¦   77,  25  ¦  108,  25
  5   ¦   48,  35  ¦   83,  34
  6   ¦   51,  30  ¦   51,  85
  7   ¦   78,  32  ¦   78, 102
  8   ¦   47,  80  ¦   83,  80
  9   ¦   80,  21  ¦   76,  84
 10   ¦   76,  97  ¦  105,  96
 11   ¦  104,  22  ¦   99, 101
 12   ¦  105,  23  ¦   99, 102
```

Figure 3: The graphical user interface of Ad Oculos.

The resulting pipe&filter structure can be extended to just multiple in/outputs of the according functions (e.g. the 'add' function has 2 inputs).

As can be seen in Figure 3 Ad Oculos is a typical example of a graphical programming tool. The focus is on the easy combination of different image processing functions. To adapt these functionalities or writing new ones every processing function is encapsulated in plug-in libraries. Changing the behavior of a function is done by recompiling the according library.

Ad Oculos has no distribution functionalities. Together with the fixed pipe&filter structure most of the core requirements from CVS and systems engineering attributes are either of no relevance or poorly done.

On the other side Ad Oculos is very easy to handle. It is shipped with a text book that explains the purpose, realization and theory of several "classical" image processing algorithms. Ad Oculos is developed over more than one decade and the community using it for education is wide spread.

Still the fixed plug-in structure and the fixed communicated data types hamper modifications between components. Typically the in/output data of cognitive vision functions are not simple integers or images. The integration of available code dealing with such kind of data is difficult too. Especially these properties makes it unsuitable for rapid prototyping of larger scale systems.

Conclusive Ad Oculos serves as a very nice teaching tool. But it provides no distribution capabilities and the integration of legacy code is difficult. Communication is shortened to simple data push. It can be used as a starting point to develop the vision basis of a single component, but not for the creation of a distributed CVS.

Ad Oculos Profile	––	–	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes	○				
Suitability for Binary Data Transfer	○				
Programmatic Coordination	○				
Data Management Facilities	○				
Dynamic (Re–)Configuration		○			
Independence of Architectural Styles	○				
Evaluation Support			○		
Distributed Systems Engineering Attributes					
Level of Transparency	○				
(Explicit) Interface Specification	○				
(Active) System–Introspection	○				
Error / Exception Handling	○				
Robustness	○				
Non–Functional Requirements					
Usability / Learning Curve					○
Ease of Modification	○				
Suitability for Rapid Prototyping		○			
Integration of Legacy Code	○				
Framework Sustainability				○	
Framework Maturity					○
Available Documentation				○	
Additional Features					
Available Communication Patterns	Data push				
Language Bindings	C				
External Dependencies	none				
Standards Compliance	none				
Supported Architectures	IA32				
Supported Operating System(s)	Windows				
License Type	Commercial, free student vers.				

Figure 4: Integration suitability assessment for Ad Oculos.

3.3.2 VisiQuest

VisiQuest formerly known as Khoros Pro is a commercial tool distributed by AccuSoft[2]. It provides a large set of libraries implementing vision algorithms (erosion, edge detectors), 3D vision, GUI construction and visualisation, OS abstraction and data access management. Additionally to the Visual Programming Environment VisiQuest provides data analysis and visualisation facilities. VisiQuest is used at more than 20 universities.

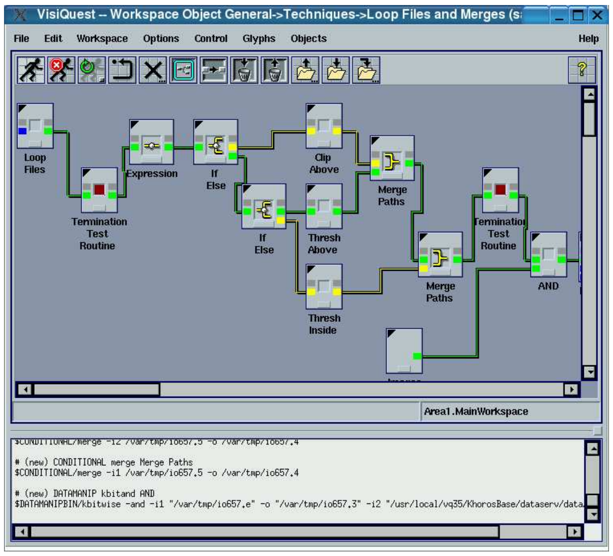
The strength of VisiQuest are its two different programming levels. The first one is the Visual Programming Environment as can be seen in Figure 5. In/output pairs of black-boxes containing the required functionality can be connected. The main difference to the mass of graphical programming tool already this programming level supports feedback loops and conditional execution paths for programmatic coordination.

The other programming level is called Craftsman. It is a more classical programming environment guiding the function programmer through all required steps to generate a program that fits into the framework. Programs generated using Craftsman can either be started from the command line, or it can be used as a function block in the visual programming environment. The Craftsman tool especially supports the programmer during the interface specification to ensure data matching between components as well as an according graphical representation in the Visual Programming Environment.

For the generation of distributed CVS especially the OS abstraction is of interest, because it provides remote execution of components. Using the TCP socket based data access mode of VisiQuest it is possible to establish a distributed system. Still all communication has to be implemented by the component and application programmer. Hence there are no features for any kind of transparency. VisiQuest also provides just an example logging client but a reimplementation of error-return-values and exception handling like C++.

Although the Visual Programming Environment provides a fast introduction of the tool the mass of available functionalities is difficult to handle. And the necessity to deal with two different programming levels increases the effort to integrate legacy code. On the other side these two levels support rapid prototyping regardless if the development focuses on one component or the composition and interplay of several components.

Originally Khoros [19] the predecessor of VisiQuest has been under the development at the Department of Electrical and Computer Engineering at the Univerity of New Mexico since 1987. Later on it is shipped by



```
$CONDITIONAL/merge -i2 /var/tmp/io657.5 -o /var/tmp/io657.4

# (new) CONDITIONAL merge Merge Paths
$CONDITIONAL/merge -i1 /var/tmp/io657.5 -o /var/tmp/io657.4

# (new) DATAMANIP kbitand AND
$DATAMANIPBIN/kbitwise -and -i1 "/var/tmp/io657.e" -o "/var/tmp/io657.3" -i2 "/usr/local/vq35/KhorosBase/dataserv/data
```

Figure 5: VisiQuest's Visual Programming Environment.

VisiQuest Profile	––	–	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				○	
Suitability for Binary Data Transfer		○			
Programmatic Coordination				○	
Data Management Facilities			○		
Dynamic (Re–)Configuration		○			
Independence of Architectural Styles			○		
Evaluation Support		○			
Distributed Systems Engineering Attributes					
Level of Transparency	○				
(Explicit) Interface Specification				○	
(Active) System–Introspection		○			
Error / Exception Handling				○	
Robustness			○		
Non–Functional Requirements					
Usability / Learning Curve			○		
Ease of Modification		○			
Suitability for Rapid Prototyping				○	
Integration of Legacy Code			○		
Framework Sustainability					○
Framework Maturity					○
Available Documentation				○	
Additional Features					
Available Communication Patterns	RMI				
Language Bindings	C++				
External Dependencies	none				
Standards Compliance	none				
Supported Architectures	IA32, SGI, ...				
Supported Operating System(s)	Linux, Windows, Mac OSX				
License Type	Commercial				

Figure 6: Integration suitability assessment for VisiQuest.

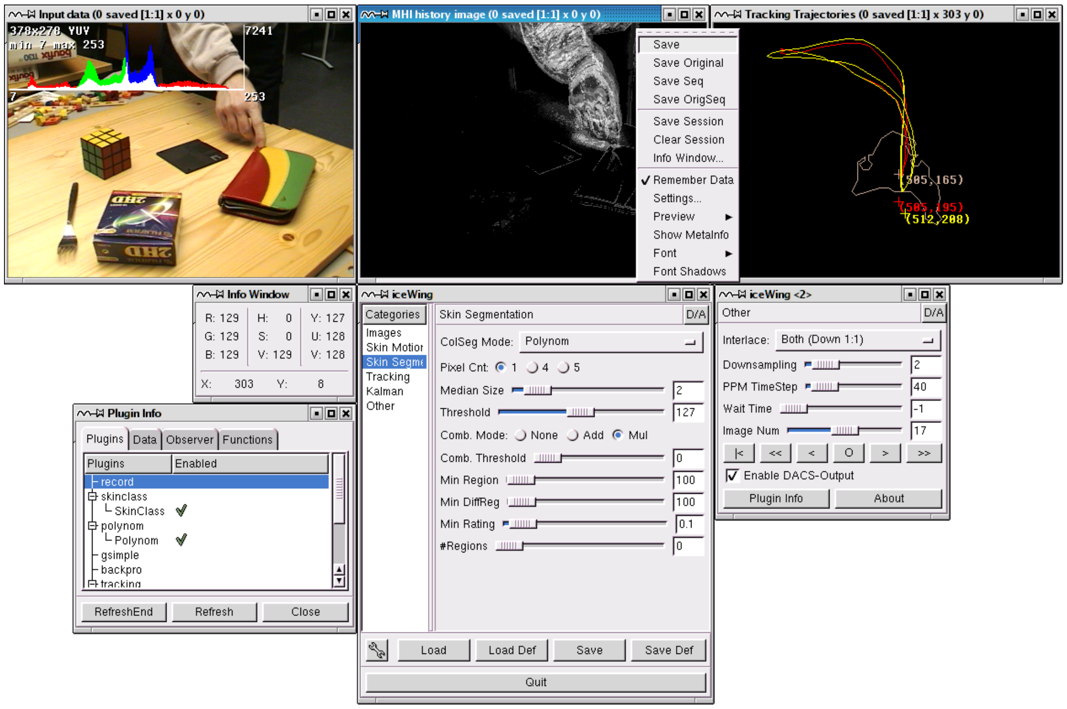
[2] http://www.accusoft.com/imaging/visiquest/

Khoral, Inc as Khoros Pro. Already Accusoft still extends the tool for e.g. chemoinformatics. Therfore the community ranges from academics to business. And the included documentation contains a lot of tutorials and examples.

VisiQuest is best suited to create system prototypes composed by combined vision algorithms. Because of the maturity of the tool there is a very large set of methods available. The visual programming level supports the fast and flexible linking of different methods. Because of the function = component paradigm the resulting architecture is always similar to pipe&filter, and the only communication pattern available is a conditional method call.

3.3.3 IceWing

IceWing, an **I**ntegrated **C**ommunication **E**nvironment **W**hich **I**s **N**ot **G**esten[3], is a graphical plugin shell. It can be used for single image analysis, but is especially useful for realtime video-stream processing and was developed at Bielefeld University, Germany in the Applied Computer Science group. It is there widely used for image processing, e.g. filters, object recognition and action recognition.



Info Window:

R: 129	H: 0	Y: 127
G: 129	S: 0	U: 128
B: 129	V: 129	V: 128
X: 303	Y: 8	

Plugin Info:

Plugins	Enabled
record	
skinclass	
SkinClass	✓
polynom	
Polynom	✓
gsimple	
backpro	
tracking	

Skin Segmentation D/A

ColSeg Mode: Polynom
Pixel Cnt: [✓ 1] 4 5
Median Size 2
Threshold 127
Comb. Mode: None Add [✓ Mul]
Comb. Threshold 0
Min Region 100
Min DiffReg 100
Min Rating 0.1
#Regions 0

Other D/A

Interlace: Both (Down 1:1)
Downsampling 2
PPM TimeStep 40
Wait Time -1
Image Num 17
|< << < O > >>
✓ Enable DACS-Output

Figure 7: Exemplary screenshot of a running IceWing system.

Predefined or self-written plugins operate hierarchically on the pre-given data and can also generate new datastreams. IceWing provides mutual communication among the plugins. There is e.g. the predefined plugin for reading images from grabber-hardware with the help of the avlib or from previously recorded video files on a hard disk. Although there are many existing plugins for object or face recognition, users can extend IceWing by writing their own plugins in C or C++. IceWing itself has no features for building distributed systems, but was extended by plugins for the AMI, see section 3.6.1, and DACS, see section 3.5.1, for parallel processing.

Not being only a batch-plugin-shell IceWing is also a highly customizable GUI platform for the plugins: It has a list of given GUI-elements and allows the plugins to simply make use of them. So plugins can show the user their current status and can let the user change parameters on the fly. Moreover, methods for easy visualization of plugin results are available. The plugins can open any number of windows and display in these windows any data in a graphical form. Figure 7 shows an example of a running IceWing systems with several instantiated plugins.

Regarding our evaluation scheme, IceWing scores well in support for user defined data types, and its suitability for binary data transfer as it was designed for real-time video processing. Working as a monolithic plugin shell, only pointers are transferred between processing modules allowing artificial data types to be exchanged.

[3]This is a reference to an older program, the predecessor of IceWing.

Coordination is supported in a way that plugins get called when new data is available for them. The usual architectural style that is naturally supported by IceWing is pipe-and-filter, even so other styles can be realized within IceWing by plugin developers. Minimal evaluation support is provided by the ability to record and replay input image data.

In IceWing only pointers to data structures are exchanged and thus there is no reliable interface specification at all. Error/exception handling is limited to programming language features of C or C++. Thus robustness is totally dependent on the skills of the plugin developers. If one plugin fails, the whole application stops working.

IceWing is very easy to use if only the existing plugins are combined in a application dependant manner, but gets more complicated if new plugins have to be developed. Modification aspects are delegated to the programming language. Although plugins are developed in plain C or C++ this is an andavantage when existing software should be reused. Therefore it is often easy to reuse external libraries in a plugin.

Concluding, IceWing is a tool for a non-distributed low-level image processing which is highly extensible and already widely used within several research projects. As up-to-date documentation and the software is freely available, IceWing may be an option low-level image processing on several platforms and architectures.

IceWing Profile	−−	−	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				○	
Suitability for Binary Data Transfer					○
Programmatic Coordination			○		
Data Management Facilities		○			
Dynamic (Re−)Configuration			○		
Independence of Architectural Styles			○		
Evaluation Support		○			
Distributed Systems Engineering Attributes					
Level of Transparency	○				
(Explicit) Interface Specification	○				
(Active) System−Introspection		○			
Error / Exception Handling		○			
Robustness	○				
Non−Functional Requirements					
Usability / Learning Curve			○		
Ease of Modification			○		
Suitability for Rapid Prototyping			○		
Integration of Legacy Code				○	
Framework Sustainability			○		
Framework Maturity			○		
Available Documentation			○		
Additional Features					
Available Communication Patterns	N/A				
Language Bindings	C, C++				
External Dependencies	GTK+, AVLIB				
Standards Compliance					
Supported Architectures	IA32, Alpha, PowerPC				
Supported Operating System(s)	Linux, OSF, MacOS−X				
License Type	GPL				

Figure 8: Integration suitability assessment for IceWing.

3.4 Robotic Frameworks

Building robotic systems is a similar complex engineering task as the construction of large-scale cognitive vision systems. Example given, visual perceptions are often also important for robotic systems. This leads to the assumption that integration frameworks originating from that domain are also useful for computer vision systems. In the following, SmartSoft as a typical representative framework from the robotics domain will be evaluated against our criteria to prove this hypothesis.

3.4.1 SmartSoft

SmartSoft [26] is a software framework developed at the Research Institute for Applied Knowledge Processing, FAW Ulm under direction of Christian Schlegel for the SFB 627 project. It is applied at universities as well as some companies.

It is designed for building distributed, component based systems in the autonomous robotic domain. Some of the robots include sensor data processing. Hence there are already some vision related implementations available. The standard architecture realised by SmartSoft is a layered hierarchy, which is the common structure of actual robotic systems. The version evaluated (cvs of 1.4.2004) is based on the CORBA implementation ACE/TAO.

The core functionality of SmartSoft is a set of communication patterns. A special communication pattern serves external access to the component state machine which enables distributed programmatic coordination. Also a wiring pattern for the dynamic composition and configuration of components is available.

A drawback of the used TAO base is the conversion of all communicated data to the CORBA::any type that prevents the direct transfer of binary data. There is an example logging client available but up to now there are no specific data management facilities implemented. The components generated by SmartSoft are connected by TAO's name service. Hence access and location transparency are available.

The component interfaces of SmartSoft are structured by the defined communication patterns. But for any modification of the interfaces all related components have to be considered. To be able to build a system using SamrtSoft the programmer has to learn the component structure, the communication patterns and the Interface Definition Language of CORBA for data definition. Hence to start prototyping by using SmartSoft can be complicated. The strength of SmartSoft appears later on in the product cycle where the prototype has to be optimised and different subversions have to be evaluated. Due to the robotic background of SmartSoft the main focus is on framework performance. Smartsoft is well suited for the integration of sensor data processing within the focus on the robotic domain.

SmartSoft Profile	−−	−	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes			○		
Suitability for Binary Data Transfer	○				
Programmatic Coordination				○	
Data Management Facilities			○		
Dynamic (Re−)Configuration				○	
Independence of Architectural Styles			○		
Evaluation Support			○		
Distributed Systems Engineering Attributes					
Level of Transparency			○		
(Explicit) Interface Specification			○		
(Active) System−Introspection		○			
Error / Exception Handling		○			
Robustness			○		
Non−Functional Requirements					
Usability / Learning Curve		○			
Ease of Modification		○			
Suitability for Rapid Prototyping		○			
Integration of Legacy Code			○		
Framework Sustainability			○		
Framework Maturity			○		
Available Documentation			○		
Additional Features					
Available Communication Patterns	Send, Subscr., Event, Config.				
Language Bindings	C++				
External Dependencies	ACE/TAO				
Standards Compliance	CORBA (ACE/TAO, IDL)				
Supported Architectures	IA32				
Supported Operating System(s)	Linux				
License Type	LGPL				

Figure 9: Integration suitability assessment for SmartSoft.

3.5 Middleware Approaches

Middleware usually connects two otherwise separate components and serves as the glue between two applications. It is, therefore, distinct from import and export features that may be built into one of the applications. This definition of middleware addresses already one of the problems for the use of a generic middleware like RPC, DCOM, XML-RPC or SOAP for integration of cognitive vision systems: Its totally domain independent and provides no specific support for image processing related algorithms or data. Instead, parallel processing and features for distributed systems are focused. In the following DACS and TAO as two representatives of this software category are described.

3.5.1 DACS

DACS, the **D**istributed **A**pplication **C**ommunication **S**ystem, is a framework for building distributed systems across different OS platforms and architectures. It has been developed in a collaborative research project at Bielefeld University, Germany. Primary goals during the development were performance and robustness. It was successfully used in the image processing and robotics domain [13].

DACS consists of a C-API, several daemons and some tools for administration and monitoring the system. The core library is based on only a minimum of necessary OS functionality, mainly Threads, Unix Domain and TCP/IP Sockets which yields a minimal number of dependencies and nicely separates data and control communication. Additional framework tools are dependant on a TCL/TK installation. Regarding communication semantics, DACS features message passing, 1:N streams and a-/synchronous remote procedure calls (RPC). Furthermore, DACS allows monitoring of distributed applications, recording and replaying of data streams with specialized framework tools. As DACS is designed with performance aspects

DACS Profile	−−	−	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				○	
Suitability for Binary Data Transfer					○
Programmatic Coordination		○			
Data Management Facilities	○				
Dynamic (Re−)Configuration		○			
Independence of Architectural Styles				○	
Evaluation Support				○	
Distributed Systems Engineering Attributes					
Level of Transparency			○		
(Explicit) Interface Specification			○		
(Active) System−Introspection				○	
Error / Exception Handling		○			
Robustness				○	
Non−Functional Requirements					
Usability / Learning Curve			○		
Ease of Modification		○			
Suitability for Rapid Prototyping		○			
Integration of Legacy Code			○		
Framework Sustainability		○			
Framework Maturity		○			
Available Documentation		○			
Additional Features					
Available Communication Patterns	RPC, 1:N Streams				
Language Bindings	C				
External Dependencies	none				
Standards Compliance					
Supported Architectures	IA32, Alpha, Sparc				
Supported Operating System(s)	Linux, OSF, Solaris				
License Type	Free for non−commercial use				

Figure 10: Integration suitability assessment for DACS.

in mind and completely written in C, it can be used for fast transfer of very large data structures.

There is only minimal support for specific architectural styles and no support for data management or dynamic reconfiguration of running modules as there is no mean of a component or object in this framework. Although evaluation support is given with the simulation tools for data streams, there is no equivalent technique for simulation of procedure calls available. Besides access transparency, DACS provides location transparency through the DACS directory daemon nameserver. Using a C-API there are no means of a sophisticated exception handling although the framework itself is very stable.

Although usability was one goal during development, specific knowledge about the API is necessary to build distributed systems with DACS. Modifications, e.g. of data types can be problematic as access methods for abstract datatypes are automatically built at compile time and existing interfaces may get broken. With optional parts in NDR[4] specifications a bit of the versioning problem might be solved but depends highly on the software developers. A caveat in using DACS for a newly starting project might be that it is no longer updated or maintained.

3.5.2 CORBA

CORBA, the so called Common Object Request Broker Architecture is a standard which is being maintained by the OMG[5]. CORBA itself is a specification, not an implementation[3]. The implementation is done by some third party and by now, several free and commercial implementations exist.

Standard compliant CORBA implementations usually follow an object-oriented programming scheme. Interoperability and portability is reached through independence of specific hard- and software and many supported programming languages. Scalability can be reached by the inherent distributed nature of CORBA objects. IDL[6] is used as a universal language for the definition of interfaces and data structures. Special components, so called „common object services" offer for example name services, which means locating objects by their name or trader services, which allow object localisation by the properties of the offered services.

These and other features of CORBA support the decoupling of components which must be an important requirement for system architectures built of components. Newer CORBA implementations also comply with real time constraints, e.g. TAO[7] a CORBA-implementation based on ACE[8] which will be described as a powerful CORBA representative in the next section.

In general, CORBA middleware can be helpful as a basis for distributed computing frameworks, but it is very generic and thus it offers only little help for the developer with a concrete task in our domain of cognitive computer vision.

Another rather problematic aspect of CORBA is that most implementations do not adhere completely to the given OMG standards. Thus, the original idea to provide one portable standard for distributed systems was foiled and typical advantages of a standards-based solution are lost.

3.5.3 TAO

TAO (The ACE ORB) [27] is an open source standards-compliant real-time CORBA middleware implementation by Douglas C. Schmidt at the Institute for Software Integrated Systems at the Vanderbilt

TAO Profile	−−	−	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				○	
Suitability for Binary Data Transfer					○
Programmatic Coordination				○	
Data Management Facilities			○		
Dynamic (Re−)Configuration			○		
Independence of Architectural Styles			○		
Evaluation Support			○		
Distributed Systems Engineering Attributes					
Level of Transparency				○	
(Explicit) Interface Specification			○		
(Active) System−Introspection		○			
Error / Exception Handling				○	
Robustness			○		
Non−Functional Requirements					
Usability / Learning Curve	○				
Ease of Modification		○			
Suitability for Rapid Prototyping			○		
Integration of Legacy Code			○		
Framework Sustainability			○		
Framework Maturity					○
Available Documentation					○
Additional Features					
Available Communication Patterns	Object Request Brokering				
Language Bindings	C++				
External Dependencies	ACE				
Standards Compliance	CORBA				
Supported Architectures	IA32, Alpha, ...				
Supported Operating System(s)	UNIX / Linux, Windows, RT OS				
License Type	open source				

Figure 11: Integration suitability assessment for TAO.

[4]NDR is the *N*etwork *D*ata *R*epresentation of DACS for the de/serialisation of transfered data.
[5]Object Management Group, http://www.omg.org
[6]Interface Definition Language
[7]The ACE ORB
[8]Adaptive Communication Environment

University. TAO is used at many universities and companies including Boeing/McDonnell Douglas, Siemens and Motorola. The target group of TAO are developers of distributed and embedded applications.

Because of the CORBA standard the communicative criteria are very well designed at TAO. The IDL supports flexible user defined data types and binary data transfer. But there is no automatic interface verification and no design to specify meta data. Also there are no tools supported for Active System Introspection. According functionalities have to be implemented by the programmer.

TAO provides asynchronous method handling and a lot of CORBA services like name, trading, logging, load balancing real-time and event. These services support several kinds of transparency and programmatic coordination. Where **O**bject **R**equest **B**roker'ing can be used to establish nearly any kind of architectural style none is specifically supported. As well there are no data management facilities implemented.

The biggest disadvantage of TAO is a fact derived from the generality of CORBA itself. CORBA tries to merge many integration and communication functionalities. Therefore the volume of available functionality is enormous which on the other side is very difficult to handle.

The strength of TAO is its leading role in the CORBA realisation. Therefore it has a great community. TAO is one of the completest open source CORBA implementations. A lot of different services and extentions are provided. And the underlying ACE environment serves as a supportive OS abstraction.

3.6 Domain Specific Frameworks

Integrating large-scale cognitive vision research projects usually starts with an overview of frameworks well suited for this task. New integration toolkits with differing focus but strong support for some core requirements of cognitive vision projects like an active memory or dynamic control were recently developed. Originating from this domain they are presented here in their own right even though they can be used for other integration tasks, e.g. in robotics.

3.6.1 AMI/XCF

The **A**ctive **M**emory **I**nfrastructure (AMI) is a framework for software integration in Intelligent Systems research that uses XML throughout. It has been developed by one of the authors of this report and applied in the European project VAMPIRE [31] to realize an augmented reality system for a digital personal assistant. The AMI concept [32] consists of three closely collaborating libraries (and corresponding tools): XCF for building distributed systems, the Active Memory itself for process coordination and data management, and a supporting library named XMLTIO that supports users with an XPath-based API with simple XML processing. The complete integration software is licensed under the GPL and is available for download at Sourceforge [2].

Throughout the whole framework XML serves as the message exchange format for integration of distributed components in the communication library of the AMI, the *XML enabled Communication Framework* (XCF). It supports (non-)blocking remote method invocation and publisher-subscriber communication semantics. All data is XML, with attachments for binary data like images. Exposed methods are bound and invoked dynamically, with schemas optionally providing run-time type safety.

AMI/XCF Profile	––	–	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes				●	
Suitability for Binary Data Transfer			●		
Programmatic Coordination			●		
Data Management Facilities					●
Dynamic (Re–)Configuration			●		
Independence of Architectural Styles					●
Evaluation Support				●	
Distributed Systems Engineering Attributes					
Level of Transparency				●	
(Explicit) Interface Specification			●		
(Active) System–Introspection				●	
Error / Exception Handling				●	
Robustness			●		
Non–Functional Requirements					
Usability / Learning Curve				●	
Ease of Modification				●	
Suitability for Rapid Prototyping				●	
Integration of Legacy Code				●	
Framework Sustainability			●		
Framework Maturity		●			
Available Documentation			●		
Additional Features					
Available Communication Patterns	RMI, 1:N Streams, Events				
Language Bindings	C++, Java, Python				
External Dependencies	ICE, BDBXML, Boost, libsigc				
Standards Compliance	XML, XPath, XML Schema				
Supported Architectures	IA32				
Supported Operating System(s)	Linux				
License Type	GPL				

Figure 12: Integration suitability assessment for AMI.

The ICE [18] communications engine forms the basis of this component and was chosen for its high performance, especially low latency calls.

Coordination between and decoupling of components is achieved through a flexible event-notification mechanism carried out in the *Active Memory* (AM). The event manager is co-located with the persistence back-end of the AM, a native XML database, the Sleepycat DB XML [10]. Event subscription can contain an XPath to narrow down documents of interest. Coordination is thus data-driven and not dependent on the presence of specific components.

Using XML as data format allows users to define artificial data structures. Although binary data could be encoded in XML, the framework supports native encoding of e.g. images for performance reasons. Coordination is carried out by the above mentioned event manager and is thus very flexible while not being restricted to this coordination strategy. One goal of the Active Memory development was the ability to access the transmitted data from different processing components. Thus, the active memory with the underlying native XML database provides generic and powerful data management facilities. Dynamic configuration is not directly supported but can be easily developed through the event notification mechanism. Evaluation support for publisher/subscriber is provided with tools that allow recording and replaying of publisher streams, a more sophisticated simulation toolkit is at the time of writing this report work in progress.

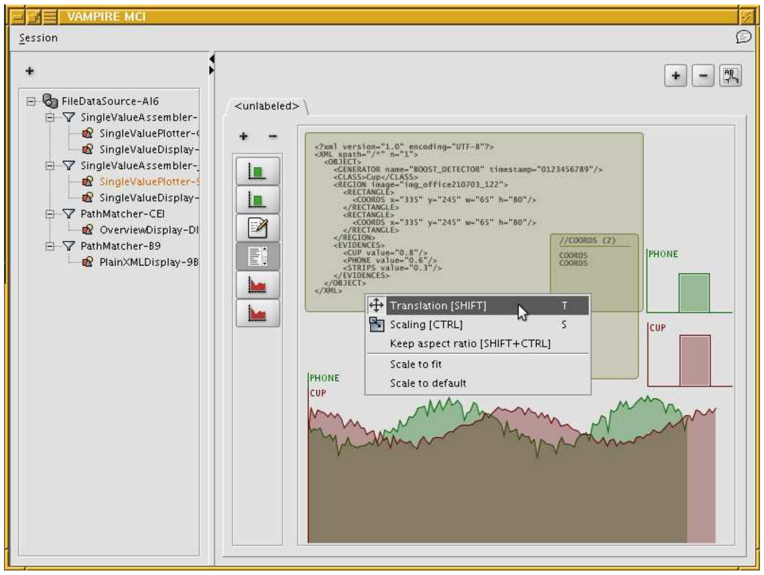

```
<?xml version="1.0" encoding="UTF-8"?>
<XML xpath="/*" n="1">
  <OBJECT>
    <GENERATOR name="BOOST_DETECTOR" timestamp="0123456789"/>
    <CLASS>Cup</CLASS>
    <REGION image="img_office210703_122">
      <RECTANGLE>
        <COORDS x="335" y="245" w="65" h="80"/>
      </RECTANGLE>
      <RECTANGLE>
        <COORDS x="335" y="245" w="65" h="80"/>
      </RECTANGLE>
    </REGION>
    <EVIDENCES>
      <CUP value="0.8"/>
      <PHONE value="0.6"/>
      <STRIPS value="0.3"/>
    </EVIDENCES>
  </OBJECT>
</XML>
```

Figure 13: Exemplary screenshot of the AMI introspection tool MCI.

With the underlying XCF as transport layer, a high transparency level is achieved, including e.g. migration transparency that allows components to change their physical location at runtime. For abstract data types, explicit interface specifications can be given and validated by XML Schemas for all parameters of exposed methods or publisher interfaces but currently no specification of complete interfaces is possible. System introspection at runtime is supported through generic framework tools (e.g. the MCI as shown in Figure 13). Even more specific introspection is possible by registering interceptors in the active communication objects of XCF. User exceptions are serialized in remote invocations and can then be handled as standard exceptions. Robustness is good due to error transparency in case of component failure and the possibility to decouple processes through the active memory.

The public framework API consists only of a small number of cohesive classes and is very easy to learn and understand given that some knowledge of XML standards like XPath or XML Schema is present.

Regarding ease of modification this approach benefits directly from XML features. The XPath-based access on data structures stills works, even when data types are evolutionary enhanced by new information.

Being a recently and still actively developed framework, maturity is rather low, but as the framework is used in two related EU-projects and by several universities it is already quite stable. Concluding, it is worth to mention that beside C++ language bindings for Java as well as (partly) Python and Matlab exist.

3.6.2 zwork

zwork [24] is an open source framework developed by one of the authors of this report at the Automation and Control Institute of the Vienna University of Technology. It is designed to dynamically connect components based on descriptive parameters of the component functionality. It is applied at the Cognitive Vision EU project ActIPret [1] and the Austrian joint research project Cognitive Vision [16].

zwork components offer their functionality on a yellow page where other components interested in this functionality can select from. To optimise the selection process a set of function (=service) specific parameters are added to every offer describing the performance of the implementation (Quality of Service). Usually the resulting architecture is a hierarchical structure, where every communication link consists of a service requester and a service provider.

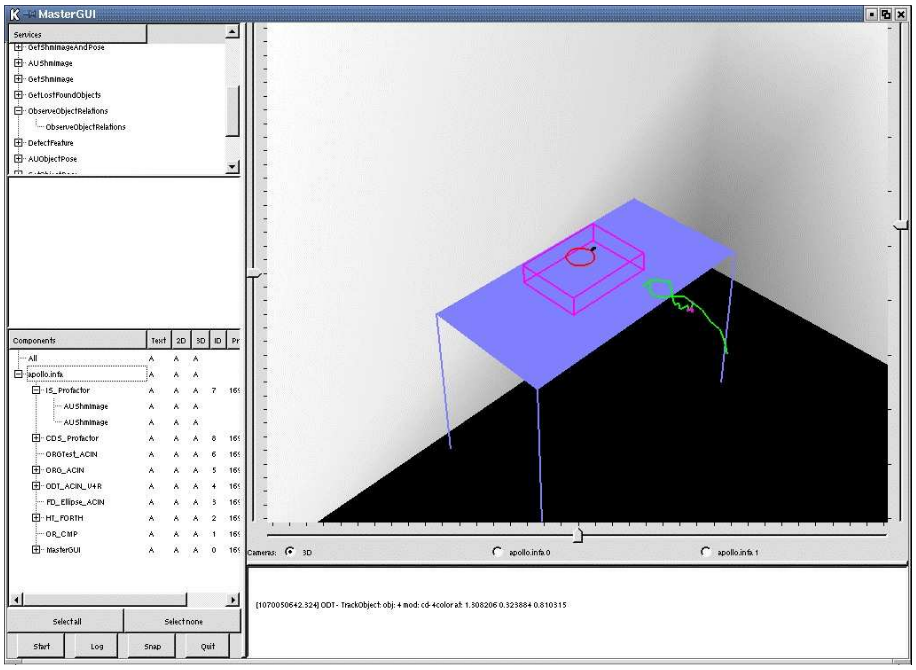
Since the focus of zwork is on the dynamic composition of avaliable components its strength are the programmatic coordination, the dynamic reconfiguration and the artificial interface specification. Since all framework specific component programming has to be done by derived classes the access to transmitted data for active system introspection is trivial. On the other side this reduces the possible robustness the framework can provide.

Usability, ease of modification and suitability for rapid prototyping and integration of legacy code are directly depending on the artificial interface generation. If the programmer can adapt the inter-component dependencies to the service based interface description these properties are nicely fulfilled by zwork. zwork also provides an optional GUI (see Figure 15) to further support these properties.

zwork is already in use at several universities. Because of its early stage in the product live cycle there is still great potential for extention and optimisation regarding robustness and usability.

zwork Profile	−−	−	o	+	++
Core Requirements from CVS					
Support for User Defined Datatypes			○		
Suitability for Binary Data Transfer				○	
Programmatic Coordination				○	
Data Management Facilities			○		
Dynamic (Re−)Configuration				○	
Independence of Architectural Styles			○		
Evaluation Support			○		
Distributed Systems Engineering Attributes					
Level of Transparency			○		
(Explicit) Interface Specification					○
(Active) System−Introspection				○	
Error / Exception Handling			○		
Robustness		○			
Non−Functional Requirements					
Usability / Learning Curve			○		
Ease of Modification			○		
Suitability for Rapid Prototyping			○		
Integration of Legacy Code				○	
Framework Sustainability			○		
Framework Maturity		○			
Available Documentation			○		
Additional Features					
Available Communication Patterns	RMI, Request, Subscribe				
Language Bindings	C++				
External Dependencies	QT for the optional GUI				
Standards Compliance	RPC				
Supported Architectures	IA32				
Supported Operating System(s)	Linux				
License Type	GPL				

Figure 14: Integration suitability assessment for zwork.



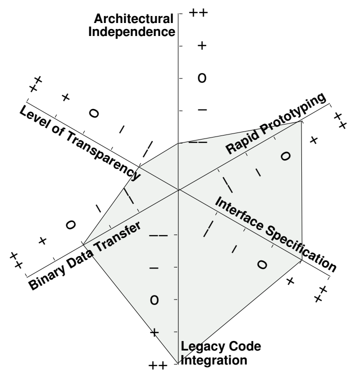
Components	Text	2D	3D	ID	Pr
All	A	A	A		
apollo.infa	A	A	A		
IS_Profactor	A	A	A	7	16
AUShmImage	A	A	A		
AUShmImage	A	A	A		
CDS_Profactor	A	A	A	8	16
ORGTest_ACIN	A	A	A	6	16
ORG_ACIN	A	A	A	5	16
ODT_ACIN_V4R	A	A	A	4	16
FD_Ellipse_ACIN	A	A	A	3	16
HT_FORTH	A	A	A	2	16
OR_CMP	A	A	A	1	16
MasterGUI	A	A	A	0	16

```
[1070050642.324] ODT - TrackObject: obj: 4 mod: cd-4color at: 1.308206 0.323884 0.810315
```

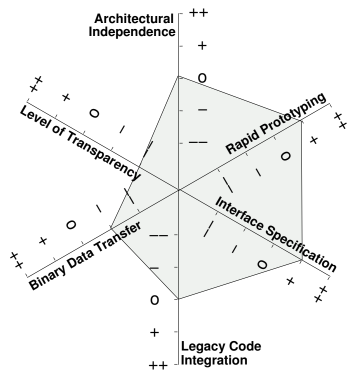
Figure 15: The Grafical User Interface of zwork.

Conclusively zwork focuses on the description of component capabilities and there dynamic semantic adaption. It is well suited for getting experience in integration and optimisation of cognitive and vision components as well as the interplay of these components.

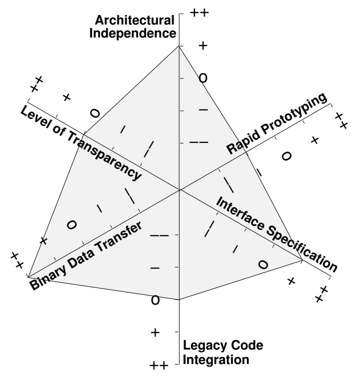
4 Comparison of Integration Frameworks



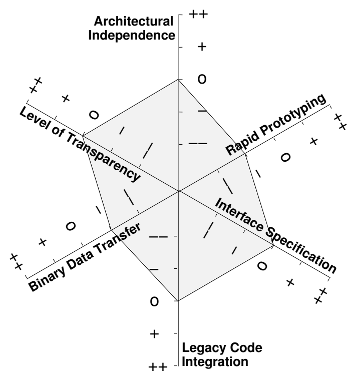
(a) Imalab



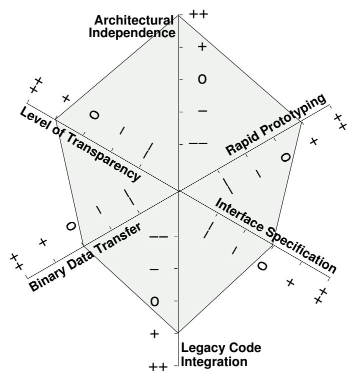
(b) Visiquest



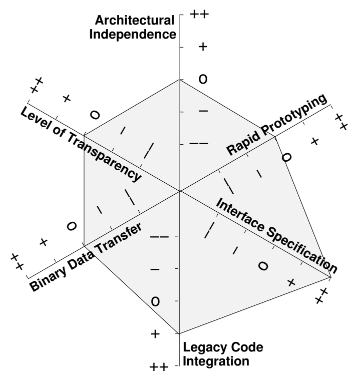
(c) The ACE ORB



(d) SmartSoft



(e) AMI/XCF



(f) zwork

Figure 16: Comparison of frameworks representative for their origin along their most discriminative features.

The goal of this evaluation is the identification of frameworks that support the development of CVS. As worked out in section 2 there are a lot of different requirements. Of course no framework pointed out as being 'the' framework. Contrarily the frameworks can be categorised according to their original intention. In this sense the break-down of the previous sections already expresses a characteristic of respective strengths.

Image Processing Libraries like RAVL and Imalab first of all provide a large collection of vision methods. They usually lack support for typical cognitive and distribution capabilities. Therefore they are more suited to generate single components than composing a complete CVS.

Interactive Image Processing Environments like AdOculos, VisiQuest and IceWing provide facilities to quickly create pipe-and-filter style vision systems allowing for rapid prototyping. However, this class of frameworks normally lacks support for distributed architectures which prevents them from being used for large scale vision systems.

Classical *Middleware* approaches like the CORBA implementation TAO are very generic and powerful. On the other hand they suffer from this genericity because it complicates their use. Moreover, they do not support requirements specific for the cognitive vision domain.

Frameworks like SmartSoft are designed for the *Robotics Domain*. They often allow for distribution and provide transparency. However, in robotics, real-time constraints are a major topic which introduces additional complexity in using these frameworks. Furthermore, their integration facilities are often closely coupled to components for robot control.

The last class identified in our study consists of two examples especially designed as *Cognitive Vision System Frameworks*: While XCF focuses on data management facilities, distribution and rapid prototyping, zwork mainly deals with the programmatic coordination and dynamic reconfiguration required in dealing with the control aspect of cognitive vision systems.

As it can be seen from Figure 16, each group of frameworks exemplified by one representative framework concentrates on different aspects in system integration. Thus, system engineers will have to carefully identify their needs and project requirements and correspondingly decide for a most suitable framework. Discriminative design criteria to start a framework selection are:

- Should the framework provide *distribution* capabilities expressed by *Level of Transparency*?
- Are you planning a single *monolithic test program*[9], a system for solving a small set of tasks or a modular CVS? According requirements are *Dynamic (Re-)Configuration*, *Evaluation Support*, *(Explicite) Interface Specification*, *Error/Exception Handling*, *Robustness*, *Suitability for Rapid Prototyping* and *Framework Sustainability and Maturity*.
- What kind of cognitive vision functionalities like *Suitability for Binary Data Transfer*, *Programmatic Coordination*, *Data Management Facilities* are of importance?
- How high is the priority of easy handling in terms of *Usability / Learning Curve*, *Ease of Modification*, *Active System Introspection*, *Integration of Legacy Code*, and *Available Documentation*.

A common trend that can be derived form the evaluation is that more sophisticated frameworks also provide more support for system integration and cognitive abilities, but also their handling becomes complex. Independent from each other this was the motivation for the authors to design their own frameworks AMI/XCF 3.6.1 and zwork 3.6.2. Of course we also carried out an evaluation before spending the enormous effort in creating a new framework. Especially providing Usability, Robustness, Evaluation Support and according Documentation are expensive. Hence we would be very pleased if this report, the identified requirements and the developed evaluation scheme support the framework selection process and provide a valuable introduction for the community. We already expressed this wish by a publication at the ICPR [33]. Because cognitive vision is a very young discipline also the supporting frameworks are under progress and an according discussions on requirements and their fulfilment is appreciated.

5 Conclusion

This report identified functional and non-functional requirements for the integration of cognitive computer vision systems. These core requirements not only represent the requirements from cognitive vision but also take into account experiences from previous integration projects and insights from software engineering. We presented an evaluation scheme for fast and straightforward assessment of the important demands on software frameworks for cognitive vision systems. Ten frameworks representing five different origin categories are evaluated. A final comparison guides the CVS developer in selecting its most appropriate one and presents a starting point for further research and development in this domain.

Acknowledgments

This work has been partly supported by the EC Vision Specific Action 13-2 and the IST Projects 2001-32184 ActIPret and IST 2001-34401 VAMPIRE.

[9]A small program typically used to prove a concept or new algorithm.

References

[1] The ActIPret Project, 2005. `http://actipret.infa.tuwien.ac.at/`.

[2] Active Memory Iinfrastructure, 2005. Software and documentation available at `http://xcf.sf.net`.

[3] H. Balen. *Distributed Object Architectures with CORBA*. Managing Object Technologies. Cambridge University Press, 2000.

[4] C. Bauckhage, G. Fink, J. Fritsch, F. Kummert, F. Lömker, G. Sagerer, and S. Wachsmuth. An Integrated System for Cooperative Man-Machine Interaction. In *Proc. CIRA*, pages 328–333, 2001.

[5] H. Christensen. Cognitive (vision) systems. *ERCIM News*, pages 17–18, April 2003.

[6] H. I. Christensen and J. L. Crowley, editors. *Experimental Environments for Computer Vision and Image Processing*. World Scientific Publishing, 1994.

[7] The C++ Template Image Processing Library, 2005. Software and documentation available at `http://cimg.sourceforge.net/`.

[8] J. Crowley and H. Christensen, editors. *Vision as Process*. Springer, 1995.

[9] CVSSP, University of Surrey. Recognition And Vision Library, 2004. http://ravl.sourceforge.net.

[10] Berkely DB XML, Sleepycat Software, 2003. http://www.sleepycat.com/products/xml.shtml.

[11] B. Draper, G. Kutlu, E. Riseman, and A. Hanson. ISR3: Communication and Data Storage for an Unmanned Ground Vehicle. In *Proc. ICPR*, volume I, pages 833–836, 1994.

[12] European Research Network for Cognitive Computer Vision Systems, 2004. http://www.ecvision.info.

[13] G. Fink, N. Jungclaus, F. Kummert, H. Ritter, and G. Sagerer. A Distributed System for Integrated Speech and Image Understanding. In *Int. Symp. on Artificial Intelligence*, pages 117–126, 1996.

[14] R. Fisher. Visual processing software environments, 2004. In CVonline: On-Line Compendium of Computer Vision. R. Fisher (ed). Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/.

[15] J. Fritsch, M. Kleinehagenbrock, S. Lang, T. Plötz, G. A. Fink, and G. Sagerer. Multi-modal anchoring for human-robot-interaction. *Robotics and Autonomous Systems*, 43(2–3):133–147, 2003.

[16] The Joint Research Project Cognitive Vision, 2005. `http://cognitivevision.acin.tuwien.ac.at/`.

[17] ActiVision Tools, MVTec Software GmbH, 2005. http://www.mvtec.com/.

[18] The Internet Communications Engine, ZeroC Inc., 2004. http://www.zeroc.com/ice.html.

[19] K. Konstantinides and J. R. Rasure. The Khoros Software Development Environment For Image And Signal Processing. *IEEE Trans. on Image Processing*, 3(3):243–252, 1994.

[20] C. Lindblad, D. Wetherall, and D. L. Tennenhouse. The VuSystem: A Programming System for Visual Processing of Digital Video. In *ACM Multimedia*, pages 307–314, 1994.

[21] A. Lux. The imalab method for vision systems. In *ICVS*, pages 314–322, 2003.

[22] The MATLAB Image Processing Library, 2005. Software and documentation available at `http://www.mathworks.com/`.

[23] Open Source Computer Vision Library, 2005. Software and documentation available at `http://www.intel.com/research/mrl/research/opencv/`.

[24] W. Ponweiser, G. Umgeher, and M. Vincze. A Reusable Dynamic Framework for Cognitive Vision Systems. In *Workshop on Computer Vision System Control Architectures*, 2003.

[25] A. Rares, M. Reinders, and E. Hendriks. Mapping Image Analysis Problems on Multi-Agent-Systems. Technical report, TU Delft, November 1999.

[26] C. Schlegel and R. Wörz. Der softwarerahmen *martsoft* zur implementierung sensomotorischer systeme. In *AMS*, pages 208–217, 1998.

[27] D. Schmidt. Real-Time CORBA programming with TAO (the ACE ORB), 2003. http://siesta.cs.wustl.edu/ schmidt/TAO.html.

[28] D. C. Schmidt and S. Vinoski. Object interconnections: CORBA and XML, part 1: Versioning, 2003. http://www.cs.wustl.edu/ schmidt/report-doc.html.

[29] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[30] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[31] The Vampire Project, 2005. `http://www.vampire-project.org`.

[32] S. Wrede, J. Fritsch, C. Bauckhage, and G. Sagerer. An XML Based Framework for Cognitive Vision Architectures. In *Proc. Int. Conf. on Pattern Recognition*, volume 1, pages 757–760, 2004.

[33] S. Wrede, w. Ponweiser, C. Bauckhage, G. Sagerer, and M. Vincze. Integration Frameworks for Large Scale Cognitive Vision Systems - An Evaluative Study. In *Proc. ICPR*, volume I, pages 761–764, 2004.

A Assessment Criteria Legend

The scaling is done by 5 different values ranging form the most negative ones like impossible [- -] over cumbersome [-] and neutral [o] to supportive [+] and the most positive one usually meaning sort of automatic [++] support for the required feature. The following lists present a description of the mapping between those values and corresponding meanings used in the evaluation.

A.1 Core Requirements from CVS

Support for User Defined Datatypes: Since Cognitive Vision deals with a big variety of data abstractions, it is impossible to know all relevant datatypes that will be communicated between all the components in preface. Therefore integrative frameworks have to deal with abilities to rely not only on pre-defined but also on user-defined data types. This is exemplified with a review of serialisation support for artifical data structures.

- - - Just standard data types can be transmitted.
- - User controlled transmission of series of messages is necessary to transfer a complex data struct (multiplexing).
- o De/serialisation has to be implemented by the user, e.g. as it is carried out with C++ class definition through overloaded stream operators.
- + De/serialisation for artificial data types is available.
- ++ Automatic optimisation of de/serialisation without any explicite coding necessary.

Suitability for Binary Data Transfer: In contrast to a lot of other sensing modalities, computer vision has to deal with an big amount of data per time. Hence the communication of image data and derivatives like processed images and features should be done in a binary format for performance reasons.

- - - No native binary data transfer at all, e.g. base64-encoded binary data is a text encoding that induces overhead.
- - No native support for binary data transfer; a second communication channel has to be established (e.g. by using a shared memory approach).
- o A binary transfer is established just for specific data types (e.g. image).
- + Native binary support for user defined data types
- ++ Binary data transfer is automatically optimised, e.g. by compression.

Programmatic Coordination: One of the key extensions from vision to cognitive vision is the purposeful, selective control or coordination of processing. This issue becomes even more important if realtime, online or dynamic systems are being developed. Hence the variety of possible component links to compose the system and all tools and templates to manage the control structure are considered here.

- - - Just one fixed program/data flow (e.g. pipe&filter).
- - It enables only acyclic-graph architectures (e.g. no feedback).
- o There are no predefined control features implemented. It has to be build by the component/application programmer.
- + There is a set of predefined control mechanisms available (e.g. central supervisor, yellow page, control pattern, blackboard).
- ++ Several control mechanisms can be artificially combined into one application (e.g. negotiation of agents).

Data Management Facilities: Another important cognitive aspect is the ability to learn and reason. Both require facilities to store, to structure and access data. Especially the flexibility and simplicity of these properties are important properties of a framework for CVS.

- - - It is impossible to store data over time (no global or local memory).

- - There are just local memories without external access.
- o The access to local memory is possible, but has to be done by the component/application builder.
- + There are data base templates and predefined data access mechanisms available (e.g. persistance service).
- ++ Artificial datatypes can be stored, queried and updated over time.

Dynamic (Re)Configuration: The flexibility of component configuration and the external access and control of it are evaluated here, ranging from no configuration support at all on the low end, to highly dynamic reconfiguration support at system runtime on the high end.

- - - There are no configuration options.
- - It enables only static, offline configuration.
- o All related features have to be implemented by the component/application programmer.
- + The component interface provides dynamic reconfiguration.
- ++ Monitoring, profiling and automatic adaption are supported by the framework (design-to-criteria scheduler).

Independence of Architectural Styles: Since the combination of cognition and vision is a relative young discipline, there exists no major trend of how all required operations can be modularised and composed to systems. Hence the flexibility of the framework to construct different kinds of system structures is evaluated by this criteria.

- - - There is only one fixed architecture.
- - The user can implement emulation layers.
- o One family of architectural styles are provided (e.g. hierarchical architecture).
- + Some families of architectural styles are provided.
- ++ No restriction to the communicative and coordination architecture.

Evaluation Support: To enable testing of single components and sub-parts of the system it is very useful to replace architectural components or layers of the system. Hence access to exchanged data and their recording and replacement/replay in the simulation case are properties evaluated by this criteria.

- - - There is no external access to exchanged data.
- - The component builder has to implement evaluation support.
- o The application builder can access exchanged data (e.g. interceptor pattern).
- + It provides profiling of system interactions and provides basic recording/replaying.
- ++ Full simulation of all framework components is supported.

A.2 Distributed Systems Engineering Attributes

Level of Transparency: It is important to disburden the programmer as much as possible from specific, additional effort emerging from distributing components. The level of transparency that is supported in a framework with regard to the construction of distributed systems yields in a good assessment for the extra effort a developer has when distributing processes.

- - - Very specific distribution, no transparency features.
- - Access transparency.
- o Access and location transparency.
- + Access, location and concurrency transparency.
- ++ The above and additional features (e.g. persistance).

Explicit Interface Specification: The benefit of a modular architecture has been proved by software engineering research already decades ago. When a monolithic program is partitioned in modules, interfaces between components have to be specified. Thus interface specification is important for the integration of large-scale software systems.

- - - There are no interface specifications at all.
- - Implicit interface specification (e.g. a plug-in interface).
- o Specification of methods and/or data.
- + Specification & verification of methods and/or data.
- ++ Specification & verification of methods and data and meta data (e.g. properties and timing).

Active System Introspection: This criteria evaluates all facilities provided to access, log, analyse and modify component interaction. This includes communicated data as well as the coordination and control flow of the system.

- - - There is no introspection functionality possible.
- - It has to be implemented by the component/application programmer.
- o Central logging facilities.
- + It supports data and interaction analysis.
- ++ It supports interactive, online data and interaction analysis and modification.

Error / Exception Handling: Where error/exception handling is self-evident for all kinds of software development it is even more important for distributed systems and system prototype generation which is the usual case for CVS.

- - - No possibility to communicate error conditions.
- - All the error/exception communication has to be done by the component programmer.
- o There are framework error messages.
- + The framework supports the specification of artificial user exceptions.
- ++ Automatic handling of non fatal framework errors (error and migration transparency).

Robustness: Where error and exception handling is responsible for detected program inconsistencies, this criteria evaluates the system behaviour if one component crashes.

- - - If one component fails, the system crashes.
- - If one component fails, the system goes controlled down..
- o If one component fails, the rest of the system stays remaining.
- + If one component fails, the rest of the system stays remaining, and the failing component can be recovered.
- ++ Complete failure transparency.

A.3 Non-Functional Requirements

For the core requirements of CVS and the systems engineering attributes a set of direct comparable properties could be specified. In case of the non-functional requirements this is impossible. They are always related to a subjective impression. To enable an integration in the graphical evaluation and an easy comparison between frameworks the are still evaluated within the [- -] to [++] scheme.

Usability / Learning Curve: This criteria reflects all the effort to realize a system. The evaluation considers this effort for minimal applications as well as complex applications. Also several dimensions of the effort are taken into account like algorithmic support, real coding effort, API complexity, any kind of help system and composition support. The characteristic of the Learning Curve gives most objective evidence. Usually the usability is inversely proportional to the amount of functional and supportive capabilities of the framework. Also any kind of graphical representation or development guide assists the programmer.

Ease of Modification: Where the previous criteria evaluates general usability this one focuses on the effort required if the interplay of components need to be modified. Of specific interest is the availability of flexible interfaces, e.g. if it is necessary to recompile the entire system if a single interface has changed. This criteria can also reflect the degree of component coupling in a distributed architecture. In CORBA this is known as the interface versioning problem.

Suitability for Rapid Prototyping: Rapid prototyping is essential at commercial and scientific software development, because early integration cycles are very useful for the evaluation of research directions, e.g. if system performance is conceptually sufficient.

Integration of Legacy Code: This criteria evaluates the amount of work needed to integrate existing modules or entire libraries for reuse in the framework. The assessment is also influenced by the flexibility of the interfaces and the available language binding as well as supported operating systems.

Framework Sustainability: The size and activity of the community is important for further development or framework adaptions on new compiler versions and updated library dependencies. A high rating in this criteria might be important for newly started projects as it gives a bit more confidence in the availability of the integration framework at the end of the projects lifetime.

Framework Maturity: The age of the framework, the number of framework revisions and of course realised projects reflect the maturity of the overall integration framework.

Available Documentation: This criteria is usually underestimated by framework developers. Still it is the key for using the framework. There exists a big list of formats like README, user manual, capability description, API documentation. But also tutorials, examples, templates and newsgroups, mailing lists, FAQ and source code may document the framework. Furthermore, the frequency and date of documentation updates contributes to the assessment of this criteria.