Issues Central to a Useful Image Understanding Environment

J.R. Beveridge, B. Draper
A. Hanson and E. Riseman

COINS TR91-85

November 1991

Issues Central to a Useful Image Understanding Environment

J. Ross Beveridge, Bruce A. Draper, Al Hanson, Ed Riseman

Computer Science Department
University of Massachusetts at Amherst *

September 1, 1991

Abstract

A recent DARPA initiative has sparked interested in software environments for computer vision. The goal is a single environment to support both basic research and technology transfer. This paper lays out six fundamental attributes such a system must possess: 1) Support for both C and Lisp, 2) Extensibility, 3) Data sharing, 4) Data query facilities tailored to vision, 5) Graphics, and 6) Code sharing. The first three attributes fundamentally constrain the system design. Support for both C and Lisp demands some form of database or data-store for passing data between languages. Extensibility demands that system support facilities, such as spatial retrieval of data, be readily extended to new user-defined datatypes. Finally, data sharing demands that data saved by one user, including data of a user-defined type, must be readable by another user.

1 Introduction

Members of the DARPA Image Understanding community are in the process of designing a new software system, the Image Understanding Environment (IUE). Currently both research and technology transfer are hampered by a profusion of different environments, each with their own image formats, etc. The IUE, by standardizing representations where possible, by providing a unified core of basic functions, and by providing an orderly means

*This work was supported by the Defense Advanced Research Projects Agency (via TACOM) under contract DAAE07-91-C-R035 and by the National Science Foundation under grant CDA-8922572.

of extending and sharing representations and functions, should greatly enhance the productivity of the Image Understanding (IU) community.

To meet this goal, we feel the IUE must satisfy the following six key objectives:

Support for both C and Lisp It is a practical reality for the IU community that both Lisp and C must be supported. It should be noted that a principle justification for using C is speed. Therefore only compiled versions of C or C++ should be considered.

Extensibility (both site and application specific) We cannot presume to know or predict the data structures and functions required to support vision research throughout the nineties. We must allow users to modify existing datatypes and to create new ones. Moreover, it is not acceptable to preferentially distinguish between system primitive types and user developed types. A system that is not thoroughly extensible is worse than no IUE at all.

Data sharing One goal is to let us share the fruits of research. The most basic requirement is that of sharing data. Data from one user or site must load into the IUE maintained by any other user or site. Clearly, this means that data generated by an IUE running in C must be readable by an IUE running in Lisp and vice-versa.

Database and query facilities tailored to vision A computer vision program must be able to store, access and manipulate large amounts of data. The IUE must support this with the relevant database technology. Data must be stored for later use, and must be retrievable by name, attribute, or spatial location.

Graphics Visual data needs to be interactively displayed. To avoid redundancy and foster compatibility, common mechanisms should be provided.

Code sharing Modules developed at one site should be executable at another. Realistically, this goal may be in conflict with those above. For example, code written in Lisp cannot be run in C. Nonetheless, code sharing, including the ability to interleave Lisp and C modules, should be encouraged whenever possible.

Over the past fifteen years the VISIONS lab at the University of Massachusetts has developed two generations of general low level vision systems, the latter integrating both C and Lisp. Well over fifty computer vision algorithms have been brought up under these systems. Of possibly even greater relevance to the IUE effort, UMass has for the past six years been using and refining an extensible database tailored to the needs of intermediate level vision [Bro89]. Since its inception this database has supported two host languages, C and Lisp.

The suggestion is not that the existing UMass environment is an adequate IUE, it is not. The extension of any current system/environment to one that meets the states goals of the IU community will take considerable care and thought. However, our experience does suggest that an IUE can be built that will satisfy the objectives stated above. Therefore, our aim in the rest of this document is to clarify these goals, their implications, and hence what we consider to be a minimum standard of performance for the IUE.

Finally, we did not originally intend for this document to stand alone and we focus the majority of our effort on laying out the design objectives which we feel should constrain any proposed IUE system architecture. To fend off doubts that the objectives are unattainable we include a system design section laying out the essential attributes of a system able to meet our stated objectives.

2 Elaboration and Implication of Key Issues

2.1 C and Lisp

The mandate of the IUE is to support the image understanding community, a community that spans a wide range of industrial and academic laboratories. At one end of this spectrum are application-oriented labs. These labs are interested primarily in efficiency, and would not consider programming in any language but C. At the other end of the spectrum are scientific labs that are developing an extremely diverse range of algorithms and subsystems. Some of these labs are committed to the rapid prototyping abilities of Lisp, and they would not consider programming in any language but Lisp. Therefore there is no question about

what language the IUE should support. It must support both Lisp and C.

A detailed discussion of the problems with integrating C and Lisp is beyond the scope of this overview. Nonetheless, one aspect of the clash between Lisp and C is so basic that it has implications for the rest of the design: Lisp code cannot be easily converted to C code and vice-versa. There is a barrier between Lisp and C that code cannot cross.

To be more specific, C code will not run on a Lisp machine. Lisp cannot be called from C without bringing in a large environment that destroys the performance of the C system. This simple fact torpedoes the dream of complete code sharing. Code developed on a Lisp machine cannot be used by an installation running C, and vice-versa. Code sharing can only happen among users with a degree of commonality (see next section).

The C/Lisp barrier also has implications for object orientedness. The fundamental idea underlying the object-oriented paradigm is that data and function should be closely coupled. Indeed, in a pure object-oriented system data and function are inseparable. It is not meaningful to discuss one apart from the other.

Unfortunately, code cannot cross the C/Lisp barrier. If the IUE is strictly object-oriented, then its objects will include code as well as data. As a result, no IUE object will be able to cross the C/Lisp barrier. Data sharing among IU labs would be sacrificed in the name of strictly enforced object-orientedness.

Instead, the idea that the IUE is strictly object-oriented should be dropped. The IUE can still retain many of the aspects of object-oriented programming. For example, functional inheritance can still be provided. The pointers linking data object to the functions that operate on them can also be supplied. However, data objects must exist and be able to be written, read and manipulated distinct from this function so that they can be transmitted across the C/Lisp barrier. (Note: C++ is not a strict object-oriented language. It allows code and data to be stored separately. As a result, proponents of C++ may find nothing unusual in this arrangement.)

2.2 Extensibility

At the risk of stating the obvious, the IUE must not limit itself to efficiently supporting only those data structures, relations and functionalities in common use today. It should be assumed from the outset that new and significant vision algorithms often utilize new and effective data representations. Therefore, the IUE must support user defined data structures and relations.

Although this may seem obvious, some of the implications are less so. One implication is that the IUE system architecture must not distinguish in a prejudicial manner between user defined and 'system' defined datatypes. The IUE must avoid architectural short cuts which provide efficient support for fixed representations, but no mechanisms for defining comparably efficient functionality for new types. Let's illustrate with a simple example. Presume there is a mechanism for fast spatial access to a set of data. The underlying mechanisms used to support such retrieval for system default types must be available at the user level in order that comparable functionality may be provided for new types.

In addition to defining new datatypes, the IUE must permit users to define and modify relational hierarchies. For example, a geometric modeling system might impose a part-of hierarchy on geometric data of the following form: vertices, lines, faces, volumes and objects. The hierarchy presumably places vertices at the lowest level, with each vertex being "part-of" a line. Similarly, lines are parts of faces, etc. For the sake of illustration, consider just the lines. They most likely also participate in relational hierarchies besides the part-of hierarchy just mentioned. For example, if they are specializations of a general class of 2D line segment, they belong to an 'is-a' hierarchy. In addition, there might be one or more relational networks defined over the lines. For example, proximal lines within a single object might be linked in a graph. The point of this example is to show that IUE data objects will often participate in several distinct relational hierarchies. Moreover, it is hard to imagine how these relationships can be defined independently of a particular system. We believe the IUE cannot hope to provide all the necessary hierarchies a priori. However, with the proper tools, new hierarchies may be defined and used with comparative ease.

Finally, we realize that no one is arguing for a closed architecture that prohibits extension.

Rather, if there is disagreement, it is over approach and degree. In particular, there appears to us to be two distinct views underlying the current discussion of the object hierarchy. Our view is that it is impossible to define a single object hierarchy which will satisfy the diverse needs of the IU community. Therefore, our concern is that the IUE do all it can to facilitate the development of new hierarchies appropriate to particular tasks. Moreover, in order to permit integration of independently developed systems, we wish the IUE to be very good at coupling together modules built using different internal representations. Our view seems to differ considerably from others who appear to see a common object hierarchy as guarantor of system wide compatibility.

A fixed hierarchy would be suitable if the task were to build a large system in which well understood and mature algorithms could be brought together. However, this is not the goal of the IUE. The IUE is supposed to support both research and technology transfer. It must therefore expedite the development of new datatypes in conjunction with new algorithms. Moreover, if it is to foster system integration, it must not do so through reliance on a fixed set of representational forms and their specializations. Instead, it must actively support integration between modules using different data types. Consider a concrete example: Algorithm A extracts line segments and algorithm B groups them. Each uses its own specialized form of line segment with properties particular to its own needs. In a strict sense, the two are incompatible because they use different datatypes. However, the information required by B, the position of the endpoints, is present on the lines generated by A. The IUE should make it easy to take the set of lines produced by A and group them with B. Moreover, the algorithms themselves should not need to be modified.

Clearly, there are advantages to standardization. Whenever possible the IUE should foster the use of datatypes drawn from a standard set of libraries. Many of the representational forms used in computer vision are mature and are ready to be standardized. Our point is that if the IUE is to support research then the same mechanisms used to define these standard libraries must be available to users to define their own. Further, the IUE cannot foster integration and hence technology transfer by simply relying upon different modules using the same datatypes. Integration requires mechanisms for coupling modules using different but compatible representations without internal modification of the modules themselves.

2.3 Data Sharing

One of the principle benefits of a common software environment is data sharing. By "data sharing" we mean nothing more than the ability of one program (module, subsystem, etc.) to easily read data files produced by another. When researchers within a lab have been able to read each others data files they have been able to leverage off each other's successes. For example, if one researcher develops a superior line extractor, then other researchers working on problems such as token tracking and geometric matching immediately have better data. The IUE promises to expand this cooperation from a local to a national level.

However, if nation-wide data sharing is to be a pragmatic reality, it must be robust. In particular, data sharing must be insensitive to site-specific system variations and minor differences in object definitions. If one laboratory extends its definition of line segment, for example by adding a time-stamp feature, this should not stop other labs from reading its data, and vice-versa. Needless to say, data files that are written before a site modifies its system should still be readable after the modification there and elsewhere.

Moreover, the ability to share data should not be limited to a few agreed-upon data types (e.g. regions, lines...). All types of data used in current or future vision research should be sharable. The general principle is that if an IUE at one site can write the data, then an IUE at any other site should be able to read it.

In order to implement robust data sharing, the IUE must 1) store the relevant data definitions in each data file and 2) have the ability to reconcile old data from files with current system definitions. Storing the data definitions inside each data file is a necessity if data files are to be reused, yet alone shared. In an evolving software environment, researchers cannot be counted on to remember the data definitions that were in use in their own lab when a given file was written; getting this information from another lab might well be impossible. To avoid this problem, the data definition must be stored in the file.

The problem of reconciling the old data to the new system definitions remains, however. Two solutions are possible. The first is to supply the system with an intelligent reader. The reader compares the current definition to the one in the file and arbitrates any conflicts. In practice this means discarding data in the file that is not included in the current definition,

and supplying default values for features not specified in the file.

A better solution is to provide the IUE with dynamic data types. Data files can then be read "as is". The application program tests for the features it needs, adding or deleting attributes and signaling an error if a required piece of information is missing.

It should be noted that nothing discussed above is beyond the capabilities of either the C or Lisp languages. Everything just mentioned, however, goes beyond the rudimentary object facilities provided by C++. In particular, the IUE will need run-time access to data definitions both for writing new files and reading old ones. Since C++ discards object definitions at compile-time, the IUE will have to provide its own mechanisms for defining and manipulating data entities.

2.4 Data query facilities tailored to vision

Computer vision programs manipulate large amounts of data. In some cases this data is highly uniform, as in the case of 5,000 straight line segments extracted from an image. In other cases the data is highly structured, as in a part-of hierarchy associated with an object modeling system. The challenge for the IUE is to support both extremes along with data configurations exhibiting characteristics of both. At the least, the IUE must manage retrieval by name, by attribute and by spatial position. Finally, the IUE must be able to incrementally read and write selected portions of the database.

The first and most obvious form of database retrieval is by name. We are clearly assuming that the IUE database maintains its own dynamic symbol table. As discussed in the previous section, this is the only truly reliable way of ensuring that a file written by one IUE environment can be guaranteed to be readable by another. Large sets of uniform objects, such as line segments, will typically have integer names. For example, line 53 from the 'desktop' image. However, as our example has already implied, we also assume that some objects will have text descriptors as names, such as 'desktop'. Finally, in order to support the types of hierarchical relationships previously mentioned, objects may be placed in a dynamic hierarchy of objects. Hence, the database can distinguish between line 53 of the desktop image and line 53 of the coffee table image by their placement in the hierarchy.

In addition to retrieval by name, which includes by position in the data base hierarchy, the database should provide attribute query mechanisms. For example, return the set of line segments in the desktop image with contrast values greater than 10 or length greater than 50. This is a fairly standard facility for any database, and the IUE is no exception.

Spatial queries are particularly important in computer vision and the IUE will need to provide fundamental mechanisms to support them. For example, it is quite common for algorithms to seek all the line segments that intersect a given segment, or alternatively all the lines with endpoints inside a given region. We expect the IUE to make such queries efficient. Moreover, the spatial query mechanism must be extensible, so that new data types created by the user can be spatially retrieved.

2.5 Graphics

There appears to wide spread agreement in terms of the basic types of interactive graphical support which the IUE must provide. The graphical interface to the IUE must support all the basic types for graphical objects: line segments, polygons, circles, arcs, splines, etc. In addition the IUE must support black and white, color and psuedo-colored images. It is also clear that the graphics interface should be based on an object-oriented architecture.

We are, for the most part, comfortable with what we have been seeing proposed for graphics within the IUE. However, there are several issues, both major and minor, that ought to be kept in mind. Most important is extensibility as it relates to graphics. If useful graphical interfaces to new algorithms are to be developed the graphics system must be open. A user who defines a new type of spatial object should be provided IUE tools which make specifying graphical representations of this object as easy as possible. Moreover, multiple representations for even apparently simple types of object should be anticipated. For example, some researchers currently draw a directed line segment as a line segment with an arrow head placed at one end. Others draw a directed segment as a segment with a tic mark placed to the right of the segment's midpoint. The IUE should not attempt to decree one of these correct and the other wrong, but rather it should provide one as a default, along with the tools necessary to implement the other. Other issues include support for animation

and the persistence of graphical displays.

2.6 Code Sharing

One of the original aims of the IUE program was to facilitate the sharing of code, thus eliminating costly reimplementations. So far in this document we have noted some of the limitations to code sharing, for example the C/Lisp barrier and the need to allow the evolution of distinct and possibly incompatible approaches to computer vision. Nonetheless, the IUE can and should provide facilities which will allow code to be shared between sites.

The idea is that software modules developed at one site should be usable at other compatible sites. By a "module" we mean a self-contained piece of software, such as an edge extraction algorithm or a geometric modeling package. Two sites are compatible if they run the same language and support interchangeable hardware. Obviously, if a module accesses specialized equipment such as a robot hand or a highly parallel processor, that code may not run in an environment without similar hardware.

The software modules will be linked together through the common database. As long as each module acquires its data from database files and writes its results into database files, then linking them together is a simple task; existing computer vision software environments such as Khoros and KBVision demonstrate this. If each module declares the types of data it consumes and produces, the IUE can check at compile-time that the modules are compatible. (For real-time applications in which the modules run in different processes, the intermediate files can be replaced with data pipes. For single-process applications, the database can simply be maintained from one module to the next).

The goal is to interleave modules from different labs. For example, a laboratory might receive a line extraction module from one site, a line grouping module from another site and a line-based model matching module from a third. These modules could then be hooked together through the database. The line extraction module takes an image as input and produces a set of line segments. The line grouping module takes the set of lines and produces several smaller sets of spatially or geometrically related line segments. Finally, the model matching module compares each of the smaller line sets to an object model.

Moreover, if a site has the capability to run both C and Lisp, it shouldn't matter if some of the modules are written in C and others in Lisp. C modules are run in C processes, Lisp modules in Lisp processes. The common database guarantees that data can be transfered from one module to the next. (If the modules use significantly different representations of lines, or if they use a different set of feature names, a small piece of translation code may have to be placed between the different modules.)

This is not to claim that all the problems associated with the C/Lisp barrier simply go away. Code is still language-specific. If a module extends the definition of an object by adding a method to it, then that method is only available in the host language of the module. For example, if the line grouping module were written in Lisp and it added a method to the line segment definition called "midpoint", that method would only be available to other lisp modules. If the model matching module were written in C, it would only get the data associated with the line segments, not the methods. Consequently, if the model matcher required the midpoints of the line segments (and did not calculate them itself), the grouping module would have to store the line midpoints as a data feature, not just supply a method.

3 System Design

The aim of this document is to focus attention on the design objectives of the IUE, not to present a detailed design. Nonetheless, we are confident that the goals set forth above can be met. Part of this confidence stems from our past experience in integrating visual modules. As stated above, research at UMass has sought to integrate modules into complete vision applications for well over ten years. Moreover, throughout that period we have maintained a combined C and Lisp environment and have frequently integrated modules written in the two languages.

Our confidence is further bolstered by our experience over the past six years both developing and using a databse to support system development and integration. Our first ISR databse supported both C and Lisp and for the first time allowed us to easily integrate modules each of which used their own customized data structures. Since that time the database has gone through many improvements. AAI included some of these improvements

when they built a commercial version of the ISR, which is available as part of KBVision. We included still more improvements in our local design of the ISR2, currently running at UMass. One research project using the ISR2 at UMass currently integrates the work of five different projects/theses, plus imported Lisp code (from MIT and SRI) and C code (from USC).

We are not claiming that the ISR2 is a finished product ready for distribution. We know we need to integrate our graphics capabilities into our database. We also know that fixed and unalterable spatial datatypes as used in AAI's version of the database are intolerable. User's must be able to define new spatial datatypes, and to use those datatypes for storing and retrieving data. Finally, we know we need to develop 3D spatial retrieval mechanisms to match our 2D retrieval mechanisms.

Moreover, the IUE debate has focused our attention on the value of object-oriented programming. Currently, the data stored in the ISR2 are not first-class objects. Nonetheless, we believe that a new database, which supports CLOS and C++ objects, 3D spatial retrieval, graphics and user-defined spatial datatypes can be designed. Moreover, this database could become the core of an IUE. Successive layers of support could then be wrapped around this core, supplying browsers, 3D modeling libraries, etc.

4 Conclusion

Before we can agree on the design for an image understanding environment, we must agree on its goals and objectives. Six fundamentals have been presented above: support for C and Lisp, extensibility, unrestricted data sharing, data base and query facilities, graphics, and code sharing. Of these six key issues, three fundamentally constrain the IUE design: 1) support for both C and Lisp, 2) extensibility, and 3) completely transportable data.

An extensible system means that the same mechanisms used to support system default objects are intelligently packaged for the user community. This is the fundamental constraint. When the IUE provides a capability for a default object class it must also provide mechanisms for extending the capability to new object classes. This constraint should not be taken lightly; considerable thought and effort will be needed to guarantee that mechanisms are

both general and efficient. Consider a simple example. We assume the IUE will permit a user to select a line segment with a mouse, and that line segments are a default type of IUE object. Building the code to permit such a query for line segments alone is simple. It will take considerably more thought to design a general mouse query facility which helps users extend this capability to new object classes, and which is equally efficient for both system and user-defined classes.

Data sharing must be absolute. Data written by one IUE system MUST be readable by any other. As we already said, this requirement carries with it fundamental constraints. First, data definitions must be stored with data. Second, the IUE must be capable of reading data definitions and loading data during run time. In short, the IUE data base must maintain its own name space.

Finally, the absolute data sharing requirement, coupled with the need to support C and Lisp, precludes the IUE data base from being object oriented in the strongest sense. In short, data moves easily between C and Lisp, code does not. When objects written in one language are read by another methods associated with those objects will be lost. Programmers are of course free to translate methods by hand.

In our opinion, the best way to satisfy these constraints is to build the IUE around a database. The database will support both C and Lisp, and will simplify code by providing convenient data query mechanisms. More importantly, it guarantees absolute data sharability, and allows software modules to be linked together. Because users are allowed to define their own database objects, the system will be extensible. Once a graphics system is integrated with the database, it will meet all six of the original goals.

References

[Bro89] J. Brolio, Bruce A. Draper, J. Ross Beveridge, and Allen R. Hanson. The ISR: a Database for Symbolic Processing in Computer Vision. *Computer*, 22(12):22 – 30, December 1989.