Advanced Scientific Visualization, a Multidisciplinary Technology Based on Engineering and Computer Science

Dean Vucinic
Department of Mechanical Engineering, Vrije Universiteit Brussel
Pleinlaan 2, B- 1050 Brussels, Belgium
dean.vucinic@vub.ac.be

Abstract *– Today's visualization tools are equipped with highly interactive visual aids, which allow analysis and inspection of complex numerical data generated from high-bandwidth data sources such as simulation software, experimental rigs, satellites, scanners, etc. Such tools help scientists and engineers in data extraction, visualization, interpretation and analysis tasks, enabling them to experience a high degree of interaction and effectiveness in solving their design problems, which become more and more complex day by day. As the variety of today's visualization tools is diversifying, there is a need for their simultaneous use within different engineering software when solving multidisciplinary engineering problems. It is evident that such tools have to be available for a combined use, in order to eliminate many well known problems of sharing, accessing and exchanging design models and the related information content. It is shown that Object-Oriented methodology is a well adapted approach to stream the software development process of future engineering applications. The three European projects ALICE, LASCOT and SERKET are given as examples in which the evolving computer software technologies have been researched and demonstrated to address the evolution of the visualization software in engineering and for information visualization in general.*

Keywords *– scientific visualization, object orientation, multidisciplinary engineering*

1. VISUALIZATION SOFTWARE

Scientific visualization (SV) [1] is performed through specialized software [2], which combines visualization techniques to display and analyze scientific data. The scientific visualization methodology defines methods to manipulate and convert data into comprehensible images [3]. The scientific visualization process starts with transformation of data sets into geometric abstractions, which are further processed in displayable images, created by computer graphics algorithms [4]. Finally, human vision, possessing the highest bandwidth of human's information input, is exploited to understand the computer generated images.

In order to develop SV Software it is necessary to combine advanced Computer Graphics (CG) and User Interface (UI) technologies with engineering content. Thus, we need to consider and integrate the mentioned methodical domains, when addressing the software development of SV tools, as part of an integrated computational environment, see Figure 1, in order to efficiently support scientists/engineers at their work in the research laboratories and industry. In industry, visualization is used to gain a more quantitative understanding of the simulated phenomena (ex aerospace product design). The results of visualization are also used in management and commercial presentations. In contrast to industry, in a research laboratory, scientists develop codes and try to understand qualitatively how simulation algorithms behave. In this context, they tend to use SV as a debugging tool. In both cases, the computational environment includes software that supports a geometrical definition (as in CAD systems), mesh generation (pre-processing), supervision of the simulation (co-processing) and display and/or analysis of results (post-processing).

Interactive visualization accelerates the problem solving design cycle by allowing the user to 'jump' at will between the various phases, so as to optimize his/her analysis tasks. The user conducts an investigation in a highly interactive manner; he/she can easily compare variants of a simulation/analysis and may intuitively develop a deep understanding of the simulation and of calculation details. An example of an integrated environment application is the 'Virtual Wind Tunnel' [5], which reproduces a laboratory experiment in a virtual reality environment, where a virtual model can be created and put to test with dramatic cost and time savings compared to what is done in the 'real' laboratory.

SV software has progressed enormously during the past two decades. One reason is the exponential increase of the computer processing power, which has led to today's low-cost PCs clusters, providing as much

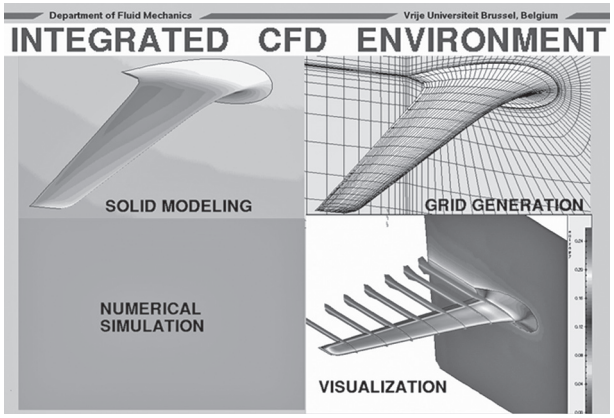

Fig. 1. Integrated Computational Environment

power as the high-end mainframes of some years ago. Development of advanced SV tools is no longer the prerogative of specialized labs with costly computer equipment. Yet, there is an undiminished demand for new visualization-enabled software, driven by continuous hardware changes and the emergence of new software platforms. Interactive visualization remains a key element of advanced engineering/scientific software, and their design must account for this fact. There are presently many commercial interactive visualization products on the market which provide SV functionality with increasing success. Such visualization systems are widely used in application areas as diverse as nuclear energy exploration and atmospheric research. In the field of engineering, such products are commonly used to visualize flow patterns and stress fields, and generally to study large multi-dimensional data sets. SV applications are used in many industries including aerospace, medicine, power production, shipbuilding, geophysics, automotive, electronics, oil, agriculture, food production, etc. SV applications are now ubiquitous in engineering and science, be it in:

- Fluid Mechanics,
- Structural Analysis,
- Electromagnetics,
- Thermodynamics,
- Nuclear Physics, etc.

For the sake of completeness, let us mention that SV has been (and is) instrumental in advancing the state of the art in industrial applications involving fluid flow modeling, such as:

- Aerodynamics of trains, cars and airplanes.
- Hydrodynamics of ships and floating structures.
- Flow in turbo-machinery and torque converters.
- Cryogenic rockets, combustion chambers simulations.
- Flow in manifold, pipes and machinery.
- Medical researches, circulation of blood in veins.

It is evident that advances in engineering software are driven by demands from many application areas, which in turn places requirements on the associated visualization software. Today, visualization software solutions with interactive 3D graphics capabilities can be categorized into four groups:

1. Visualization Applications
2. Modular Visualization Environments
3. Visualization Toolkits
4. Integrated Modeling Environments

A. Visualization Applications

Stand-alone visualization applications are software solutions, which offer direct functionality to the user, who is responsible for defining the data set required to be loaded for performing the visualization task. Some of the well known visualization software tools for the CFD and Finite Elements Analysis (FEA) are given in the following list:

- EnSight from CEI [6],
- FieldView from Intelligent Light's [7, 8],
- TecPlot from Amtec Engineering Inc. [9],
- CFView from NUMECA [10],
- PLOT 3D, NASA [11],
- VISUAL2-3 from MIT [12],
- ParaView from VTK [13],
- VisIt from Lawrence Livermore National Lab [14]

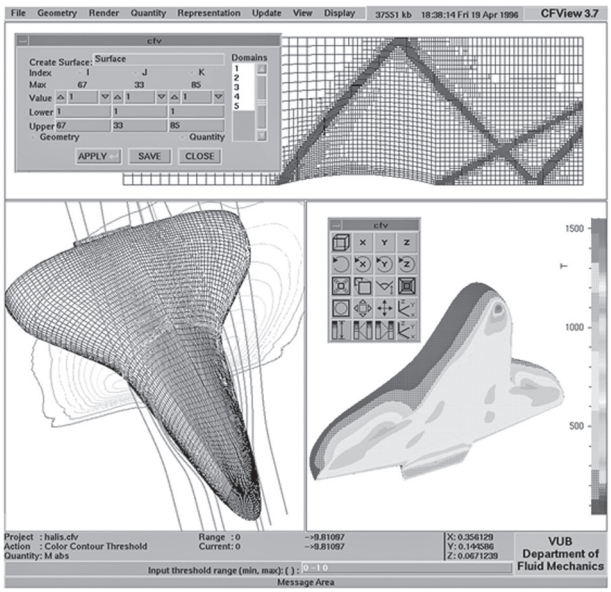

Fig. 2.: CFView the scientific visualization system

Such programs are appropriate for users who need off-the-shelf visualization functionality. Such software implements the 'event-driven' programming paradigm which is suitable where all functions are launched by the user interacting with the Graphical User Interface (GUI). This is the case for CFView [2]; see Figure 2, a scientific visualization application developed by the author over the 1988-98 period. CFView started as an academic application in 1988 and it was continuously upgraded in the following years. In the mid 90's, CFView was taken over by the VUB spin-off company NUMECA and integrated in 'FINE', that nicely illustrates the variety of visualization tasks that need to be performed to solve an engineering problem.

B. Modular Visualization Environments

Modular Visualization Environments (MVE) are programs often known as 'visualization programming environments'. Examples are [15]:

- Advanced Visual Systems AVS [16],
- Iris Data Explorer from Silicon Graphics [15, 17],
- OpenDX, the IBM's Data Explorer [18],
- PV Wave from Visual Numeric [19].

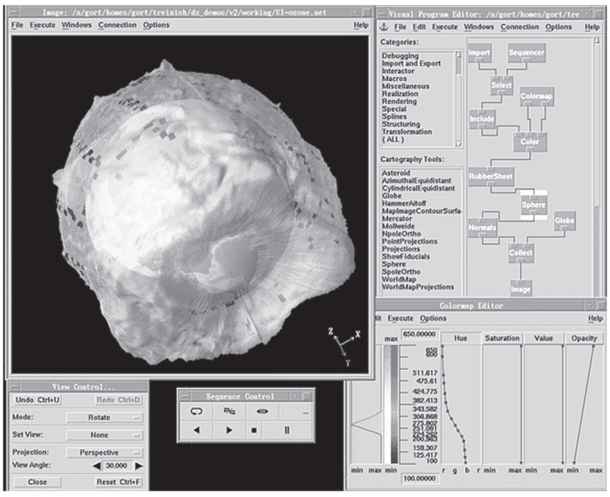

Fig. 3. The OpenDX Application Builder

Their most significant characteristic is the visual programming paradigm. Visual programming intends to give users an intuitive GUI for them to build customized visualization applications. The user graphically manipulates programming modules displayed as boxes, which encapsulate the available functionality. By interconnecting boxes, the user defines the data stream from one module to another, creating thereby the application. The MVE can be viewed as a 'visualization network' with predefined building blocks, which often needs to be quite elaborate in order to be useful to the user. The freedom given to the users to design their own visualization applications is the strength of so-called 'application builders'. This class of software implements the 'data flow paradigm', with a drawback that iterative and conditional constructs are difficult to implement. For example, PV Wave uses an interactive fourth-generation programming language (4GL) for application development, which supports conditional logic, data sub-setting and advanced numerical functionality in an attempt to simplify the use of such constructs in a visual programming environment. The interactive approach is usually combined with a script-oriented interface, and such products are not easy to use 'right out of the box' and have a longer learning curve than stand-alone applications.

There is an ongoing debate on whether the 'best' way to procure visualization software is to use stand-alone applications or to build applications using MVEs. Time has shown that both approaches are equally accepted as there is no alternative. The suggested visualization solution is a compromise between the previous and this one. For example, the GUI of CFView looks very much like the one of a stand-alone visualization application; internally though, CFView is an object-oriented system which has a flexible, modular architecture of the application builder. This is to say that a new component can be integrated in the core application structure with a minimum coding effort; also, the resulting effects from the modification propagation are kept limited.

A. Visualization Toolkits

Visualization Toolkits are general-purpose object-oriented visualization libraries, usually present as background components of SV applications. They emerged in the mid 1990's, and the two representative examples are VTK[20] and VisAD[21]:

The Visualization ToolKit (VTK) is an open-source software system for 3D computer graphics, image processing and visualization, now used by thousands of researchers and developers in the world. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python. VTK supports a wide variety of visualization algorithms (including scalar, vector, tensor, texture and volumetric methods), advanced modeling techniques (such as implicit modeling, polygon reduction, and mesh smoothing, cutting, contouring and Delaunay triangulation). In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging / 3D graphics algorithms and data.

The VISualization for Algorithm Development (VisAD) is a Java component library for interactive and collaborative visualization and analysis of numerical data. VisAD is implemented in Java and supports distributed computing at the lowest system levels using Java RMI distributed objects. VisAD's general mathematical data model can be adapted to virtually any numerical data that supports data sharing among different users, different data sources and different scientific disciplines, and that provides transparent access to data independent of storage format and

location (i.e. memory, disk or remote). A general display model supports interactive 3D, see Figure 4, data fusion, multiple data views, direct manipulation, collaboration, and virtual reality. The display model has been adapted to Java3D and Java2D, and virtual reality displays.

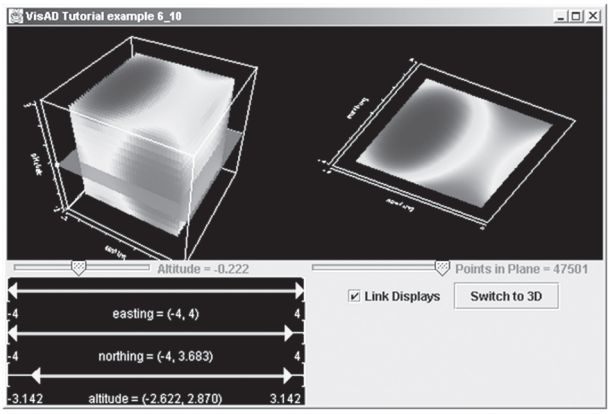

Fig. 4. VisAD application example

B. Integrated Modeling Environments

Integrated Modeling Environments (IME) is software that combines two or more engineering applications and visualization systems to solve a multi-disciplinary problem. For example, the naval architect shapes the ship hull in order to reduce the ship's hydrodynamic drag, while the stress engineer calculates the ship's steel structure. Both use visualization to analyze the data generated by the hydrodynamics and stress calculation solvers. The visualization software may be able to process the CFD flow-field solver data and the FEA stress-field solver data in a unified manner, giving the two engineers the possibility to work in a compatible way, interfacing simultaneously 3D representations of hydrodynamic and structural problems. An example of such integration is the Product Life-cycle Modeling (PLM) developed by Dassault Systèmes and the CFD solver technology developed by ANSYS, Inc., where the FLUENT CFD flow modeling approach is integrated in CATIA CAD tools throughout the whole product lifecycle [22].

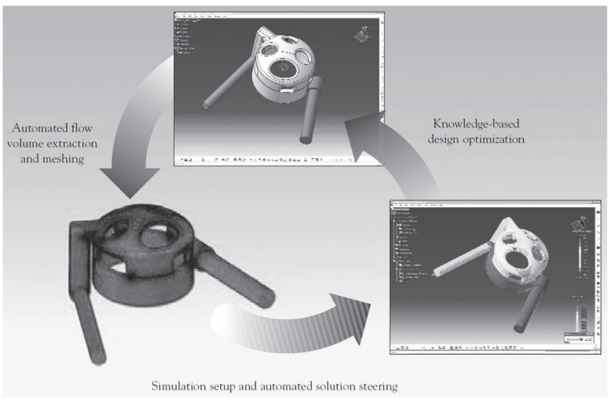

Fig. 5. The integrated modeling environment from Dassault Systèmes and ANSYS, Inc.

2. OBJECT ORIENTED METHODOLOGY

Computer hardware has improved drastically in quality and performance in the last 30 years, much faster than software quality and complexity. The trend is drawn qualitatively in Figure 6. The main reason for this situation is to be found in the reusability of hardware components (chips), which are the cheap and reliable building blocks of hardware systems, small and large. To date, software components with similar properties simply do not exist, and reusable software 'chips' are not commercially available. The effort to design and produce such software would be too large, and standardization is not pursued by software makers who keep customers captive with proprietary software and computer platforms. As a result, software production cannot keep pace with hardware technology, a situation often recognized as symptomatic of a 'software crisis'. The key idea is to try and produce visualization software that could intrinsically evolve as fast and as cheaply as hardware.

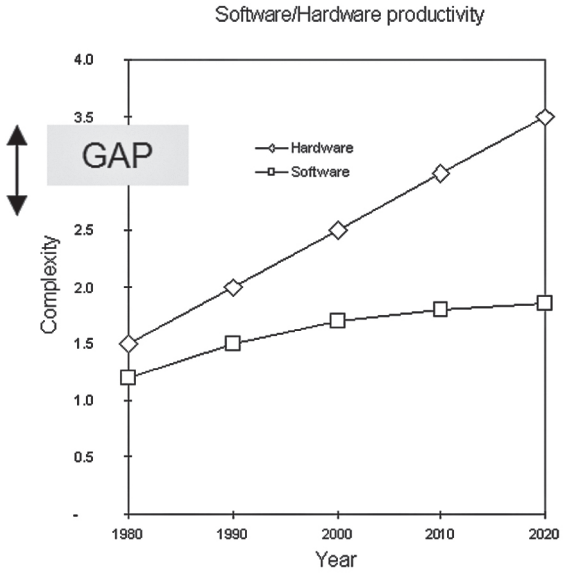

Fig. 6. Comparison of Hardware/Software productivity

In this respect *Object Oriented Methodology* (*OOM*) for constructing software components is a well adopted approach to be considered, as it is a fairly universal approach that can be applied to solve many types of complex problems. The goal of *OOM* is to reduce the system complexity by decomposing it in manageable components called *objects*. Experience has shown that solving problems in a piece-wise manner leads to better quality and easily scalable solutions. The system is 'cut' into component pieces represented by 'objects' that interact through well-defined interfaces, by exchanging information through *messages*. An interesting feature of *OOM* is that objects can be created and developed independently, even with no a priori knowledge of the application in which the objects will be used. The existence of an object is independent of any specific applica-

tion. An interesting consequence of having many reusable software objects would be that it would then make sense to get hardware designed to fit the available software (and not the reverse as is the case today). The principles of software reusability and portability are fundamental to foster software productivity. Reusability is an intrinsic feature of all *OO* software and their efficient exploitation promotes the computer network to become a commercial market place, as the Internet, in which such general-purpose and specialized software components need to be available, validated and marketed [23].

The OO approach has led to the emergence of *Object Oriented Programming* (*OOP*) with specialized *OO* programming languages, such as *Smalltalk [24], CLOS, Eiffel [25-27]*, *Objective C [28], C++ [29], Java, C#* and other derivatives, which apply encapsulation and inheritance mechanisms to enhance software **modularity** and improve component **reusability**. It is important to stress that the greatest benefit of *OOM* is obtained when *OOM* covers the full software life-cycle, from the requirements specification phase to the software delivery phase. When an application is created applying *OOM*, reusability in different development phases can be expected. First, the *OOP* brings in object-oriented libraries, which provide components validated in previously developed applications. Second, software design of previously modeled software could be reused through the established design patterns. Previously developed components may be reused for a new application which does not need to be designed from scratch, which is an obvious advantage. Improvements that could be brought to the existing, re-used objects would also improve the 'older' applications that use the same objects.

It is interesting to note that the *OOP* paradigm has shifted the emphasis of software design **from algorithms to data** (object, class) **definitions** [30-32]. The object-oriented approach can be summarized in three steps. The system is first decomposed into a number of objects characterizing the problem space. The properties of each object are then defined by a set of methods. Possibly, the commonality between objects is established through inheritance. Actions on these objects and access to encapsulated data can be done randomly rather than in a sequential order. Moreover, reusable and extensible class libraries can be created for general use. These are the features which make *OOP* very attractive for the development of software, in particular for interactive software.

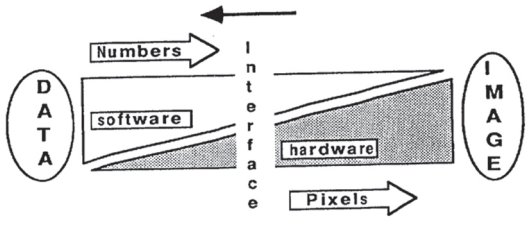

Fig. 7. Graphics Engine as a combined software/hardware solution

It should be mentioned that *OOM* does not directly reduce the cost of software development; however, it markedly improves the quality of the code by assuring consistent object interfaces across different applications. Estimated software construction times are often incorrect. **Time** and **resource** allocation tend to be largely underestimated in software projects, not uncommonly by factors of 2 to 5, especially where innovative features are to be developed. Unfortunately, for software construction planning we do not have an underlying engineering, scientific or mathematical model to calculate the software development time required, when starting a new software development process. The theoretical basis of how to best construct software does not exist. The ability to plan project costs, schedule milestones, and diagnose risk is ultimately based on experience, and could be only valid for a very similar application done in the past and applying the same development environment.

It is also important to ensure the production of **portable** code, i.e. a code that can run without need of adaptation on computing platforms other than its 'native' platform. Porting, adapting software to a computer system other than the one for which it was originally designed, can be a tedious and costly process. Portability can be improved by adopting standards supported by various hardware/system platforms. For example, one may adopt the OpenGL standard which is supported by graphic boards. This ensures that only a small kernel of code must be modified before recompilation for another hardware platform. A graphics engine typically processes floating-point input data to generate graphics. An example of graphics data models are lines and polygons. Hence, one assumes that line drawing and polygon filling are functions provided by the graphics engine, and one needs not be concerned with developing low-level graphics routines. One can therefore focus on generating the data sets needed to 'feed' the graphics engine.

To develop the visualization software, our approach must be 'multi-disciplinary' in the sense that it puts together an application engineer and a computer specialist in order to develop different application layers, as shown in Figure 8. The software development environment needs to enable evolution of the software under development and has to provide a framework for porting applications across different hardware/operating systems/windowing systems. Also, it has to simplify the process of creating interactive graphical applications, enabling the application engineer to have the application software layer under control and hide the lower software layers of the system, as depicted in Figure 8. Thus, the object-oriented approach is appropriate to introduce necessary abstraction levels and for organizing the inherent complexity present in the development of the scientific visualization software.

The fundamental concept in *OOM* is the "*object*"; it is the elementary 'building block' for mapping scientific and engineering concepts to their software equivalents. The object is an abstract construct, which ap-

proximates (in a simplified manner) the understanding of the real concept under consideration, which is often quite complex. Consider, for example, how physics of fluid flows is described in terms of numerical equations and how these equations are modeled by software objects. These objects are useful because they are identifiable elements with a well-defined purpose: each object performs a given function by encompassing a certain mathematical or physical 'intelligence' For example, an object modeling a second-order differential equation, or an object modeling the viscosity of a liquid at a given temperature, etc. such that it can be 'reused' by the software engineer with no need to understand the internal working details of the object. The obvious reused object in real life is a car. We need it to go from one place to another, but we do not need to know how it is built. We use it, and this is the way software engineers are supposed to reuse objects.

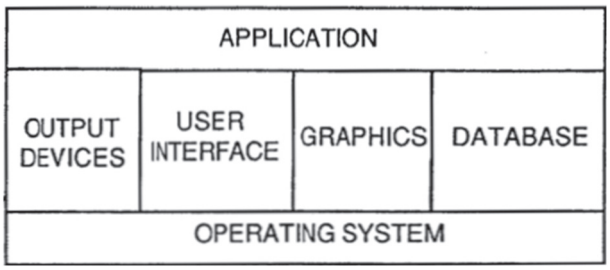

Fig. 8. Software components distribution

The fundamental concept in *OOM* is the "*object*"; it is the elementary 'building block' for mapping scientific and engineering concepts to their software equivalents. The object is an abstract construct, which approximates (in a simplified manner) the understanding of the real concept under consideration, which is often quite complex. Consider, for example, how physics of fluid flows is described in terms of numerical equations and how these equations are modeled by software objects. These objects are useful because they are identifiable elements with a well-defined purpose: each object performs a given function by encompassing a certain mathematical or physical 'intelligence' For example, an object modeling a second-order differential equation, or an object modeling the viscosity of a liquid at a given temperature, etc. such that it can be 'reused' by the software engineer with no need to understand the internal working details of the object. The obvious reused object in real life is a car. We need it to go from one place to another, but we do not need to know how it is built. We use it, and this is the way software engineers are supposed to reuse objects.

The *software model* is a fundamental element in *OOM* software development. The model describes the knowledge mapped in the software in a formal, unambiguously defined manner. Such a precise specification is both a documentation and a communication tool between developers and users; recall that the term 'developer' includes application analysts, software designers and coding programmers (see Figure 9).

In the software development process, the analyst creates an abstract model that will be partially or fully implemented. The designer uses that model as a basis to add specific classes and attributes to be mapped onto one or more OOP languages. The designer specifies the detailed data structure and functional operations/processes, which are required by the application specification

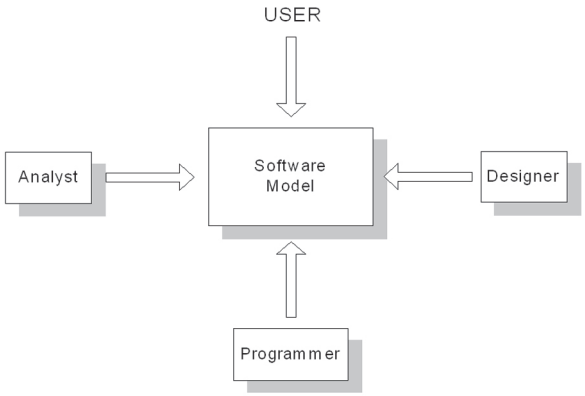

Fig. 9. Software model as communication media in the software development process

Finally, the programmer receives the analyst's and the designer's models for implementation into source code. The source code is compiled to produce the executable software. Software modeling is then an iterative and incremental process which maps abstract concepts into formal constructs that eventually become reusable software entities.

In *OOM*, the object model comprises a data model and a functional model. Specification of an object includes a description of its behavior and the data necessary and sufficient to support its expected functionality. The data model describes pertinent data structures, relations between the objects and the constraints imposed on the objects. The functional model describes objects' behavior in terms of operations. From the data model point of view, the primary concern is to represent the structures of data items important to the scientific visualization process and the associated relationships.

3. THREE EUROPEAN PROJECTS

The three European projects ALICE, LASCOT and SERKET are given as examples in which the evolving computer software technologies have been researched and demonstrated to address the evolution of the visualization software in engineering and for information visualization in general.

A. Alice – QFView – towards the transparent visualization of numerical and experimental data sets

The development of QFView in the ESPRIT-IV "ALICE" project (EP-28168) extended author's research towards using the World Wide Web for designing and building up distributed, collaborative scientific environments [33, 34]. QFView was developed in a web-oriented client-server architecture (e.g. Java, JDBC) which allowed openness and modularity, as well as improved flexibility and integration of visualization components (current and future). A core element was creation of a central database where very large data sets were imported, classified and stored for re-use. The distributed nature of QFView allows the user to extract, visualize and compare data from the central database using World Wide Web access. QFView integrates experimental and computational data processing (e.g. flow field mappings with flow field visualization), see Figure 10.

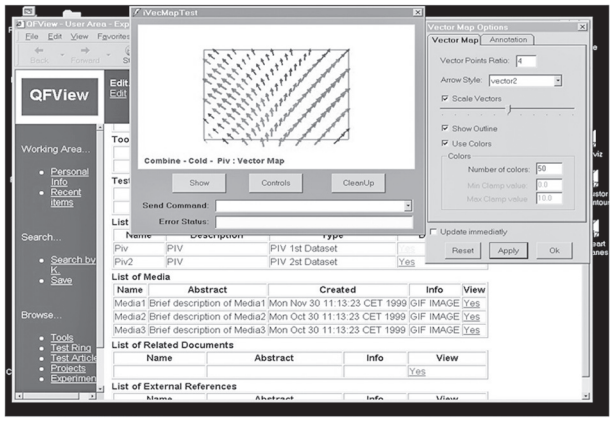

Fig. 10. QFView Web Interface

B. LASCOT –Visualization as a decision-making aid

The LASCOT project [35] is part of the EUREKA/ITEA initiative. The Information Technology European Advancement (ITEA) program for research and development in middleware is jointly promoted by the Public Authorities in all EU Members States and some large European industrial companies.

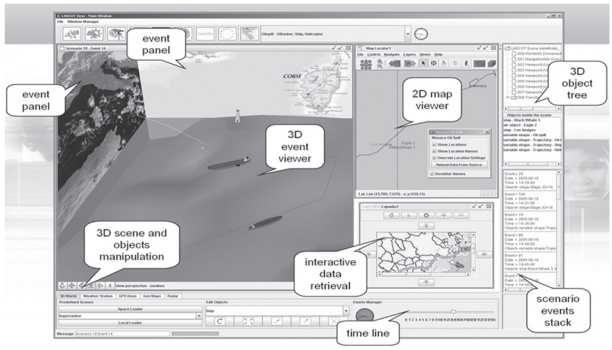

Figure : LASCOT 3D graphical user interface

Fig. 11. LASCOT 3D graphical user interface

The goal of LASCOT was to design, develop and demonstrate the potential benefits of distributed collaborative decision-support technology to the "future cyber-enterprise in the global economy" demonstrating the following issues:

- Support access to traditional information systems and to Web data;
- Enable situation assessment and provide decision-support facilities as well as simulation and validation facilities to support business decisions;
- Include current, enhanced-as-required security tools;
- Make use of visualization technology for critical tasks such as decision-making and knowledge management;
- Produce an online learning application to facilitate embedding of the platform by the users.

C. SERKET – Security situation awareness

The SERKET project [36] explored a solution to the issue of security in public areas and events by developing an innovative system whereby dispersed data from a variety of different devices are automatically correlated, analyzed and presented to security personnel as 'the right information at the right time'. The aim was to design and develop an open-software platform that can be deployed at low cost.3D software development in SERKET is centered on the visualization and presentation engine, with special attention placed on the application of X3D (*eXtensible 3D*) and XML (*eXtensible Markup Language*) standards. The graphical middleware correlates, combines, annotates and visualizes sensor data and related metadata (the application context is security). Using sensor data analyzed by other processing and data fusion components, the graphical middleware builds 3D scenes which represent the objects detected by the sensors and the operational status of sensors at their locations. Objects in the 3D scenes are annotated with metadata and/or with links to metadata describing the security context in relation to the displayed 3D objects. The 3D display of the situation removes ambiguous and provides a highly understandable overview of the situation to the security end-user, who is able to switch between different levels of viewing details and select desired viewpoints at each level (locations of video cameras define the available viewpoints). The 3D model of situation-security awareness is parameterized in space and time as shown in Figure 13.

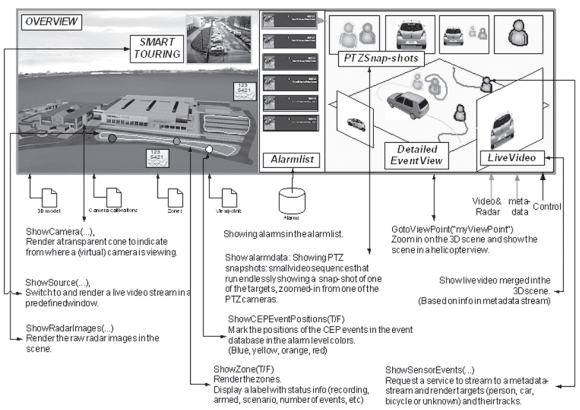

Fig. 12. The SERKET application

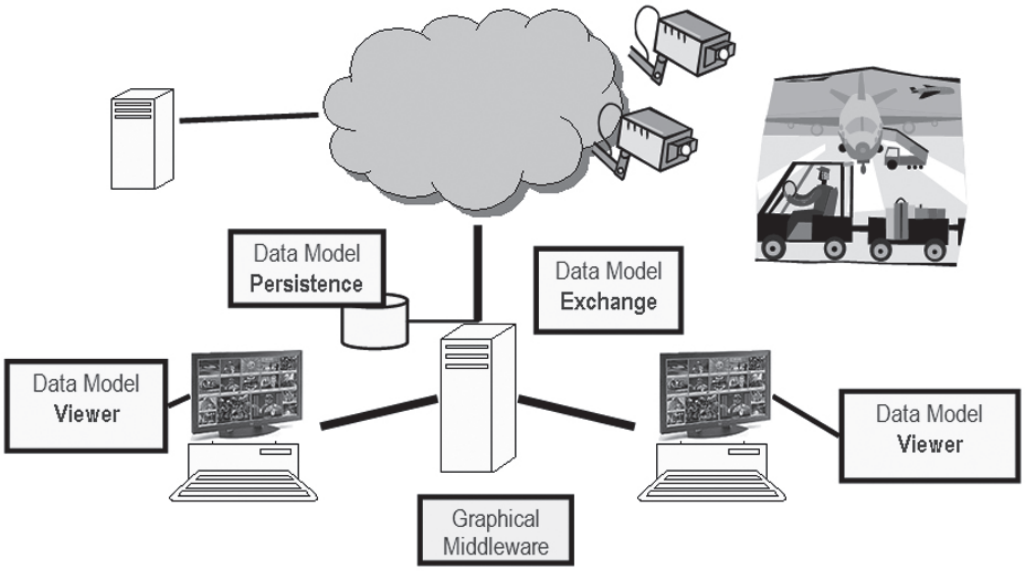

Fig. 13. The SERKET architecture overview

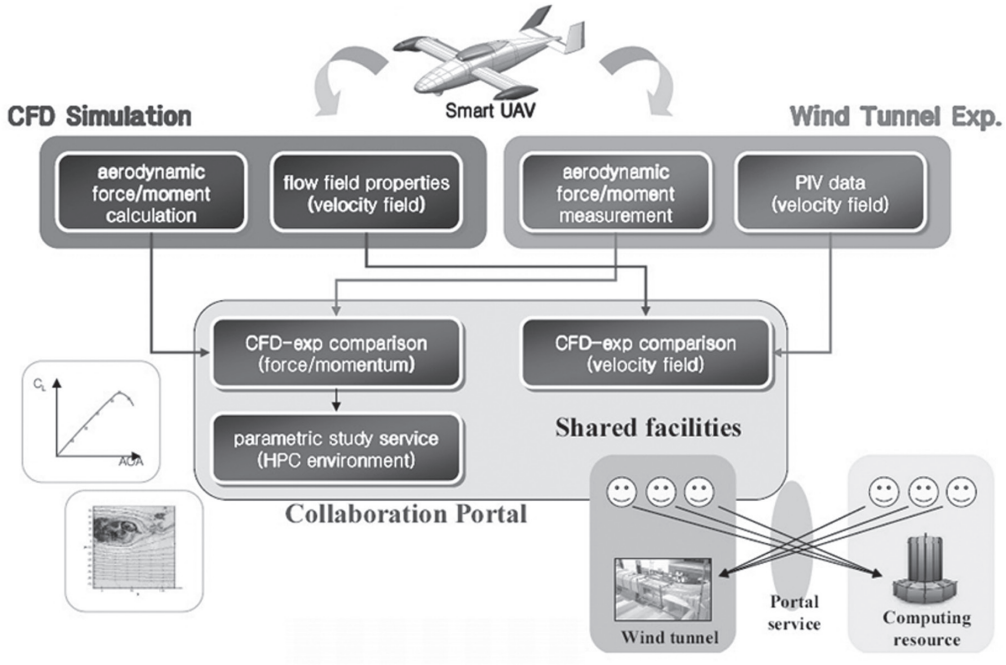

Fig. 14. Example of an Integrated Modeling Environment [37]

4. TOWARDS INTEGRATED MULTI-DISCIPLINARY ENVIRONMENTS

Today's trend in software development is towards more intelligent, multi-disciplinary systems. Such systems are expected to capture engineering intelligence and put in the hands of the engineer advanced tools for designing new products or performing investigations. The Integrated Modeling Environment (IME) [38] concept is quite recent, yet its roots can be found in 1st-generation CAD-CAM tools. An IME system attempts to offer to the engineer a homogeneous working environment with a single interface from which various simulation codes and data sets can be accessed and used. In the fluid mechanics application area, an IME system needs to integrate the latest CFD and EFD 'good working practice'; the system must be constantly updated so that at any time it runs on the most-recent software/hardware platform, see Figure 14.

An IME system consists of an Internet portal from which the investigator is able to access information/ knowledge /databases and processing functions, at any time and wherever they are located/stored. He/she has access to accurate and efficient simulation services, for example to several *CFD* solvers. Calculations can be performed extremely fast and cheaply where solvers are implemented as parallel code, and grid computing resources are available. Results obtained can be compared with separate experimental results and other computations; this can be done efficiently by accessing databases that manage large collections of archived results. The possibilities for benchmarking and exchanging knowledge and opinions between investigators are virtually infinite in an IME environment. Clearly though, a prerequisite for an *IME* environment to work is its adoption by its user community, which agrees on a specific codex that enables and guarantees openness and collaboration. Typically, an *IME* system will open Web-access to:

- Computational Services: selection of simulation software and access to processing and storage resources,
- Experimental Services: access to experimental databases with a possibility to request new measurements,
- Collaborative Services: chat and video-conferencing, with usage of shared viewers (3D interactive collaboration).

Visualization is required to support many tasks in *IME* software. This poses the problem of building/selecting data models that can be used by visualization components to present the information correctly to the users, whilst offering to them tools for real-time interaction in a natural, intuitive manner. The *IME* can include wall-displays connected to high-performance, networked computing resources. Such systems and architectures are no longer a mere vision: they are becoming reality, which opens new challenges for scientific visualization software researchers and developers. In Figure 15 and Figure 16 large multi-tiled display walls driven by a system for parallel rendering running on clusters of workstations (e.g. Chromium [39]) can adequately satisfy the requirements of high resolution large-scale visualization systems.

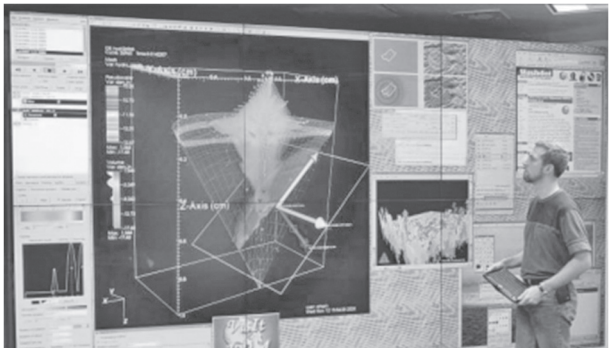

Fig. 15. Scientific visualization with Chromium

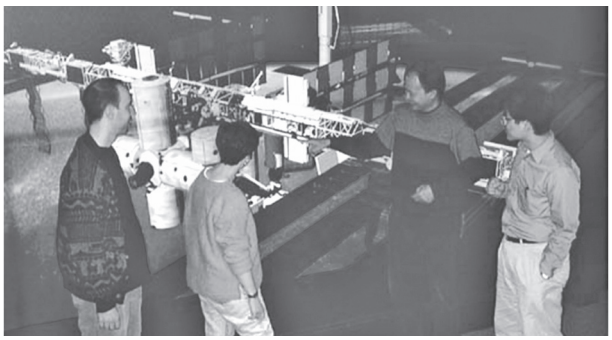

Fig. 16. NASA Space Station on display wall

5. CONCLUSION

Innovation in visualization systems poses simultaneous challenges of: building better, faster and cheaper computer-aided solutions to ever more complex scientific, engineering and other multi-disciplinary problems; developing sophisticated methodologies and algorithms; harnessing the power of upcoming technologies; re-using and leveraging the power of legacy systems and solutions; and working in increasingly shorter design and production cycles.

The paper addresses author's research over many years, in which the continuous intention was to combine engineering and computer science domains, trying to contribute to the improvements in software development methodology for constructing scientific visualization software, whose role in the multidisciplinary engineering environment today has become an obvious prerequisite. The paper describes the problem of advancing the state-of-the-art of scientific visualization systems using object-oriented methodologies and programming techniques, which are found appropriate for designing and building interactive visualization systems that meet all the requirements placed on them by engineering disciplines: correctness, accuracy, flexibility, performance, as well as by computer science disciplines: compatibility, reusability, portability. In particular, we have shown the three European project examples whose high degree of interactivity and user-friendliness can be achieved with such software solutions. More importantly, we have provided evidence that scientific visualization has deeply changed the very nature of the investigative process itself by allowing the researcher to explore and view the physical world in an intuitive, interactive and deeply illuminating manner.

The next generation engineering visualization tools will more and more associate semantic information to 3D models. They will engage web-based software standards like X3D (*eXtensible 3D)* and Semantic Web, in order to enhance visualization and manipulation of the graphical content in a distributed engineering network. The envisaged software architecture will support ontologies: to interface knowledge-based systems, to promote a web-based software solution and to enable automation of perpetual engineering tasks.

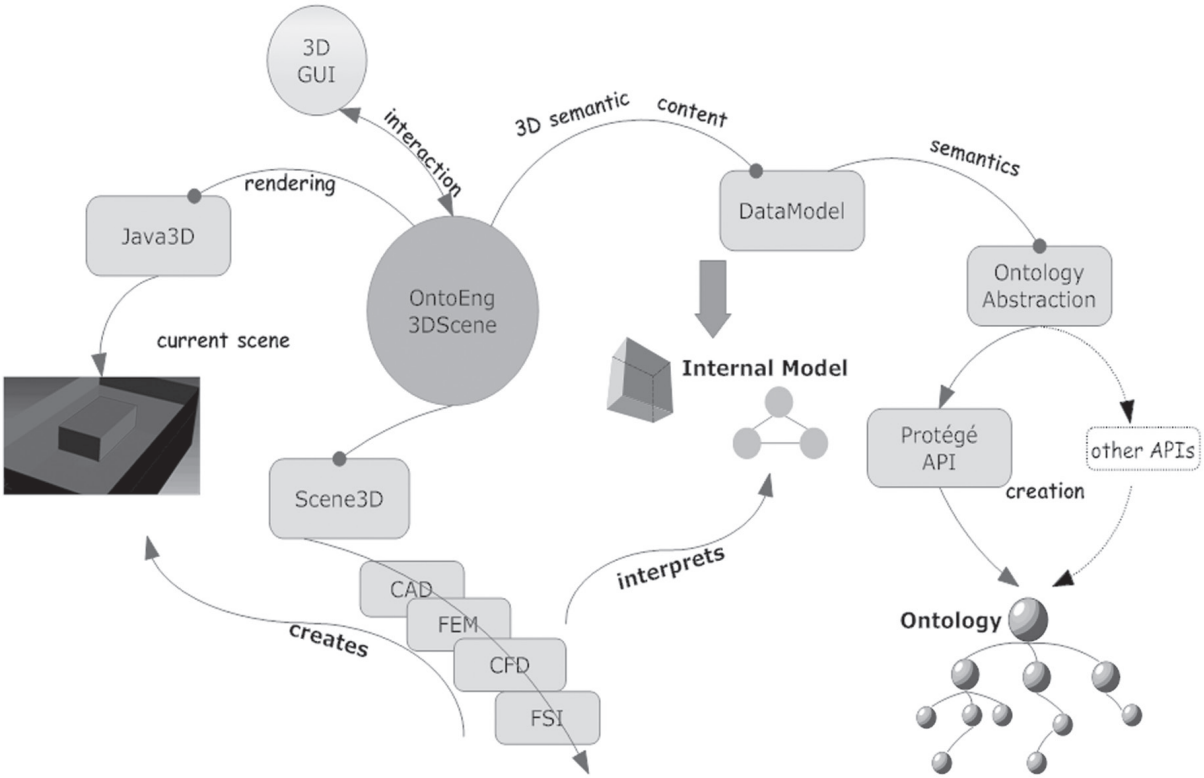

Fig. 17. Architecture overview of the ontology based visualization [40]

ACKNOWLEDGMENT

This paper would not have been created if the long-lasting research and development activity of Dean Vucinic had not been supported by the European Commission (EC) for the ALICE project (EP-28168) (1988-2001) and the Belgian national authorities (IWT) in collaboration with the EC for the ITEA projects LASCOT (2003-2005) and SERKET (2006-2007), what is gratefully acknowledged by the author.

REFERENCES

[1] M. Göbel, H. Müller, B. Urban, Visualization in scientific computing. Vienna; New York: Springer-Verlag, 1995.

[2] D. Vucinic, M. Pottiez, V. Sotiaux, C. Hirsch, "CFView - An Advanced Interactive Visualization System based on Object-Oriented Approach," in AIAA 30th Aerospace Sciences Meeting Reno, Nevada, 1992.

[3] D. Vucinic, "Development of a Scientific Visualization System", in Department of Mechanical Engineering, PhD Thesis: Vrije Universiteit Brussel, 2007.

[4] J. D. Foley, Computer graphics: principles and practice, 3rd ed.: Addison-Wesley Publ., 2006.

[5] A. B. Hanneman, R. E. Henderson, "Visualization, Interrogation, and Interpretation of Computed Flow Fields – Numerical Experiments," in AIAA Modeling and Simulation Technologies Conference, Denver, CO: AIAA-2000-4089, 2000.

[6] "Ensight, CEI Products Overview - extreme simulation software", in http://www.ensight.com/product-overview.html: Computational Engineering International (CEI) develops, markets and supports software for visualizing engineering and scientific data, 2007.

[7] E. Duque, S. Legensky, C. Stone, R. Carter, "Post-Processing Techniques for Large-Scale Unsteady CFD Datasets", in 45th AIAA Aerospace Sciences Meeting and Exhibit Reno, Nevada, 2007.

[8] S. M. Legensky, "Recent advances in unsteady flow visualization", in 13th AIAA Computational Fluid Dynamics Conference Snowmass Village, CO, 1997.

[9] D. E. Taflin, "TECTOOLS/CFD - A graphical interface toolkit for network-based CFD", in 36th Aerospace Sciences Meeting and Exhibit Reno, NV, 1998.

[10] "CFView a visualization system from Numeca," http://www.numeca.com, 2007.

[11] P. P. Walatka, P. G. Buning, L. Pierce, P. A. Elson, "PLOT3D User's Manua," NASA TM-101067 March 1990.

[12] R. Haimes, M. Giles, "Visual3 - Interactive unsteady unstructured 3D visualization", in 29th Aerospace Sciences Meeting Reno, NV, 1991.

[13] "ParaView – Parallel Visualization Application", in http://www.paraview.org, 2004.

[14] B. J. Whitlock, "Visualization with VisIt", California, Lawrence Livermore National Laboratory: http://www.llnl.gov/visit/home.html, 2005.

[15] J. Walton, "NAG's IRIS Explorer", in Visualization Handbook, C. R. J. a. C. D. Hansen, Ed.: Academic Press, 2003.

[16] C. Upson, "Scientific visualization environments for the computational sciences", in COMPCON Spring '89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers., 1989, pp. 322-327.

[17] D. Foulser, "IRIS Explorer: A Framework for Investigation", Computer Graphics, vol. 29(2), pp. 13-16, 1995.

[18] "OpenDX is the open source software version of IBM's Visualization Data Explorer", http://www.opendx.org/, 2007.

[19] "PV-WAVE, GUI Application Developer's Guide", USA: Visual Numerics Inc., 1996.

[20] W. Schroeder, K. W. Martin, B. Lorensen, The visualization toolkit, 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[21] W. Hibbard, "VisAD: Connecting people to computations and people to people", in Computer Graphics 32, 1998, pp. 10-12.

[22] "Fluent for Catia V5, Rapid Flow Modeling for PLM", http://www.fluentforcatia.com/ffc_brochure.pdf, 2006.

[23] B. J. Cox, A. J. Novobilski, Object-oriented programming: an evolutionary approach, 2nd ed. Reading, Mass.: Addison-Wesley Pub. Co., 1991.

[24] A. Goldberg, D. Robson, Smalltalk-80: the language. Reading, Mass.: Addison-Wesley, 1989.

[25] B. Meyer, Reusable software: the Base object-oriented component libraries. Hemel Hempstead: Prentice Hall, 1994.

[26] B. Meyer, Eiffel: the language. New York: Prentice Hall, 1992.

[27] B. Meyer, Object-oriented software construction. London: Prentice-Hall International, 1988.

[28] L. J. Pinson, R. S. Wiener, Objective-C : object-oriented programming *techniques*. Reading, Mass.: Addison-Wesley, 1991.

[29] B. Stroustrup, *The C++ Programming Language*, Special Edition ed.: Addison Wesley, 1997.

[30] G. D. Reis, B. Stroustrup, "Specifying C++ concepts", in *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages,* Charleston, South Carolina, USA: ACM Press, 2006.

[31] B. Stroustrup, "Why C++ is not just an object-oriented programming language", in *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum),* Austin, Texas, United States: ACM Press, 1995.

[32] R. Wiener, "Watch your language!," *Software, IEEE,* vol. 15, pp. 55-56, 1998.

[33] D. Vucinic, J. Favaro, B. Sünder, I. Jenkinson, G. Tanzini, B. K. Hazarika, M. R. d'Alcalà, D. Vicinanza, R. Greco, A. Pasanisi, "Fast and convenient access to fluid dynamics data via the World Wide Web", in *ECCOMAS European Congress on Computational Methods in Applied Sciences and Engineering 2000,* Barcelona, Spain, 2000.

[34] D. Vucinic, M. R. Barone, B. Sünder, B. K. Hazarika, G. Tanzini, "QFView - an Internet Based Archiving and Visualization System", in *39th Aerospace Sciences Meeting & Exhibit,* Reno, Nevada, 2001.

[35] "LASCOT project - home page", in *http://www.bull.com/lascot/index.html*, Bull, Ed., 2005.

[36] "SERKET project - home page," http://www.research.thalesgroup.com/software/cognitive_solutions/Serket/index.html, Ed.: Thales Research & Technology, 2006.

[37] M.-J. Jeong, K. W. Cho, K.-Y. Kim, "e-AIRS: Aerospace Integrated Research Systems", in *The 2007 International Symposium on Collaborative Technologies and Systems (CTS'07),* Orlando, Florida, USA, 2007.

[38] C. M. Stone, C. Holtery, "The JWST integrated modeling environment", 2004, pp. 4041-4047, Vol.6.

[39] G. Humphreys, M. Houston, Y.-R. Ng, R. Frank, S. Ahern, P. Kirchner, J. T. Klosowski, "Chromium: A Stream Processing Framework for Interactive Rendering on Clusters", in *SIGGRAPH*, 2002.

[40] D. Vucinic, M. Pesut, A. Aksenov, Z. Mravak, C. Lacor, "Towards Interoperable X3D Models and Web-based Environments for Engineering Optimization Problems", in *International Conference on Engineering Optimization,* Rio de Janeiro, Brazil, 2008.