VISUAL-GAMA
Graphical data visualization and monitoring for maritime container terminals

Pedro Jorquera, Toni Barella, Diana Llobregat and Roberto Vivó

{pjorquera, tbarella, rvivo}@dsic.upv.es, diallovi@eui.upv.es

Computer Graphics Group, Department of Computer Science (DSIC)
Polytechnic University of Valencia, Camino de Vera s/n, 46022 Valencia, SPAIN

Abstract

We present a contribution to the development of visual applications for planning, monitoring and integral management of container terminals. We introduce a system that improves on current commercial applications by providing real time access and visualization of the container terminal. Our system is accessible using modern wireless devices and an event based transaction engine. We show how the integration of different technologies creates a final product that outperforms other systems for container terminal management.

Key words: *Container terminal, graphic user interfaces, wireless devices, OpenGL, Web, XML.*

1. Introduction

Container terminal are a great environment for developping visual tools for management, planning and monitoring where all the information involved in the processes might be graphically and clearly presented. Normally, those container terminals use to have an information system that displays data by using poor alphanumeric interfaces without the ability to adapt to the changes like advanced graphical user interfaces.

In the last three years we have been involved in the development and installation of integral management system of a terminal container. It automates most of the production processes of such enterprises (GAMA[7]). Currently the system is being exploited by Marítima Valenciana, S.A., one of the greatest container terminals of Europe, with about two million movements of containers/year on an extension of 3 km^2. Some applications concern with computer graphics and have relationship witn other fields such as XML, web, GUI, multithreading, etc. We show here the problems we have found and the solutions we have choosen, particularly in those applications with a relevant graphical content.

2. Visualization problems

The real-time visualization of large out-of-core models presents some problems that can be solved with innovative techniques like the ones we present here. Those techniques solve the two fundamental problems: the real-time visualization of very big datasets and the modification of the data in applications designed with an event oriented model.

Other issue is vessel loading and discharging (stowage planning): a list composed of univocal associations among a yard location (container) and a ship position (slot). Thus, we must show an elevated number of elements at the same time, and for each one, it is necessary to show a great amount of information simultaneously in a reduced area while the user needs the maximum information available to reduce the number of permissible errors.

Other problem we had to solve was the remotely information acquisition to support mobile platforms such Pocket PCs and SmartPhones and eventually to emit the information out of the enterprise. Ours objectives were to develop real-time system, with advanced GUI's and adaptability to the terminal changes, maximizing the graphical representation with forms and colors with filtering, and giving support for wireless devices.

3. Navigator

Navigator is a 2D 1/2 visualization application that represents everytime the state of the container in the terminal, showing graphically containers, vessels, trucks and other mobile elements. The application is highly customizable and scalable and fits easily on existing information systems. It adapts itself to the topological characteristics of any container terminal with a powerful and simple graphical user interface.

Navigator visualizes in real time the state of the totality of mobile and static elements. These are respectively located by means of tracking devices and reference positions. Other visualization systems lack of this advanced feature [56]. To do that it uses a completely event oriented model disconnecting it from the system data base. On one hand it loads the necessary data structures to make up the graphical representation, loading the position of the containers and location of the machines. On the other hand it builds a map of precompiled OpenGL[2] command lists of geometry representing, among others, the static container groups using different coloring policies. It also precompiles vessels and cranes, buildings, road marks, etc. Everything is also located in the UTM coordinates calculated from the GPS projection of the container terminal.

The precompiled GL lists are stored in a hierarchical structure that allows us to find quickly which of them are needed to be represented at the same time. Navigator compiles only the GL list that has implied some modifications by external events, balancing the computing cost of geometrical compilation.

To achieve a true real-time visualization system was necessary to make Navigator multithreaded so that the process of external events as well as the geometrical compilation did not delay the representation of the visualized information. The geometrical compilation is carried out in a separated execution thread whose work is simply to regenerate GL lists based on new external events of changes on position or state of elements. At the same time, the main thread of the application reads the precompiled GL lists of geometry making decissions about which GL lists has to visualize. It represents them in the viewport and processes the user interface events.

We had some problems to solve since the render context of OpenGL were shared by both execution threads. The first thread (the one who is compiling the GL lists) uses the lists of the render context in WRITE ONLY mode, whereas the second (the one who represents in the viewport) uses these lists in READ ONLY mode. The solution was to associate to each GL list a flag. It informs about if the list was involved in an reading operation, a writing operation or it did not have an active operation.

When the viewport needs to be painted the reading thread determines which compiled GL lists must represent, being based the decision on the parameters of the application like visible layers and level of detail. Next it determines which GL lists fall within the viewing frustum by using frustum culling techniques based in the bounding boxes of each GL list. Once it has determined the potentially visible set of objects (in this case the set of potentially visible OpenGL lists) it carries out its representation reading the flag associated with each list and delaying the execution of the lists involved in an operation of writing and prioritizing those that are not being compiled. Figure 1 summarizes this process.

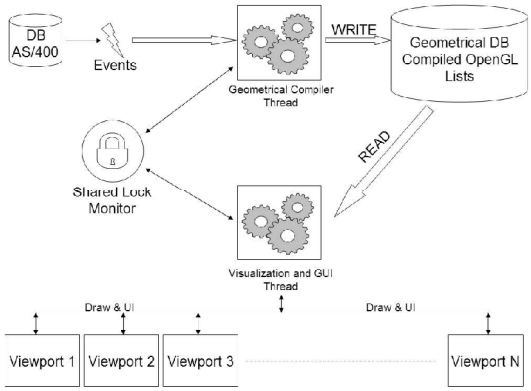

Figure 1: *Threads synchronization for shared GL contexts.*

4. SMES

SMES is a stowage planning system based on a 2D front-end, a sequence engine and a database. The stowage planning process is made manually, which implies a high degree of interaction with the user. In addition, SMES is feeded with information from a database resident in a server. This implies a multilayer architecture, with the application at the top, the planning engine at the middle and the database server at the bottom. Thus, we have focused on offering a simple and intuitive interface to the user, which allows us to change others modules with no effect on the visual layer.

The visual interface uses multiple views to show different information. The main views are: a *Yard View* to visualize the containers grouped by hierarchical structures as the real organization of the terminal; a *Vessel View* to show the content of a cross-sectional section of a ship (<deck-covers-hold>); a *Text Info View* to give detailed textual information, making the user easier to trace the sequence

We have solved the problem of visualizing a great amount of stowage information in a reduced area by using a graphical language based on iconography, coloration and partial changes in the morphology. Thus it is possible to visualize up to ten items simultaneously, without stretching excessively the painted area. On the other hand, this information is practically static, since changes use to be on the states of

the elements, and not on the amount of elements to visualize. It led us to choose a 2D representation method with precalculated geometry. In order to avoid the blinking problem, we have used the technique of double buffer over the native drawing API[8].

5. TReS

The need to operate with containers in a more suitable graphical form give us the idea of creating a reutilizable view that could be componentized and integrated in more applications. Taking that into account we developed an application (TReS) to make uploads and downloads of containers from a graphical view. From TReS the user can upload a container from a truck or download from a crane, all visually made and synchronized to the system database. Crane operation is visualized in real-time and the productivity is highly incremented.

The application precompiles geometrical information in a 2D form updating it with the external events arrivals and uses double buffer for its representation. The topological information is read from a XML representation and the application creates an efficient data structure for updating and finding information. The compiled geometry is customizable by other XML files that configures the data, aspect, colors and symbols and such information is used for creating the new compiled bitmaps by events arrivals. Figure 5 shows a screenshot of this application.

6. System Monitoring and Web Interface

An very important agent of our development is the system operation monitor. Its objective is the knowledge generation and its distribution to other agents. It compounds performance counters for the different machines that operates in the terminal. In addition, it offers real-time information for the predictive control of the machines that move the containers and evaluates the benefits of the system.

In addition to the software agents involved in the terminal operative, we wanted to give support to management decision of human operators who consumed this information in a more suitable graphical form. For it we developed a set of web applications with the objective to offer this information by means graphical user interfaces and intuitive forms. The monitoring information is presented as bar charts or pies as usual.

Developed architecture is based on the DNA client/server model of three layers[1] where the generation of dynamic graphs is made in the server side. Normally the web applications have been developed by means of techniques based on CGIs where server generates HTLM dynamically. However, the images (GIF, JPG) associated to these pages are not generated dynamically: usually they are a resource (files) stored locally in the server.

Our system allows us to create dynamically not only content HTML of the pages, but also the associated graphs of monitoring information as figure **??** shows. The application server chosen to deploy the web applications of monitorization was Jakarta Tomcat and the language used for the implementation was Java by using the Servlets API[4] and J2 Enterprise Edition API[3].

We did not consider other possible solutions of implementation like sending the information in XML from server to clients delegating the processing and the generation of graphs to the clients. This would have invalidated very popular potential platforms of development such SmartPhones and PDAs that practically does not have hardware processing resources.

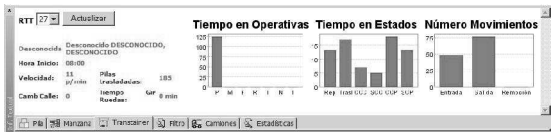

Figure 2: *Graphics server-generated and consumed by other agents.*

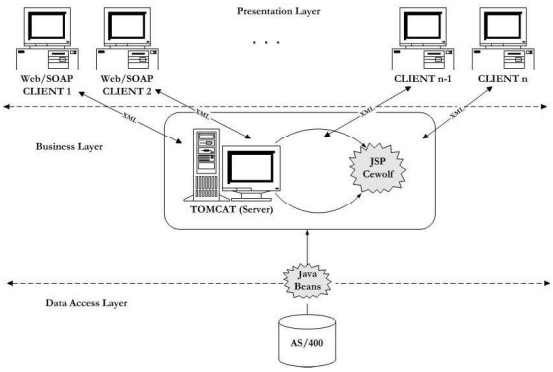

Figure 3: *Web Interface Architecture.*

7. Wireless Devices Support

Container terminal's workers are in permanent mobility: container's manipulators, trucks and cranes drivers and yard agents are moving around the terminal without possibility of using a PC but with the same requirements of access to the information system that the rest of users. One good solution would be to provide them with graphical tools similar to the developed desktop applications. The main task was to find a solution to the problem of the accessibility of the information remotely.

We developed a system based on the three layers client/server model (figure 2). This allows to acquire the information using standard Internet protocols. The client applications developed over mobile platforms (Pocket PC, Em-

bedded Linux, Java Applets) can send and receive messages using XML, by means of a contractual language of access to remote objects.

Developed architecture.

Web Service (PDBCServer): The system data base is accessible by means of native software librarys or through bridges ODBC. We created for it a web service that maintains its connection with the data base and publishes certain access methods to the information. This service runs in an web application context under a Jakarta Tomcat server[4].

Publication of remote access methods: We qualified a mechanism by means of it is possible to invoke a certain method of the web service remotely and client platform agnostic. To do that we codified in the URL the name of the method and the arguments of execution like parameters of a request HTTP/GET.

Consumption and presentation of the information: We carried out the development of graphical applications that represent the information of the container terminal with similar features to the desktop applications. In order to make the access to the remote methods easier we created a library DLL (PDBCClient) that publishes methods that the web service implements. Those methods do the task of parameters serialization of HTTP requests, making the connection over TCP/IP and deserializing the returned XML data.

Developed mobile applications use the services of this library and display the information graphically over platform Pocket PC using GDI. The great advantage of this visualization system is its transparent scalability and the availability of the information in a heterogenous form since it is based on open Internet protocols.

Figure 4: *Pocket PC Navigator.*

8. Conclusions

Big enterprise environments like container terminals need new solutions to new visualization problems. Our development is implanted and currently running in one of the biggest container terminals of Europe. Our work makes the following contributions: it makes real time visualization of these environments possible by using OpenGL, it supports rendering very large amounts of data organized graphically and it allows using wireless devices by generating the graphics on the server side. Our results have been used to increase the productivity at the container terminal.

Acknowledgements

The GAMA project is being supported by grants 1FD97-2158-C04-01 and TIC2002-04166-C03-01 from the European Community and the Spanish Ministry of Science and Technology. We thank GAMA-team for their work and collaboration. We are also grateful to Maritima Valenciana for their support of our research efforts and especially to Eduardo Orellana for his faith in our work.

References

1. Microsoft Corporation, 2000. *Microsoft Windows DNA: Distributed iNterneted Applications*. http://www.microsoft.com/dna. 3
2. M. Woo, J. Neider, T. Davis, D. Shreiner *OpenGL Programming Guide*. Addison Wesley, 1999. 2
3. Sun Microsystems, Inc. *Java 2 Enterprise Edition*. http://java.sun.com. 3
4. The Apache Project. *Tomcat API Specification*. http://jakarta.apache.org. 3, 4
5. COSMOS N.V. *Cosmos Container Software*. http://www.cosmosworldwide.com. 2
6. Total Soft Bank, Ltd. *CATOS*. http://www.tsb.co.kr. 2
7. Simarro, R., J.L Navarro, I. Huet, E. Orellana *Simulación de una terminal marítima de contenedores*. Workshop en Metodologías de Modelado y Simulación de Sistemas. Barcelona 2001. 1
8. David J. Kruglinski. *Inside Visual C++: Updated for Version 5.0 and Internet Development*. Microsoft Press. 3

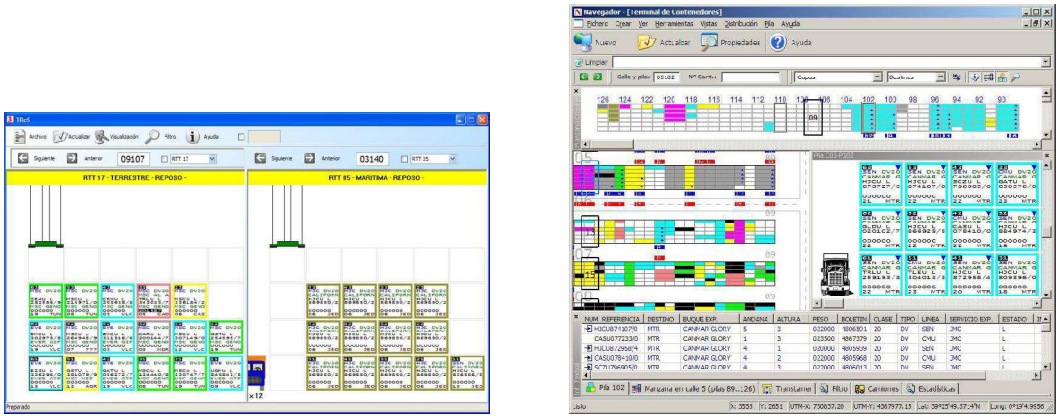

Figure 5: *TReS and Navigator applications.*

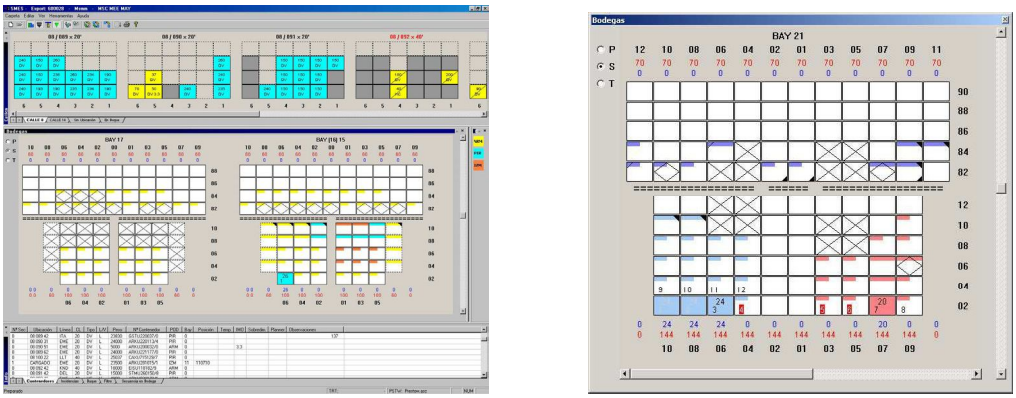

Figure 6: *General View of SMES, and detail of a Bay.*

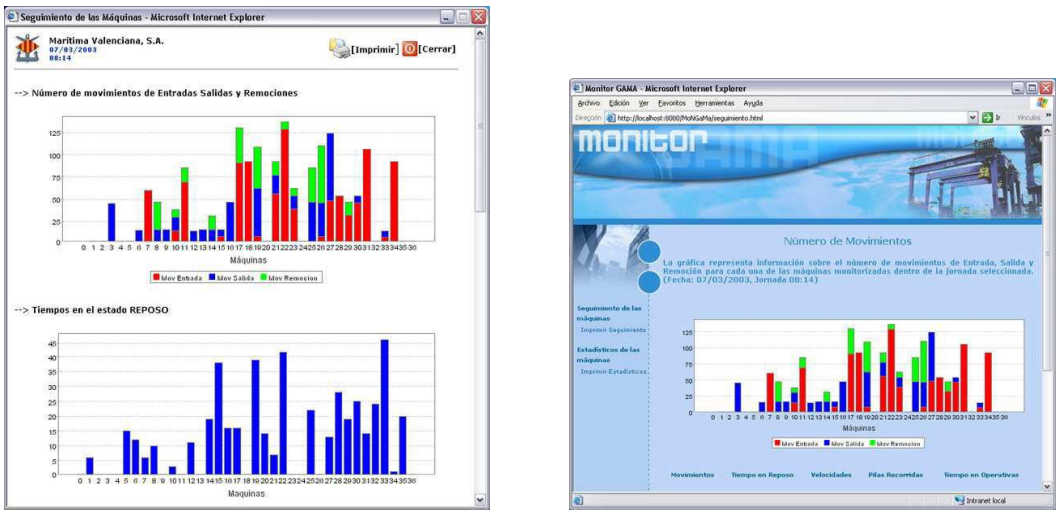

Figure 7: *System Monitoring Web Interface.*