Yacht captain training system based on VRML and Java cooperation

Jiří Chludil, Břetislav Černík
xchludil@cslab.felk.cvut.cz, xcernikb@hwlab.felk.cvut.cz

Department of Computer Science and Engineering
Czech Technical University
Prague / Czech Republic

Abstract

The paper describes ship recognition – an implementation of parts of training system for yacht captains. This program is based on *Virtual Reality Modelling Langu*age (VRML) and Java cooperation. The Java is generating and operating through *External Authoring Interface* (EAI) VRML world where visual data of training program are displayed.

Keywords: VRML, EAI, Java, training, yacht

1 Introduction

The aim of this project was a creation of experimental system based on VRML and Java cooperation, which is a perspective tool for yacht captain training. Tests for ship discrimination are a part of captain test. Currently the test is executed in written form only. It is based on using of color printed images with ships. This form is from aspect of reality deficient and archaic. Interesting experiment, how to put this test near matter, is the using of virtual reality.
Our solution uses utilises web environment where the 3D scene with ships is present in VRML window, while the movement of ships, and their behaviour an controlled by Java applet. The applet is also responsible for evaluation of user's answers.

The structure of this paper is organised as follows. The interface between EAI and Java is described in Chapter 2. The implementation of the Captain training system is described in chapter 3. Implementation details are described in Chapter 4. Chapter 5 contains discussion on the future work and problems. The last chapter contains conclusion.

2 Node manipulation layer over EAI

Since the limited possibility of EAI we have decided to implement a special layer between EAI and Java. The purpose was to create a system of classes above EAI. These classes will be represented by single types of nodes. This approach is simplified the creation of arbitrary type node (by a creation of class instance). Advantage of this layer implementation is easy access to parameters of node and this access is performed by means of existing class methods. This approach eliminate frequent mistake – the attempt at to change or read node non-existing parameters. Probably the greatest contribution is a possibility to create a special node.

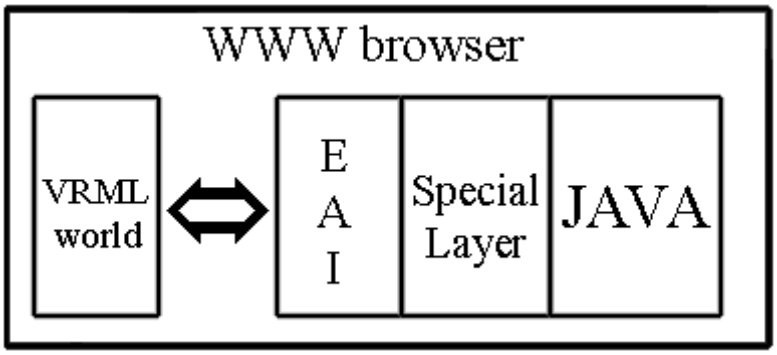

Figure 1: Implementation system block scheme.

2.1 External Authoring Interface for VRML

EAI defines an interface between VRML world and external environment. This interface allows following functions:

- sending event into node of VRML (ExposedField and EventIn only),
- reading last script funds in parameter node (ExposedField and EventOut only),
- generation new node from string or from URL address,
- obtaining information about events generated from parameters (only EventOut),
- Searching named nodes (only existing in already opened VRML and tagged by statement DEF),

2.2 Create node

One instance of class represents one particular node in VRML. The constructor contains all necessary code for communication with VRML scene via EAI. For example the creation of *PointLight* node with default parameter can be done via the following code:

Sun = new VrmlPointLight(fvrmlBrowser);

where *fvrmlBrowser* refers to a browser class.

2.3 Setting and reading parameters of node

Access to parameter of existing node is handling by class methods that are represented by this node. Methods are tagged by names of parameters according to VRML specification. Methods for reading and changing parameters have identical names. A method for parameter changing has structure:

void classinstance.parametername(value)

and for parameter reading:

value classinstance.parametername(void).

The following piece of code shows how to get a radius from *PointLight* and to increment it by 10.

```
float  param=Sun.radius();      //reading parameter
Sun.radius(param+10);           //changing parameter
```

2.4 Special nodes

The layer above EAI make possible a creation of a special construction (for example node + JavaScript). The approach to these constructions is the same like to normal node (for user

this construction will be new node type). For example a node *Shipview* is a special node calculating from two input position parameters a turn angle of a second position observed from first position.

3 Captain training system implementation

Suitable tools for 'Captain training system' presentation is VRML. This language is very restricted from point of view of 'intelligence' but on the other hand it makes possible to modify VRML world using external programme via EAI. Suitable language for this external programme is Java. Probably the greatest advantage of the simultaneously using VRML and Java is the possibility to display VRML browser and Java applet on some HTML pages.

The Java part is implemented as a applet. The applet has to be placed into the same HTML page where is the VRML browser window, it is the only way to external program can communicate with an embedded VRML world. The applet provides interaction with user and communication with VRML scene.

3.1 Captain training system modes

This application is based on classical learning process: education & test. The first mode is education. In this mode a user can set-up various mutual position of ships. A user knowledge of ship recognition can be verified by responses to test questions.

The second mode is testing. In this mode the scene is generated automatically. A user can only respond to questions appeared after the simulation had started.

3.2 Scene structure

VRML browser provides the entire ship animation. On start the Java applet provides just scene creation for VRML browser, it is to create a node hierarchic structure for simulation of ship and its movement and set up selected node parameters into this hierarchic structure in dependence on user scene setting. The brief description of hierarchy follows.

The position interpolators **PositionInterpolator** (class *VrmlPositionInterpolator*) control the ship movement. Each ship has own interpolator. The timing data from **timer** (class *VrmlTimeSenzor*) was routed into the interpolator by **routs** (class *VrmlRoute*) and the ship positions in range from starting to ending are interpolated. by these timing data in range from 0 to 1. These positions are routed to **ships** as their new positions (*class Yacht*). These positions are simultaneously routed into a **special node** which is intended for user position and orientation calculating (*class VrmlTransform*). It is calculate user position and orientation to see second ship from actual positions of both ships. The result values are forwarded by routs to **Viewpoint** (*class Viewpoint*). The **Viewpoint** is a node represent scene point of view. Finally, **Viewpoint** and **Timer** are connected. The reason of this connection is a fact, that a simulation is started by **Viewpoint** selection. When **Viewpoint** is selected send actual time to **Timer** and it starts to generate a number from 0 to 1 over given time. All above mentioned parts are nodes in VRML scene and their detail scheme is on **Figure 2**.

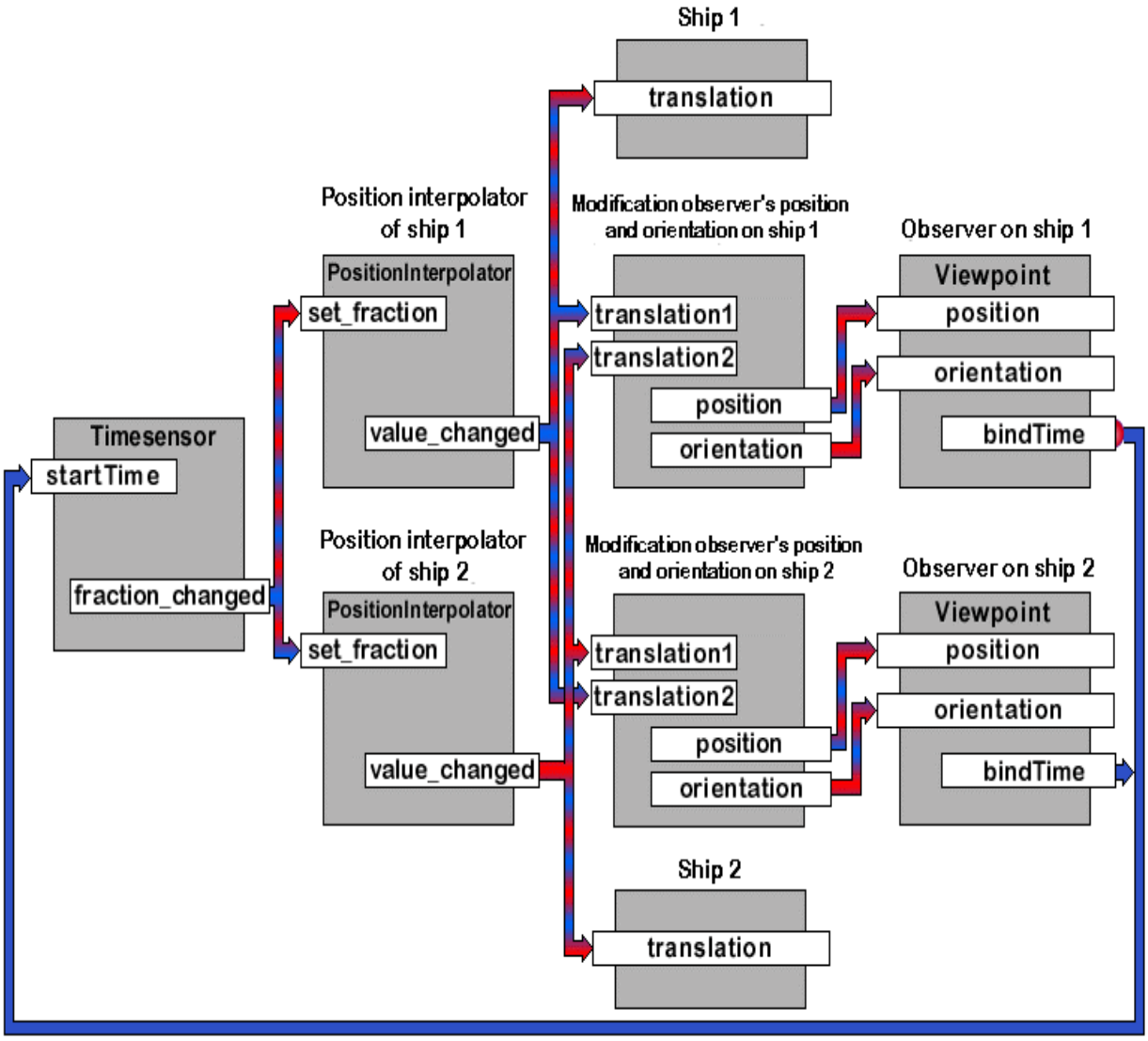

Figure 2: Animation scheme for synchronised movement of two ships and corresponding viewpoints

3.3 Captain training system GUI and control

Applet GUI consist of three menu:

1) Mode selection menu
2) Simulation control menu
3) Boat in scene control movement and setting environment menu

3.3.1 Mode selection and Simulation control

The mode change can be realised through selection in introductory menu by buttons Teaching or Testing.

After selection a panel allowing to start scene that is corresponding with question is displayed(Figure 3). Button Start initialises scene. Immediately after start a scene-related question

appears (typically to recognise a approaching ship by positional lights or sound) Button Stop finishes the scene. Button Setup (active in teaching mode only) serves setting movement of boat and environment. The last button Menu allows user to return into previous menu Change mode.

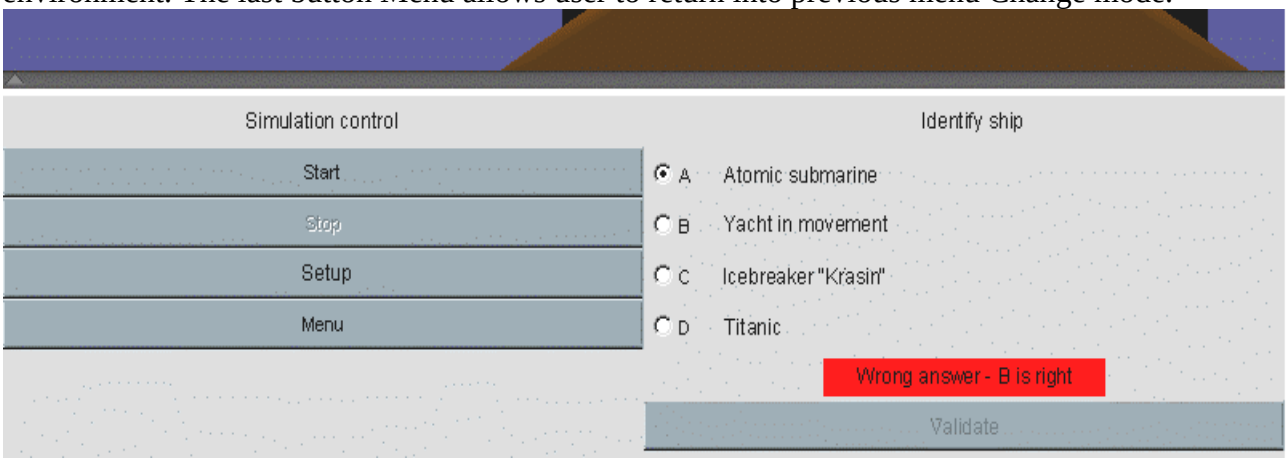

Figure 3: Simulation control layout

3.3.2 Boat in scene movement control and environment setting

In part **Setup of scene** user can set movement trajectory of both boats. The trajectory is defined by values justified by sliders: **Ship1 speed** (*in navy miles per hour*), **Ship2 *speed*** *(in navy miles per hour*), **Bilateral angle** (between both sail), **Distance of ship** (*in meters*)**, Time of scene** (*in second*) and **Time** (actual position of boats represented in percentage 0 - initial position and 100 - final position). The trajectory setting is displayed in preview. Blue and red lines determine ship's 1 and ship's 2 trajectories. The trajectory length is computed as a product of speed of given ships and total time of scene. Green circle represents distance of ship. Black full circle represents actual position of ship (it is depend on parameter **Time**). Finally user can select on which boat be found yourself. For it serves switch **Observer is on**.

In part **Weather** can user set the Sun intensity by slider **Light** and the fog intensity (0 - environment without fog) by slider **Fog**. These parameters are displayed in preview too. Box on left-up represents sky color from blue to black (in dependency on parameter light) and white wheel represent fog intensity – increasing represent better visibility.

The buttons **Cancel**, **Ok** and **Use** has the same function as Windows standard.

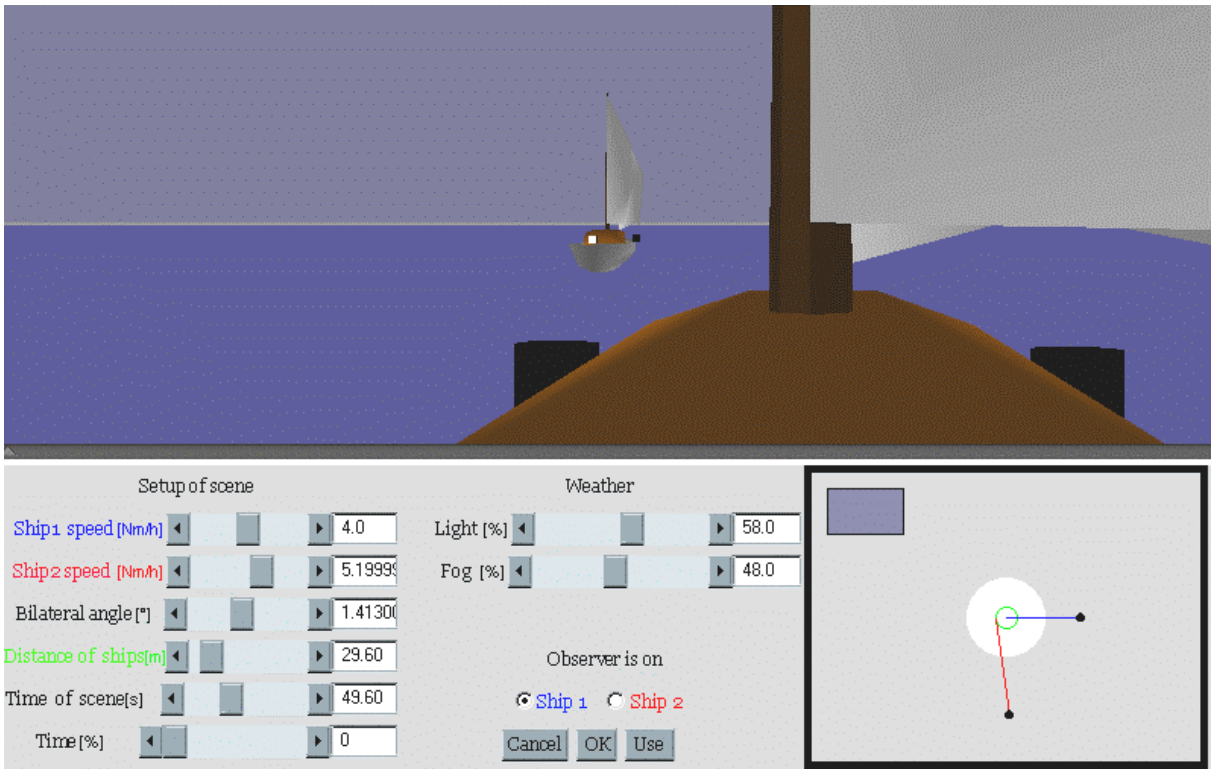

Figure 4: Boat in scene movement control and environment setting layout

3.4 Question and answer system

Questions and answer texts are stored in the external file. One question and four answers (one is right.) are randomly chosen after loading whole file. A part of every question is also name of VRML file with model of relevant ship. The ship determinate by question is displayed. Question and answers are displayed after simulation start in random sequence. The answer is validated as right or wrong until simulation closure only. After the closure answer is always validated as wrong due to timeout.

3.5 Applet of Captain training system in HTML page

The HTML file containing embedded wrl and class files has one required syntax. Field MAYSCRIPT means that the applet can get the VRML plugin browser object instance from the browser.

```
<embed src = "root.wrl" border=0 height="300" width="800">
</td><applet
        code =" Ship.class"  name = Ship
         width = 800  height = 200
mayscript>
</applet>
```

4 Details

4.1 Class hierarchy

The layer above EAI used in this project is specially modified for us project demands. The layer is realised as layer and its Class can be divided into several groups:

- **VrmlTerm** defines all constants, as are types of node, names of property, initial values etc. It doesn't contain any method to implement. All necessary constants are providing for class by Implementation.
- **VrmlNode** defines methods collective for all nodes.
- **VrmlRoot** is a child of *VrmlNode*. In addition it defines method collective for all nodes with children. It is for addition and removing children and define property and event.
- **VrmlBrowser** is a child of *VrmlRoot*. In addition it defines method for communicate with scene, for node creation, for node into/from scene addition/remove, for node creation from string and from URL address. They are used as initial parameter at generation of all nodes from string or URL address. This class creates own node.
- **Vrml…** is a child of *VrmlNode*, *VrmlRoot*, and their children. It defines method for given type node from VRML. Class Methods are analogical as properties of nodes from VRML. It contains following classes and interfaces:
  VrmlRoot(VrmlGroup, VrmlTransform),
  VrmlSound, VrmlMedia(VrmlAudioClip, VrmlMovieTexture),
  VrmlViewpoint, VrmlBackground. VrmlFog, VrmlLight(VrmlPointLight),
  VrmlInterpolator(VrmlColorInterpolator, VrmlCoordinateInterpolator,
  VrmlNormalInterpolator, VrmlOrientationInterpolator,
  VrmlPositionInterpolator),
  VrmlSensor(VrmlTimeSensor)
  The other classes of models node aren' t meanwhile implemented because weren't required for this project.
- **VrmlEventIn** defines class, which setup required node property. Usually it is generated by node during a connection using *(_)VrmlRoute*.
- **VrmlEventOut** defines class, which respond for node event. . Usually it is generated by node during a connection using *(_)VrmlRoute*.
- **VrmlRoute** defines class, which connect *(_)VrmlEventOut* with *(_)VrmlEventIn* analogical as it is in VRML. It contains values of all events type. In addition it make possible to add *VrmlEventIn* created before and after current *_VrmlEventIn.* It makes possible change value or ignore event. For example it make possible to connect output event scalar type with input event colour type. The additional value change is possible before proper value substitution.
- **VrmlEnviroment** defines class, which describe environment of scene. It make possible set up lighting, fog, background, relief and Viewpoint's event, which value will be change.
- **VrmlShip** defines class, witch describe and control ship. Set up ship model , lights on ship board and boat sound.
- **ShipView** propagates class *_VrmlViewpoint* and implement *_VrmlEventIn*. It adjust position and direction of observer along new ships position. In contrast to common event it responses to two different sources of position events. After revenue any changes it computes new angle for observer direction and set up it together with his position. Class is unchanging node Viewpoint parameter directly but per superior node Transform. At direct editing of node Viewpoint properties there are occur a time delay of rotation and observer into up short time are looking completely elsewhere.

4.2 Implementation and compilation

The applet has been implemented in Java language and compiled using JDK v1.1.5. Java uses classes stored in zip file in VRML browser directory (name of this file is npcosmop21.zip). This implementation has been tested on platform Windows 95/98/NT using Explorer 5.0 and CosmoPlayer 2.1.

5 Problems and Future work

Several problems have occurred during recognition of this application. The most difficult problem was a disappearing of a number of VRML parts within scene editing. The routes disappeared most frequently from unknown reason. This error initiated total system crash. The use of MSIE 5.0 eliminates this problem partly but under other browsers this problem remains.

We are going to further extend the system. The next version should allow a direct (on–line) communication between instructor and recipients.

6 Conclusion

Described implementation shows how to use Java and VRML cooperative system in practical application. We have shown the role of VRML and Java control applet for training and evaluation purpose. The project is leaded by a professional yacht captain and it is aimed to internet community

7 References

[1] Žára J. : VRLM 97 Laskavý průvodce virtuálními světy, *Computer Press*, 1999 (in Czech)

[2] Java Tutorial, *Sun Microsystems*,1999. http://java.sun.com/docs/books/tutorial

[3] Marin, Ch.: Proposal for VRML 2.0 informative annex, External Authoring Interface, *Silicon Graphics Inc*., 1997

[4] Haškovec J. : Rukověď námořního jachtaře – sešit 1 a 2, *Czech Offshore Yachting Association*, 1999