ON THE SCALABILITY OF VISUALIZATION IN MANUFACTURING

Haage, M., Nilsson, K.

Dept. Computer Science, Lund Institute of Technology, P.O.Box 118, S-22100 Lund (SWEDEN)
T. +46 (0)46 {2223036, 2224304} F. +46 (0)46 131021 E-mail: {mathias, klas}@cs.lth.se

Abstract - *Computer graphics plays an important role in modern engineering of manufacturing systems, both during design using virtual engineering environments and also as part of user interfaces to various machines. Existing and emerging systems today make use of software components, usually providing a graphical view to the user. In manufacturing, 3D graphics is desirable to visualize geometries of equipment and work pieces, sometimes also via small dedicated user interfaces. The established industrial technology does, however, neither scale down very well to such small platforms, nor do they scale up to safe operation of large systems. We put forward a notion of executable visualization and propose a solution based on the Java platform, using Java3D for 3D visualization in combination with VRML for external representation. Fully implemented prototypes including both real and virtual industrial robots, and industrial case studies, have verified the scalability which appears to be unique.*

1. Introduction

Virtual manufacturing environments are emerging to meet the demands of rapid planning and reconfiguration in production systems. That in turn is driven by the need to rapidly respond to product changes, thereby achieving shorter time to market and increased profitability.

The similarities between factory automation and both virtual reality and enterprise computing have been observed [19]. There are, however, also important differences. In the world of factory automation, as well as in other areas such as autonomous systems etc., we have to cope with the differences between the real and the virtual world, not only during creation but also frequently during tuning, operation, and maintenance of the system. This imposes additional requirements on the user interaction, which may take place first in an engineering environment and later in a production environment (using the same software component). Furthermore, software components may be used for, or tightly together with, the control of physical machines. In order for this to scale up to large, as well as down to small systems, we will first have to find a sound software technology. Secondly, we need an appropriate technique for supporting graphical user interfaces. As the most challenging case, we focus on the need for 3D graphics, which clearly applies to many programmable machines such as industrial robots.

We propose the notion of *executable visualization* graphics as a term for the encapsulation of graphics and renderer together in a software component. The immediate advantage is that one may use a high-level graphical description language but still be able to render on systems providing only low-level rendering capabilities. Other advantages are customized navigation and animation capabilities, easy management and customization of graphical descriptions, possibility to use template software components for creation of a whole class of visualizations, and construction of visualization components for heterogenous environments. We believe this approach will be highly useful when dealing with small application specific visualizations.

After presenting some related approaches, we will first look at the issue of scalability and its implications for manufacturing software. That points out some unsolved issues concerning software components and graphics. The main part of the paper is the subject of executable visualizations. A prototype implementation using Java technology is presented followed by a discussion with application examples. Finally, conclusions are drawn.

2. Related work

Web visualization is a field which is in the beginning of its development. This work has largely been inspired by the efforts of Dr. Mikael Jern to create componentbased web visualizations [18]. Visualizing medical data on the web is a problem because most medical examinations produce huge amounts of data. The low bandwidth of Internet produces the need for data reduction techniques reducing the amount of data being sent to the web visualization client. Dr. Jern is considering client-server solutions based on intelligent component clients containing rendering, custom navigation, and visualization functionality to assist the user in searching the dataspace while minimizing the data transfer rate.

The German Institute of Space Flight lead by Dr. Hirzinger has made an early attempt to use web-based visualization for telepresence [17]. Their group has developed virtual robots used for online prediction of robot movements in teleoperating environments.

Most notable within manufacturing visualization is the field of digital manufacturing, creating large scale visualizations covering entire plants. The production shall be suited to the product: the goal of digital manufacturing is to connect manufacturing control to the product planning process. By simulating the entire production process in a virtual environment shorter product cycle times are achieved, as well as lowered costs, guaranteed quality and shorter product-to-production time spans [20].

Robot visualization is being used in simulation and control software for robot manufacturing [26]. The field of digital manufacturing rely on plant visualization [20]. Visualizations are put to use in various situations including control, monitoring and offline programming tasks. The typical visualization is part of a large system running on a dedicated workstation. However, the Internet has created a demand for light-weight visualizations capable of running on low-end, low-cost personal computers to form the backbone of new types of user interfaces.

3. Scalability

Within automation, there is currently a clear trend towards creating application software by graphically composing available software components. The issue now is to select the most promising approach to accomplish the concept of executable visualization.

Components used in industry today are mostly written in C/C++ by programmers well acquainted with the 'restart-the-computer culture' which they have learned to accept; believing in the utopia that you will finally find that last error. Of course finding all faults can be done in theory, but in practice the use of an unsafe language like C/C++ implies that the engineering effort is too high. This means, for example, that even if only one out of 50 used components at some time contains a bad pointer (due to manual memory management) or an array index out of bound, that can affect data which in turn may cause the entire application to crash.

As applications get larger and more complex, and considering that components are more frequently used in safety critical application (such as hazardous chemical processes), we need to worry about safe and dependable operation. That involves several issues such as redundancy, supervisory control, error handling, etc. But more logically, to make sure those features really work, we need to ensure proper program execution. This implies that the use of unsafe languages, for other than well restricted/encapsulated local interfaces or drivers, will have to be abandoned for control systems. This is a necessary but not sufficient condition for safe operation.

For a language to be called safe, we use the definition that *all possible executions are defined in terms of the language itself*. This implies, for example, that it has to be abandoned to: use absolute memory addresses, create dangling pointers, index outside an array, cast-away type checking, or reference uninitialized memory. If any of this would be allowed, execution can result in something that is neither expressible as a program nor desired. We talk about core dumps, 'blue screens', and the like. A program written in a safe language can also crash, but only in a controlled way; for instance by throwing an exception to the invoking application, which cannot be damaged by illegal memory access. Instead, measures can be taken to manage the application in an appropriate way.

When it comes to the actual control of industrial processes and manufacturing equipment, special care is needed to obtain real-time performance, and also to maintain operator interaction on the factory floor. Automatic memory management, or garbage collection, which is part of the Java program execution and a cornerstone of the scalability of Java, is often referred to as an obstacle for real-time performance. That is, however, not true. In our group it has been proved in theory, and demonstrated in practice on a real industrial robot, that well designed automatic memory management works fine and predictable even in hard real-time systems [23].

Encouraged by these results, and realizing that Internet and enterprise computing techniques are applicable even down to the field-bus level [21, 24], we have focused on operator interaction and graphical user interfaces which play a key role in programming, configuration, and operation of manufacturing equipment. As the most challenging case, we consider industrial robots and the need to handle description and presentation of geometries. We then need a technique that provides both scalable/safe operation, and visualization that scales well from powerful workstations down to dedicated devices on the factory floor.

The natural choice today is the Java language. Java has already made its way into enterprise computing, and since the same requirements show up in factory automation, Java appears to be very well suited for the task. Thus, we try to use Java for its safety, which is required to obtain scalability, and history has taught us that in the long run it is the scalable techniques that survive.

4. Executable visualization

We would like to put forward the notion of executable visualization as a term for software components containing a graphical description and customized code to render the description.

4.1 Four aspects of usability

A hard coupling between rendering and description provides a number of possible advantages for the user such as customized graphical descriptions, customized rendering, self-contained graphical descriptions and platform-independent execution and behaviour.

Customized graphical descriptions. A problem that has existed during a long time, but has exploded with the introduction of Internet, is the large number of existing file formats. It is not feasible to equip each computer with readers for every file. One solution to this problem is to introduce generic file formats and have generic file viewers. This is the normal approach on the Internet today (Adobe Public Document Format for WEB publishing, HTML for browsing and VRML for 3D graphics). This solution is, however, not feasible for specialized situations needing customized functionality. The normal approach is to develop specialized software tools. We propose another way; by focusing on the data and enhancing it with custom functionality we achieve a essentially self-contained data format. It will be possible to directly export CAD geometry with enhanced functionality without worrying if the target computer has the ability to render the enhanced CAD format.

Customizable renderers. The property of customization is important as it allows executable visualizations to be customized for special tasks, with special demands on navigation, control and feedback functionality. An executable visualization has the ability to modify its renderer to incorporate customized behaviour, for instance to enhance a static graphical description with animation capabilities, to provide customized navigation capabilities for user interaction, and to incorporate external control interfaces. It is to be expected that demands on functionality will vary extremely depending on situation.

Software component packaging of graphics. A system might be heterogenous from several different points of view: different processing power and memory capacities, different capabilities for visualizing graphics and different platforms. An executable visualization should be able to render and produce reliable results in a heterogenous environment. Our prototype implementation achieves platform- and environment-independence by utilizing the Java and Java Beans component technology.

Template software components. Using the notion of executable visualization, it will be possible to speed up development of similar visualizations by creating a template software component containing a customized renderer and use it with all graphical descriptions. For instance, it will be possible to create a renderer providing custom functionality for rendering robot geometry and use this renderer to create executable visualizations for a whole product range of robots.

4.2 Approach to building

When building executable visualizations established technologies should be used as much as possible in order to achieve rapid development, low maintenance, and high platform independence. We therefore propose

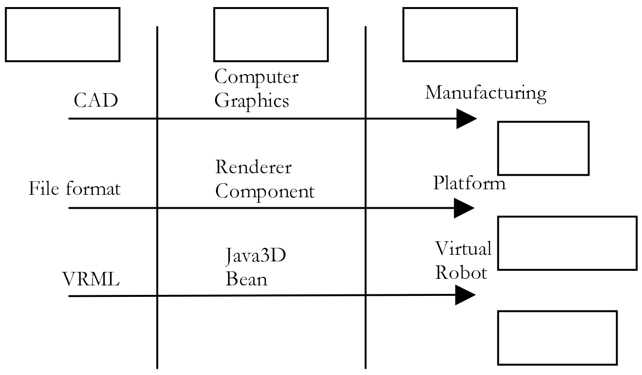


Figure 1 A staged approach for the creation of an executable visualization with a manufacturing perspective. Left: product development, Middle: graphics, Right: manufacturing.

an implementation in three stages as shown in Figure 1.

The first stage consists of a graphical description of the visualization, for instance CAD geometry resembling an industrial robot. It is important that the file format of the description in this stage conforms directly to the source of the graphical descriptions. In our robot example, we want to use CAD geometry also for the robot that is to be used in the manufacturing task. The executable visualization should store the geometry in a CAD format, for instance VRML.

The second stage consists of the renderer together with customization code. The available range of renderers today goes from API-accessible renderers to pure standalone rendering applications. A problem is the customization code. Customization may potentially be provided through three sources; through changes to the graphical description file, through customization by developing a renderer based on a rendering API, or through customization of a standalone renderer. That is, there are three approaches:

The first approach involves a modifier that annotates the description file with customization code. This demands, however, that the language used to express the graphical description contains the power necessary to provide demanded functionality. Most CAD systems of today are able to export static geometry to VRML. Built into the language of VRML is the possibility of script-driven animation, something CAD systems do not utilize. The creation of a moving robot from a static robot model could, in the second layer, involve the annotation of the VRML description with scripts driving an animation. A standard renderer of VRML might then be invoked. As a counterexample, VRML is not able to express all kinds of functionality. We might want to express a customized navigation capability like semantic zooming, which is a technique

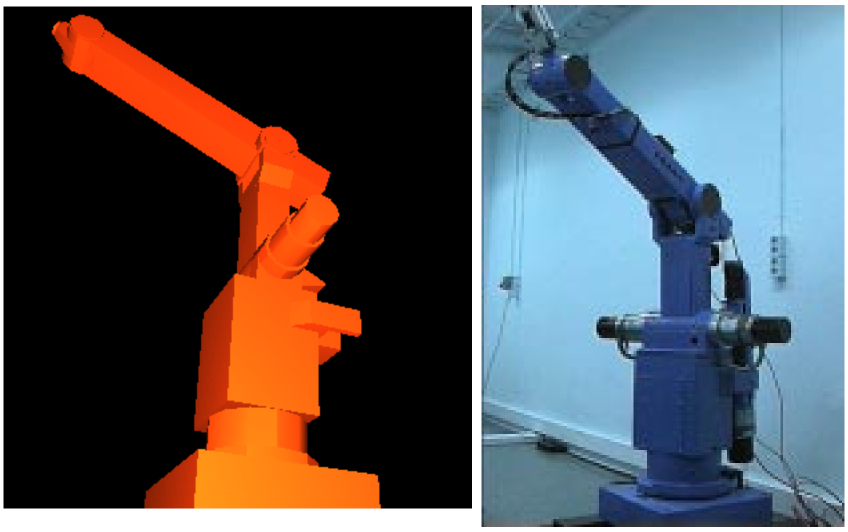


Figure 2 Executable visualization showing virtual ABB IRB6 robot to the left and the real robot available in our laboratory to the right.

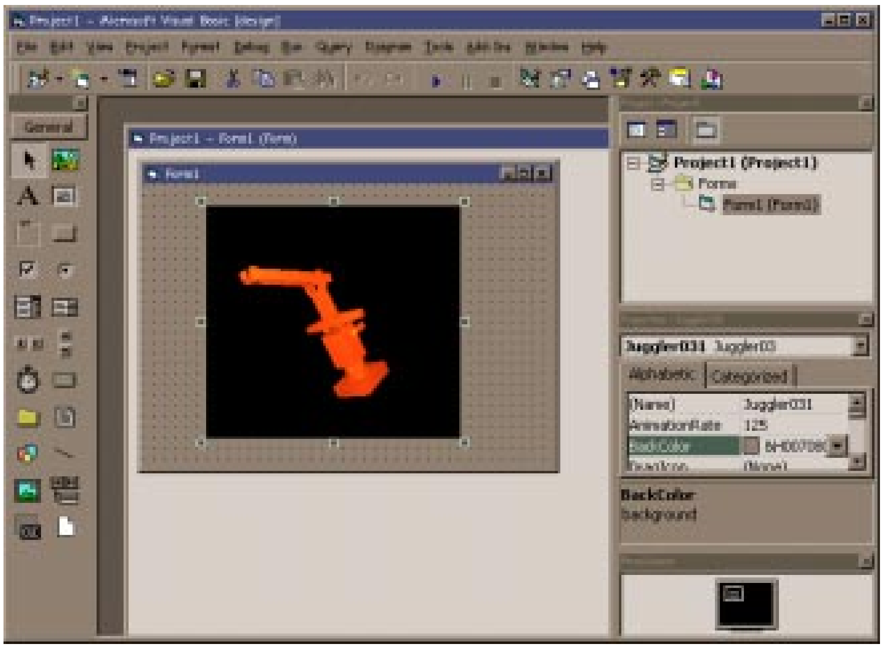
for chosing the wanted detail level in a visualization. In the VRML this might only be achieved with great difficulty because the language lacks constructs for handling such functionality [16].

The second approach relies on the availability of a rendering API, preferably able to read the file format to be rendered. The advantage of having a renderer available through an API is that it provides a high degree of freedom for customization; you are essentially free to develop your own custom renderer on top of the API. The VRML workgroup has developed a VRML rendering API running on top of Java 3D, a software package available for the Java 2 platform [2, 3, 4, 14].

The third approach uses standalone renderers with customization ability. The Cosmo player [12] (a VRML viewer) uses a link called the *External Authoring Interface* [13] to enable the Java language to connect to the viewer and affect the VRML model.

The third stage encapsulates the visualization into a common component technology. This enables the executable visualization to be incorporated as a component into the application, with a component tool like Microsoft VisualBasic and Java Beanbox.

The conclusions to be drawn from this section is that the implementation of executable visualizations as we propose them will not state a problem. However, the preferred way to do it is the second approach, using a rendering API, which provides most customization power. Some experiences from using that approach now follows.



(Name) Juggler031
AnimationRate 125
BackColor &H00708C

Figure 3 Executable visualization as an ActiveX component in Microsoft Visual Basic, using a Java Beans-ActiveX bridge.

5. Implementation

Component technologies such as ActiveX, JavaBeans, CORBA, COM, DCOM provides the infrastructure upon which executable visualizations may reside as natural extensions. Tools for dealing with components are quite common [10, 11]. There exists a lot of renderers (Java 3D, Cosmo, OpenInventor, OpenGL++), converters between different file formats, and a few neutral file formats for the exchange of geometry between different systems. As mentioned, have chosen to use VRML.

The core Java 2 platform from Sun Microsystems Inc. may be extended with a package called Java3D also available from Sun. Java3D is an API capable of rendering high-level 3D graphical descriptions onto OpenGL and DirectX platforms. Java3D is closely related to VRML. In fact, Java 3D was designed to allow easy transitions from VRML to Java 3D. By using Java 3D it is possible to encapsulate VRML graphics with a VRML renderer into a component and customize the rendering through the Java language. We have utilized this for creating virtual industrial robots available as Java Bean components or ActiveX components, executable in a number of user interface environments such as Visual Basic, Internet Explorer (HTML) and Java standalone application. Most notable is that the component due to Java runs in these different environments without modification.

5.1 An industrial robot implementation

Our prototype implementation of an executable visualization for visualizing industrial robots. The prototype is based on the Java 3D renderer enhanced with VRML. The robot is encapsuled as a JavaBean component and is able to run in an ActiveX environment through a ActiveX-JavaBeans bridge, see Figure 3.

The prototype expects static geometry in VRML as directly exported by one specific off-line programming system[1], but should handle VRML exported from other systems with no problem. Our example robot (ABB Irb-6) is shown in Figure 2. The VRML describing the robot is annotated with named nodes in order for the

component to recognize rigid robot parts among the geometry, see Figure 4.

At the initial stage of our project there were no VRML loaders available for Java3D. However, several efforts are being made to create VRML loaders, the most notable being the formation of a Java3D and VRML Working group within the Web3D Consortium[2]. As we saw that several implementations were on their way but were not quite ready when we needed them, we wrote a simple tool, translating VRML geometry into its Java3D equivalent. This was easily accomplished as most CAD systems only utilize a small subset of VRML when exporting geometry. The disadvantages of this solution is that we do not retain the original file format within the component and we have to recompile the component in order to change geometry.

```
#VRML V1.0 ascii
Separator {
DEF IRB-6 Separator {
        DEF IRB-6:IRB_6-0 Separator {
            MatrixTransform {
                matrix 1 0 0 0
                       0 0 -1 0
                       0 1 0 0
                       0 0 0 1
            }
            Separator {
                Material {
                    diffuseColor 1 0.38 0
                }
                Coordinate3 {
                    point [
                        0 0.28125 0.17,
                        -0.27 0.28125 0.17,
                        -0.27 0.225 0.17,
                        -0.27 -0.225 0.17,
```

Figure 4. VRML geometry exported from the IGrip system. Note the named node IRB-6. That node represents a rigid body part of the robot.

The renderer has enhanced the original static view with animation capability. The robot is able to move its individual joints, according to data supplied at runtime. This is accomplished by "hooking" the identified rigid robot parts onto a linked structure of Java3D transform objects. Finally, the resulting objects are made into a JavaBean and transformed into an ActiveX component using the available JavaBean-ActiveX bridge from Sun Microsystems Inc.

The prototype is well suited to act as a template component to create similar visualizations for other robots, for instance an ABB Irb-2000 robot, which is also available in our laboratory. The geometry used for creating the robots may be as simple as the VRML geometry available on the ABB product page[3]. A spin-off usage of the component is to visualize motion data recorded from the real Irb-6 robot.

Both the robot models and the software platform (Java 2) are freely available on the Internet.

6. Applications & Discussion

The use of computer graphics in the personal computer market is, as mentioned, affecting the manufacturing market. Soon the use of graphics and particularly 3D graphics is not going to be a nice feature but a demand from the customer. The use of graphics in the manufacturing industry today is mostly concentrated towards large-scale visualization (digital manufacturing) and CAD system design. Since that is more or less established technology, we will now see how this scales

[1] The IGrip System, http://www.deneb.com/

[2] http://www.web3d.org/WorkingGroups/vrml-java3d/

[3] http://www.abb.se/robotics/product/index.html

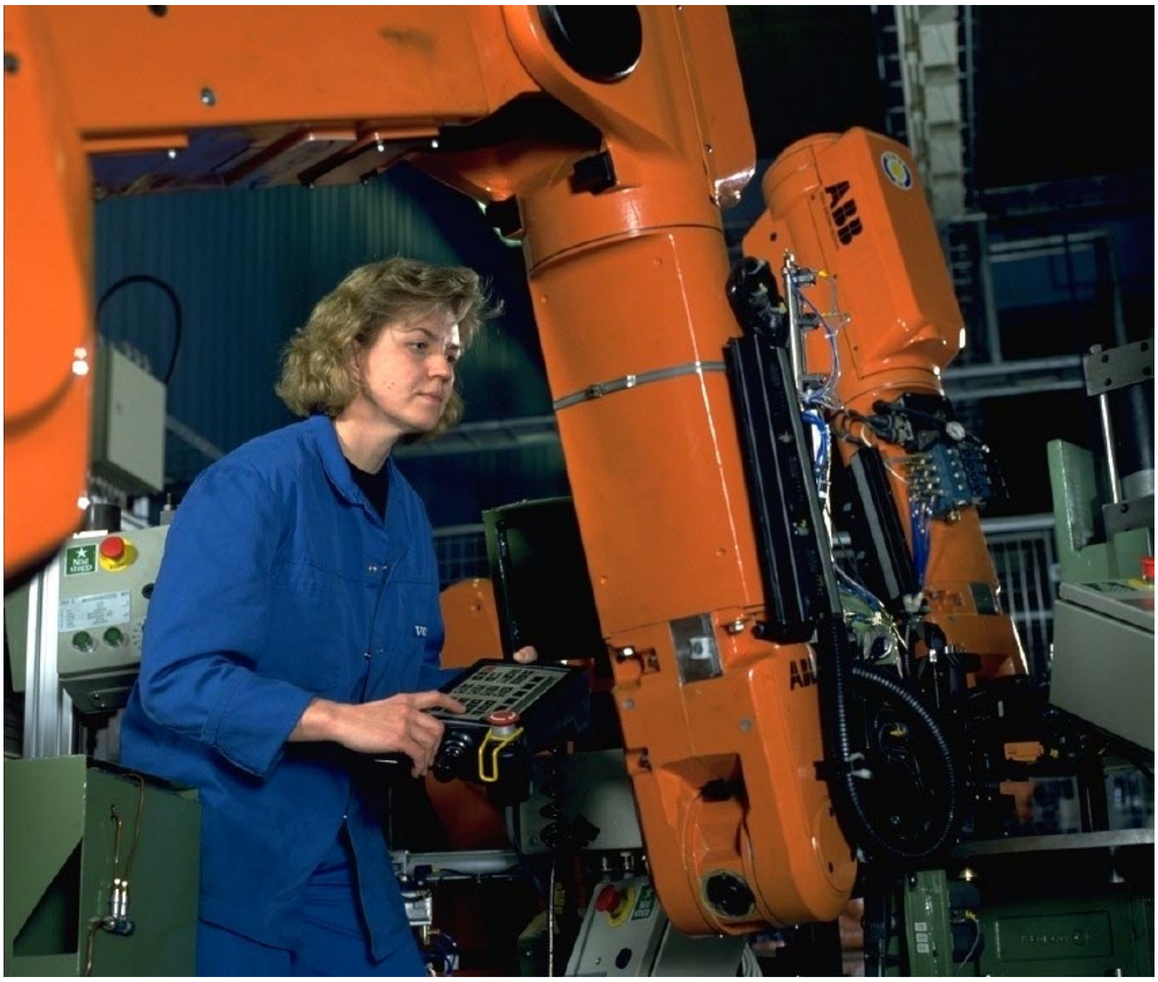


Figure 5 *Robot operator/programmer at Volvo, using a hand-held terminal for on-line changes. (With permission from Volvo and ABB).*

down to dedicated end-user interfaces for use on the plant floor.

Whether or not we need a powerful online interface to a machine or robot very much depends on the manufacturing task. For instance, assembly of circuit boards is in most cases well specified from an CAD/off-line model, which among other things uses a data-base with descriptions of the physical components to assemble. In such case, an initial calibration of the robot relative to some fixtures may be enough, and only a very simple on-line interface is needed.

In other application areas, accurate off-line modeling is much harder, for instance due to unmodeled dynamics of the manufacturing process. This is often the case within large application areas such as welding and deburring [22]. Therefore, such robots are often handled via a small user interface that the operator can carry around close to the manufacturing process, see Figure 5. There is of course also a tradeoff to be done between a hand-held simpler interface and a more powerful interface via, for instance, a PC connected directly to the machine. We leave that decision to the industrial development. Instead, we consider the techniques that can be applied in a flexible manner. An interesting alternative is to have a complete Windows platform even in the hand-held device. That may, however, not be the most efficient solution, but it can in any case be used beneath the principles we propose. As an example, let us consider an arc-welding application.

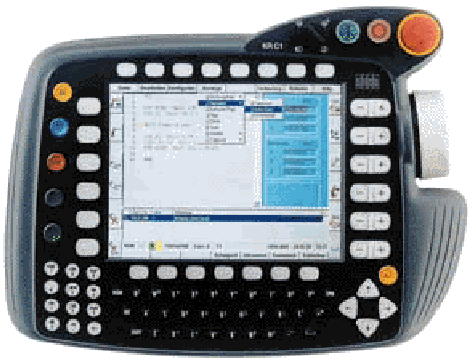


Figure 6 Hand-held terminal from KUKA providing a robot interface on a Windows-95 platform.

6.1 Arc-welding example

Assume manufacturing a product includes, among other things, welding two metal pieces together. Due to tolerances of the work-pieces and the difficulties to exactly predict the outcome of welding operation, there is a need for an appropriate end-user interface by which the production engineer can tune the welding operation. Such tuning may need to be different for different parts of the seam. Furthermore, there is a desire to let the operator adjust the welding in terms of the geometry of the welding seam, rather than on some less understandable voltage or wire-feed parameter. For this purpose, there is a trend towards having knowledge about the welding process stored in databases that can be accesses via the factory network.

To our knowledge, there are no really good such interface today. Instead, we base this discussion on want-lists from production engineers. Given the specifications for the user interface needed for a certain application, one could of course implement it directly in, for example, C++ on the Win32 platform utilizing available ActiveX/DCOM components written the same way. That would, however, impose restrictions on safety and flexibility. Instead, we suggest a Java-based implementation. To further explain our approach, each of the input sources depicted in Figure 7 will now be reviewed.

The purpose of the CAD data input is to obtain up-to-date coordinates on which the machine/robot operations can be defined. For brevity, retrieval of calibrated coordinates back to the CAD system [22] is not treated here. A variety of existing CAD data formats exist today, but many of them are too limited for 3D-description, internal/proprietary, platform specific, etc. Therefore, we have chosen the VRML format; most systems can export the object geometry in VRML. The VRML format is an platform neutral ISO/IEC standard designed for the Internet.

Visualization is today mainly used for off-line programming; in on-line programming the physical equipment and work-pieces are there, so why visualize it? Reasons include:

1. To make referencing and description of coordinates during programming more user friendly.
2. To make monitoring of ongoing machine operations more understandable.
3. To tune and optimize the manufacturing process.

Items 1 and 2 are obvious but let us see what the third item could mean in arc-welding. Examples:

The welding technician may want to study the weld-seam profile along the path, both the CAD-model and the actual work-pieces, and confirm the generated welding settings from the engineering department. Figure 8 shows the end of seam having a small gap.

In case of deviations between the model and the real world objects, there should be a way of calibrating the model, and to simulate the effect of using the welding settings (such as currents, wire-feed, path speed, and weaving amplitude). Since the seam properties changes along the seam, one can easily imagine the benefits of having a customized visualization and navigation along the track. Using Java3D with imported VRML models, in combination with a customized rendering and navigation tool appears to be very useful.

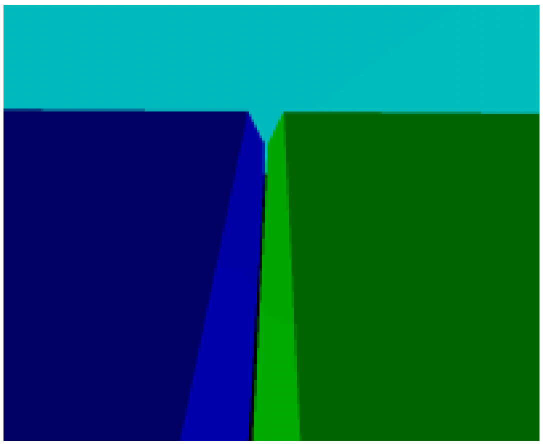


Figure 8 Ending of seam to be welded, from a CAD system via VRML.

The input from the application knowledge database may concern guidelines and rules how different settings effect each other, but also specific rules and restrictions how the welding has to be performed to meet certain quality requirements [1]. Today, such functionality is data driven from tables and special data formats. This limits flexibility and complicates the implementation of end-user tools since parsing and data conversions between different formats often have to be done. Instead, we suggest the use of Java objects, for the same reasons as in enterprise computing. The advantages of obtaining not only data but also methods add a new degree of freedom in the way application know-how can be expressed. Clearly, to have embedded systems running methods dynamically loaded via the network requires a safe language.

The Java technology we use appears to be very flexible and scalable, and systems can be built more easily based on well-designed APIs and software that are freely available on the Internet. The Java objects and beans that make up a scalable application can of course also be compiled or wrapped into current Windows technology for usage in systems available today.

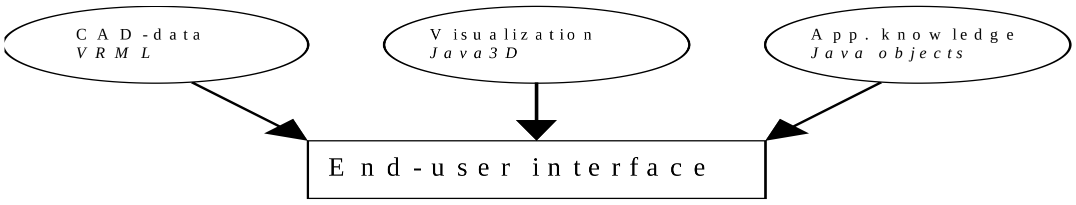


Figure 7 The end-user interface incorporates the geometry, the possibly customized visualization and navigation, and the application know-how

7. Conclusions

The use of computer graphics in user interfaces today poses some problems. Graphical description languages are either too low-level (OpenGL) or lack the expressiveness (VRML) needed for use in user interfaces. The diversity of description languages is another problem that causes compatibility problems between systems. This may be solved by creating customized visualization components containing both graphical description and executable code, what we call executable visualizations. The major benefit of using the notion of executable visualization is that it exploits the large body of CAD geometry to provide cheap visualizations that are easy to create, to maintain and to update by customizing the component rather than the geometry itself.

We have shown on the Java platform that it is possible to create executable 3D visualizations which animates robots exported as static geometry objects from an object library while making minimal intrusion into the robot geometry description, thus providing cheap domain controlled animation to a whole product range of robots. The resulting visualization is packaged as a software component and is directly executable in a Microsoft environment, as well as in a Java environment and a browser environment. Also, the Java platform has the additional advantage of being free software. This makes it possible to freely create and distribute computer graphics in the form of Java software components.

The arc-welding example illustrates the need for this type of visualization in the industry. The need to incorporate domain specific information in user interfaces is essential for advanced control applications, and our prototype implementation shows that the proposed techniques accomplishes that in a feasible way.

Together with the portability and scalability of the Java 2 platform, our approach appears to have unique benefits, which should be of great value within the field of manufacturing.

8. Acknowledgements

This research is supported by NUTEK (The Swedish National Board for Industrial and Technical Development) within the research program for Complex Technical System, grant 97-09713 Open Control Architectures.

Within our university, special thanks go to our colleagues at the Production Engineering department for valuable discussions and CAD models, and to the dept. of Automation Control for sharing the robotics laboratory with us. Boris Magnusson, at our department, is acknowledged for comments on safety and languages.

We are also grateful for the interaction with people at several ABB companies (Corporate Research, Robotics, Flexible Automation, and Automation Systems).

9. References

[1] European Committee for Standardization, *Specification of welding procedures according to EN 288,* B-1050 Brussels.

[2] Sun Microsystems Inc., *The Java® 2 Platform*, http://www.javasoft.com/products/jdk/1.2/

[3] Sun Microsystems Inc., *The Java^{TM} Tutorial*, http://java.sun.com/docs/books/tutorial/ index.html

[4] Sun Microsystems Inc., *The Java 3D$_{TM}$ API Specification*, http://www.javasoft.com/ products/java-media/3D/index.html

[5] Sun Microsystems Inc., *The Java Beans 1.01 Specification*, http://www.javasoft.com/beans/ docs/spec.html

[6] Sun Microsystems Inc., *Using JavaBeans with Microsoft ActiveX Components*, http://www.javasoft.com/products/plugin/ 1.2/docs/script.html

[7] Chen, W., *ActiveX Programming Unleashed*, Sams Publishing, 1996.

[8] Bargen, B., Donnelly, P., *Inside DirectX*, Microsoft Press, 1998.

[9] Eddon, G., Eddon, G., *Programming Components with Microsoft Visual Basic 6.0; Second Edition*, Microsoft Press, 1998.

[10] Pattison, T., *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0*, Microsoft Press, 1998.

[11] Verbowski, Chad, *Integrating Java and COM*, Microsoft Corporation, Jan 1999, http://www.microsoft.com/java/resource/ja va_com.htm

[12] PLATINUM technology, inc., Visual Computing Group, *Cosmo Player*, http://cosmosoftware.com/

[13] PLATINUM technology inc.., *The External Authoring Interface (EAI)*, http://www.cosmosoftware.com/products/ player/developer/eai/

[14] VRML Consortium, *The Virtual Reality Modelling Language ISO/IEC 14772-1:1997*, http://www.vrml.org/Specifications/

[15] VRML Consortium, *X3D Market Requirements*, http://www.vrml.org.

[16] Tamiosso, F. S., Raposo A. B., Magalhäes, L. P., Ricarte, I. L. M., *Building Interactive Animations using VRML and Java*, Proceedings X Brazilian Symposium on Computer Graphics and Image Processing, IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.

[17] Hirzinger G., Brunner B., Koeppe R., Landzettel K., Vogel J., *Teleoperating space robots. Impact for the design of industrial robots*, ISIE ’97. Proceedings of the IEEE International Symposium on Industrial Electronics. IEEE, New York, NY, USA; 1997; 3 vol. 1635 pp. p.SS250-6 vol.1.

[18] Jern, M., *3D data visualization on the Web*, Proceedings 1998 MultiMedia Modeling.MMM'98 (Cat. No.98EX200). IEEE Comput. Soc, Los Alamitos, CA, USA; 1998; x+231 pp. p.90-9.

[19] Atherton, R., W., *Moving Java to the Factory*, IEEE Spectrum, december 1998.

[20] Krusell, P., *Den digitala drömfabriken (in swedish; eng: The ideal digital factory)*, PLASTForum, nr 3, 1998, Sweden.

[21] Lumpp, T., Gruhler, G., Küchlin, W., *Virtual Java Devices; Integration of Fieldbus Based Systems in the Internet*, IECON '98, Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.98CH36200), IEEE, New York, NY, USA; 1998; 4 vol. xxix+2635 pp. p.176-81 vol.1.

[22] Nilsson, K., *Industrial Robot Programming*, PhD thesis, Department of Automatic Control, Lund Institute of Technology.

[23] Henriksson, R., *Scheduling Garbage Collection in Embedded Systems*, PhD thesis, Dept. of Computer Science, Lund Institute of Technology, July 1998, LUTEDX/(TECS-1008)/1-164/(1998).

[24] Entezarjou, I., *Internet- och javateknik för fältbussystem (in swedish; eng: Internet and Java technology for Fieldbus systems)*, Dept. of Computer Science, Lund University, November 1998, LUTEDX/(TECS-3081)/1-76/(1998)

[25] Braun, R., Nielsen, L., Nilsson K., *Reconfiguring an ASEA IRB-6 robot system for control experiments.*, report TFRT-7465, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, October 1990.

[26] Deneb Robotics, Detroit, USA, *IGRIP Users Manual*, 1995. IGRIP is a registered trademark of Deneb Robotics Inc.