Development of R-Cubed Manipulation Language
– The specification of RCML and RCTP –

Wei-Chung Teng*, Akira Nukuzuma**, Naoki Kawakami*, Yasuyuki Yanagida*, and Susumu Tachi*

*School of Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 JAPAN
{waldo, yanagida, kawakami, tachi}@star.t.u-tokyo.ac.jp
**Takatsuki Laboratory, Minolta Co. Ltd.
1-2,Sakura-Machi,takatsuki-Shi, Osaka, 569-8503 JAPAN
nuu@eie.minolta.co.jp

Abstract

R-Cubed (R^3: Real-time Remote Robotics) is the concept and technology which attempts to provide a solution for people to telexist anywhere in the world by controlling remote robots as his avatars in real time base through the network. An R-Cubed system is supposed to allow people to control remote robots from terminals installed on homes, offices, or any public booths connected to Internet or other dedicated networks. When constructing an R-Cubed system, topics involved include the mechanism to model and construct interactive remote environment virtually, the manipulating interface selection discipline which depends on the characteristics of remote robots, and the network protocol for real-time communications between user side and robot side. In this paper some efforts devoted to the design of R-Cubed system are shown, including the specification of RCML (R-Cubed Manipulation Language) which describes remote environments and robot characteristics, and RCTP (R-Cubed Transfer Protocol) which is in charge of network communication between users and robots in real time base. An implementation example using mobile robot is also introduced.

Key words: Tele-existence, R-Cubed, RCML, RCTP, VRML

1. Introduction

The concept of telexistence (tele-existence) is proposed in 1980. It allows human, if emancipated from the restrictions of time and space, to "exist" in a location other than he/she really exists, or a virtual space. R-Cubed [1], stands for Real-time Remote Robotics, is proposed in 1995 as a project which try to construct a society that everyone can freely telexist through a network. That is , networked telexistence [2]. The execution image of R-Cubed system varies from controlling humanoids which have sensors equipped to provide sensory information, by dedicated devices like master manipulators and head-mounted display (HMD), to the low end systems by which users can control remote movable camera by a personal computer. In the meantime real-time data transfer channel is employed to provide a basis for users and robots to communicate with each other. That is because only with real-time response can human retains the sensation of presence [3].

In this research we try to provide a solution to the problems one would have to face when implementing a R-Cubed system on low end systems:

- A modeling mechanism that can describe a three dimension world, both in shapes and actions available for objects inside the world. The models would be used when users are immersed in virtual spaces or as an assistance when the robots can not provide enough sensory information.
- Standardization on the method in handling various user-interface devices and various controllable robot mechanisms, as to make sure they can cooperate in any possible combination. So are the sensors on robots and display devices installed in terminals.
- A network transfer protocol that provides real-time communication.

To overcome these problems, we propose a new description language RCML (R-Cubed Manipulation Language) and a protocol R-Cubed Transfer Protocol (RCTP), with their specifications be introduced in paragraphs below.

1.1 Related Works

1.1.1 VRML

Since the VRML 1.0 standard was specified in 1994, the use of VRML has become the most popular way to construct and access virtual worlds. VRML 2.0 provide

the mechanism on designing and handling the actions of objects, and has become an ISO standard known as VRML97 [4]. This provides the users stable circumstances to access a number of virtual worlds.

However, even we can simulate the real world by writing VRML scripts, it's difficult to simulate ourselves, i.e., the avatar in virtual world. In VRML the shape of avatar is restricted as a cube, with 2D or 3D pointer the only interface to interact with the world. By the way, although some implementations show the possibility on controlling virtual robot modeled with VRML [5], a standard way to model controllable mechanism and sensory information is still necessary for various types of robots.

1.1.2 Controlling Robots via WWW Browser

On the other hand, several projects on controlling actual robots or robot arms over ISDN and/or Internet are reported, with some of them opened to the Internet [6,7]. In general, the "Internet controlled robots" are based on Hyper-Text Markup Language (HTML) and Hyper Text Transfer Protocol (HTTP) standards. This kind of approach has the advantage that users can control robots by using general WWW browser. However, there are some implied problems remain:

- The types of user interface are limited to those supported by HTML language, i.e., the conventional graphical user interface (GUI). And HTML does not provide a standard way for other input/output devices such as three dimensional position/orientation sensors, which are more intuitive way to control the robots.
- Using HTML, the information exchanged over the network is treated as symbolic information attached to correspondent GUI manipulation, rather than control information itself. This is not suitable for continuously controlling robots by specifying numerical parameters such as hand position, head orientation, etc., though it may be sufficient for controlling robots by symbolic commands.
- The HTTP protocol is not designed for real-time transference.

The problems exist as long as the system is constructed based on HTML and HTTP, and any effort that try to decrease or eliminate these disadvantages within the frame of existent language and protocol may result in constrained, not intuitive systems.

2. Basic Concepts

On implementation of the RCML/RCTP system, the following concepts and definitions apply.

2.1 Server/Client Architecture

The RCML/RCTP system is of client/server architecture. Server stands for the controlling system software running at the robot site, and client stands for the software running at the user's computer.

2.2 Objects

The concept of object is introduced here to define the functional input/output units on the system. According to their functionality, the objects is grouped into three categories: *system object*, *input object*, and *output object*.

System objects include server object and client object, which refer to, in respective, the server and client's computer hardware and the RCML/RCTP software running above it, in the base that only one client can control the robot at a time. Where input object and output object mean the functional input and output unit for sending and receiving commands, data and messages between the information space created by computer, and the real world. For example, digital camera is an input object, where a monitor belongs to output object. It should be noted that it is not the device itself to be considered as a object, but the functional unit to be. So a force display equipped with 3D position sensors is an output object and at the same time an input object.

Input/output objects can still be classified into five types according to the character of data they handle: *video input/output object*, *audio input/output object*, *control input/output object*, *text input/output object*, and *binary input/output object*. Video input objects are used to get live video stream, and audio input objects are dedicated to live audio stream. Control input objects are used to get control data of various degrees-of-freedom, such as 6 DOF position/orientation sensors, 2D or 3D mice, and so on. Text and binary input objects are for arbitrary kinds of symbolic information, in text form and binary form respectively. An example of text input/output object is a command string used by command-based robot control. Each type of output object is the counterpart of corresponding type of input objects.

Table 1. Objects in RCML/RCTP system

System Object	Input Object	Output Object
Server	Video Input	Video Output
Client	Audio Input	Audio Output
	Control Input	Control Output
	Text Input	Text Output
	Binary Input	Binary Output

2.3 Input and Output Devices

The input and output devices, both in user site and robot site, are the actual interfaces which intermediate the real world and the information space. There are three ways to implement devices:

- Hardware implementation
- GUI implementation

- Virtual implementation.

Hardware implementation refers to physical devices like mouse or joystick. GUI implementation means graphic interfaces like scroll bar, button, *etc*. And virtual implementation should be applied to simulated virtual devices when users tele-exist in virtual spaces.

2.4 Translators

To design RCML/RCTP system to be applicable for various types of robots and devices, the way data transferred over the network must be standardized and be independent of the robots and/or devices connected at each site. To satisfy this requirement, software modules called *translators* are provided to translate the specific-format data into the standard format and vice versa. For example, the data returned from some 6 DOF position sensor may contain 6 degrees for 6 joint respectively. By input translator it is converted into standard format, coordinate in Cartesian coordinate system, and transferred into the other site. This coordinate would then converted again to suite the specific output device in the opposite site. For another example on sensory information, the live video stream took from robot's camera would be at first encoded into some standard format and send out by network, and then video player program in the user's site would translate this format and show it on the screen, no matter of normal display or some HMD.

Translator is independent of physical device, nor is translator equal to unit of program. A good example is the output translator for video stream in last paragraph: it includes the standard format translator and the display driver bound to the display device.

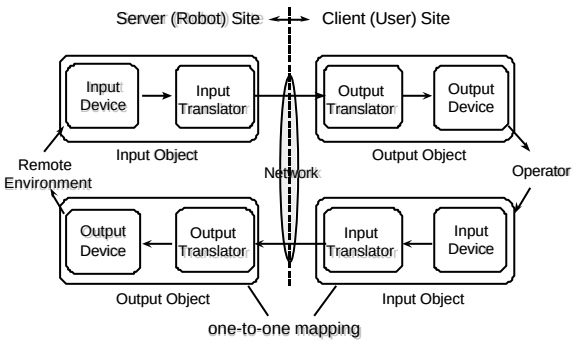

Fig.1 Functional Units and Information Flow in The RCML/RCTP System

3. RCML

3.1 Purpose

The R-Cubed Manipulation Language is designed as a file format for describing real world or virtual spaces and for robot characteristics such as degrees of freedom, control parameters, and sensor information. RCML should fulfill the following requirement:

3.2 Design Criteria

RCML has been designed to fulfill the following requirements:

- Consistency: the description methods and accessing mechanisms should be the same for both real and virtual worlds. Users control the remote robots in the same manner no matter in real worlds or in virtual spaces.
- Compatibility: RCML should be compatible and, if possible, be transparent to existent standard.
- Uniformity: there should has an universal way to describe all kinds of input/output devices so that they can be treated in the same way.
- Portability: RCML should not define or assume any special function or characteristic depending on any particular operating platform.

3.3 Design Strategy

In recent years, VRML has become a universal standard for describing virtual world. The latest specification VRML97 indeed provides a standard and popular way to construct interactive three-dimensional objects and virtual worlds. So we consider it adequate to adopt VRML97 as the base format to describe remote environments, controlled robots themselves, and robot characteristics.

In fact, RCML is designed to be grammatically compatible with VRML97. And for the robot related part, we fortunately have VRML be designed flexibly such that it could be extended easily. By describing the robot related information in the form of VRML extension node, PROTO node, RCML retains the transparency to VRML. That is to say, any legal RCML file would also be translated correctly by VRML browser, with RCML-related part neglected. So users who have only VRML browser installed can still access RCML files and browse around inside the virtual worlds.

3.4 Node Structure

A typical RCML file should include two parts: a virtual world, which is the "copy" of real remote world, written as normal VRML files, and a RCML-defined PROTO node containing control parameters and sensory information description.

The structure of RCML extension nodes are constructed in the same way with *objects*. It contains 3 level with the RCML_Robot grouping node as the "root" node. System object nodes and Input / Output object nodes are contained in RCML_Robot and stand for system objects and Input / Output objects respectively. Control objects

are classified furthermore into small unit RCML_ControlInputData or RCML_ControlOutputData nodes. Table 2 lists all kinds of nodes and shows their relationship.

Table 2. List of Nodes in RCML

Level 1 Node		RCML_Robot
Level 2 Node	System Object Node	RCML_Server RCML_Client
	Input Object Node	RCML_VideoInput RCML_AudioInput RCML_ControlInput RCML_TextInput RCML_BinaryInput
	Output Object Node	RCML_VideoOutput RCML_AudioOutput RCML_ControlOutput RCML_TextOutput RCML_BinaryOutput
Level 3 Node	RCML_ Control Input Node	RCML_ControlInputData
	RCML_ Control Output Node	RCML_ControlOutputData

3.4.1 The RCML_Robot Node

RCML_Robot node includes all level 2 nodes as parameters. A example for RCML_Robot is as below:

```
RCML_Robot {
  field SFNode  Rcml_ControlInput { … }
  field SFNode  Rcml_ControlOutput { … }
  field SFNode  RCML_VideoInput { … }
  …
}
```

3.4.2 The RCML_System and RCML_Client Node

The RCML_Server node contains information about the server's system, with the definition as:

```
PROTO RCML_Server [
  field SFString language    "RCML /1.0"
  field SFString os ""
  field SFString serversoft   ""
] {
  Script {
    field SFString  language IS language
    field SFString  os IS os
    field SFString  serversoft IS serversoft
    url   ""
  }
}
```

The *language* field specify the version of RCML supported by this server. The names of OS and software are specified in *os* and *serversoft* field respectively. And the *url* stores the URL of RCTP server. RCML_Client node is defined in the same way, so we omit its explanation here.

3.4.3 The RCML_VideoInput and RCML_AudioInput Node

```
PROTO RCML_VideoInput [
  field SFString name         ""
  field SFString translator   ""
  field SFInt32  channel      1
  field SFInt32  xsize        0
  field SFInt32  ysize        0
  field SFInt32  color        0
  field SFFloat  samplerate   0
] {
  Script {
    field SFString   name IS name
    field SFString   translator IS translator
    field SFInt32    channel IS channel
    field SFInt32    xsize IS xsize
    field SFInt32    ysize IS ysize
    field SFInt32    color IS color
    field SFFloat    samplerate IS samplerate
    url   ""
  }
}
```

The RCML_VideoInput node specifies parameters for controlling the video input device. The *name* is intended to store the name of the video input device. The *translator* specifies the name of Input translator program. The *channel* field is intended to store the number of the channel. A value of 1 means the video input is a normal one like CCD camera, and 2 indicates stereo video input. The *xsize* field and the *ysize* field are intended to store the video resolution in pixel. The *color* field is the number of colors on the input, with unit as bit. The *samplerate* field is, as what he say, the video sample rate (Hz). A sample rate of 0 means still image, or a picture. Finally the *url* field is intended to store the URL for the server of the video input device.

RCML_AudioInput node has a similar definition and is omitted here.

3.4.4 The RCML_ControlInput and RCML_Control-InputData Node

The RCML_ControlInput and RCML_ControlInputData may be the most important input object nodes. RCML_ControlInputData describes the control parameters of robot's every input object, one by one; And RCML_ControlInput groups them together.

```
PROTO RCML_ControlInputData [
  field SFString  name                " "
  field SFInt32   value_type float_type
```

```
  field MFVec2f  ralue_range          0 1.0
  field MFFloat  range_scale          1.0
  field SFInt32  control_type         h_scroll
] {
  Script {
    field SFString    name IS name
    field SFInt32     value_type IS value_type
    field MFVec2f  ralue_range IS ralue_range
    field MFFloat     range_scale IS range_scale
    field SFInt32     control_type IS control_type
    url   ""
  }
}
```

The *name* field is intended to store the name of the robot control input. The *value_type* field specifies the type of the input object, with default value as "float_type". The *value_range* field is intended to store the range of the parameter. The *range_scale* field is intended to store the size of the range. The *control_type* field is used when the input object is some kind of GUI. The *url* field is intended to store the data object server URL of the control input object. An example for a horizontal scroll bar to control a camera's pan movement is showed here:

```
RCML_ControlInputData {
  name              "pan_move"
  value_type        float_type
  value_range       -1.57 1.57
  control_type      h_scroll
}
```

As for the RCML_ControlInput node, it defines some global controll parameters and may contain numbers of RCML_ControlInputData nodes.

```
RCML_ControlInput {
  field SFInt32   time_stamp_size     0
  field SFString  time_unit           10ms
  field SFBool    data_mask           FALSE
  field SFInt32   command_mode        Time_Driven
  field SFBool    control_lock        FALSE
  field MFInt32   command_interval    -1 0
  # And any number of:
  field SFNode    Rcml_ControlInputData
}
```

RCML_ControlInput cooperates with RCTP and defines the number of parameters to be sent, the time intervals for sending data, *etc*. The detail will be introduced in RCTP section.

The *time_stamp_size* field is intended to store the size of the command time stamp, and the allowed values are 0, 2, 4 , 6, or 8 (bytes). Its default value 0 implies that no time stamp is included in the command packets. The *time_unit* stores the unit of time stamp. The *data_mask* field determines whether the system uses fixed or variable length format when sending command packets. Its default value FALSE means variable length data format.

The *command_mode* field represents the command issuing strategy for the client program. Its default value Time_Driven means that commands are sent in time driven mode. The *control_lock* field determines whether permission of the server is needed to send next command. If its value is FALSE, the client can send next command without permission of the server. When system is in time driven mode, the values in *command_interval* field indicate the minimum and maximum time interval in millisecond between adjacent command to be sent. "command_interval 10 100" means that the next command should be sent in a period of 10msec and 100msec after current command be sent. If server does not receive any command after the maximum interval of time passed, it is considered that the connection is terminated. If one of the two values in the *command_interval* field holds negative value, the command is not sent. The maximum interval must be grater than or equal to the minimum interval, or the result is not defined.

3.4.5 The RCML_TextInput and RCML_BinaryInput Node

The RCML_TextInput node specifies text input parameters to control a robot. This node has the same *name* and *translator* parameters defined as in RCML_VideoInput.

```
PROTO RCML_TextInput [
  field SFString  name         ""
  field SFString  translator   ""
] {
  Script {
    field SFString name IS name
    field SFString translator IS translator
    url   ""
  }
}
```

RCML_BinaryInput node is defined in the same way with RCML_TextInput and is not showed here.

Output object nodes are Input object nodes' counterpart, and are defined in almost the same way as Input object nodes do. The detail of their description can be found on http://www.star.t.u-tokyo.ac.jp/projects/RCML.

Finally some aliases used in the definition above are arranged in the table below:

Table 3. Aliases and their meanings

Name of Alias	Value	Meaning
float_type	0	The float type
integer_type	1	The integer type
bool_type	2	The Boolean type
h_scroll	0	The horizontal scroll bar
v_scroll	1	The vertical scroll bar

Time_Driven	0	The command is issued at constant interval
Event_Driven	1	The command is issued when a user command is revised

4. RCTP

4.1 Purpose

RCTP is designed to cooperate with RCML. The two main jobs for RCTP are to negotiate for assigning effective pairs of input/output devices in robot and user sites, and to transfer the control commands from user and status information from robot in real-time.

4.2 Scenario of RCTP Connection

The process of a RCTP connection is divided into 3 phases. In the beginning the users access some Web pages describing the information of controllable robots, and then download the RCML file. RCML browser is invoked at this time to parse the RCML file. This part is called *the greeting phase*. Computers that have no RCML browser installed may act as if a VRML file is download and invoke the VRML browser.

After the user picks out the controlling objects and decides the corresponding input device, RCML browser would try to build a network connection with RCML server. After connection is established, RCML browser would request to get the control permission of robot's objects. If the requested device is free and functions well, server will assign a unique ID number to the device and acknowledge it back with all initial characteristic values appended. All this process should follow the specification of HTTP/1.1, and the part is called *the negotiation phase*. If negotiation phase completes smoothly, RCML browser would send the GO method in order to start the controlling phase. Control messages and system information transferred in this phase are binary based and contrived to meet the real-time needs, however the Video/Audio and other Sensory information are left to dedicated protocols. This phase is call as Live Session Phase. Figure 2 shows the whole process of a RCTP connection:

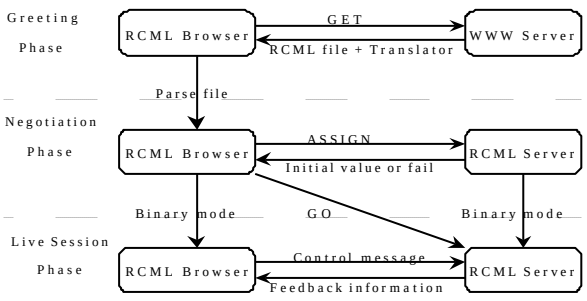

Fig.2 Scenario of RCTP Connection

4.3 Phases in RCTP

The process of a RCTP connection is divided into the following three phases which correspond to their functionality.

4.3.1 Greeting Phase

In this phase users access URL of RCML file, download the file and relative translator modules which may be written by Java language. All these jobs can be done by a WWW browser, so this phase has no relation to RCTP in technical.

4.3.2 Negotiation Phase

The negotiation phase happens when users want to control remote robots. When the client requests for connection, the server checks if other client is controlling the robot. Because only one client is allowed to control the robot at a time, server may refuse to the control request by a server busy message. However, users can still request for sensory information for a robot controlled by other user. In this phase, the client assigns controllable objects at the remote site with the available devices, as well as assigning remote sensory information channel with the local output devices. For example, if the server site has a Control Output object with 6 DOF, the client may attempt to assign a Control Input object with 6 DOF. This assignment procedure may be regarded as "making the pair of input and output objects".

Since HTTP plays a good role in negotiation and querying, the format used in negotiation phase follows the HTTP/1.1 specification [8], with two RCTP extended methods are employed here though. The ASSIGN method request for control permission for one object of remote robot, and the GO method is sent when negotiation ends smoothly so that the system can switch over into the next phase. Example is given below:

(Client) ASSIGN head RCTP/1.0
CRLF
(Server) 201 OK
ID: 1
Current Value: 30.0 10.0 5.0
CRLF
(Client) ASSIGN arm RCTP/1.0
CRLF
… (the ASSIGN method is repeated)
(Client) GO RCTP/1.0
(Server) 201 OK
(now starts the live session phase)

4.3.3 Live Session Phase

Data transferred through the network can be categorized according to the object type they belong to. Because

technique for live video/audio broadcasting have been developed for several years and obtained remarkable improvement, transfer protocols to video and audio objects are open to the existent ones. As for control objects, a compact format is used for network packet.

In general situation multiple control input/output object pairs exist and share the same parameter recorded in RCML_ControlInput and RCML_ControlOutput nodes, such as time unit and commend interval. So it gives us a good reason to collect all commands from control input objects together at the same packet when we send them to the server. The same technique applies on data from Text and binary output, which may be in charge of reporting the status of robots. Detail of the packet structure is showed as figure 3.

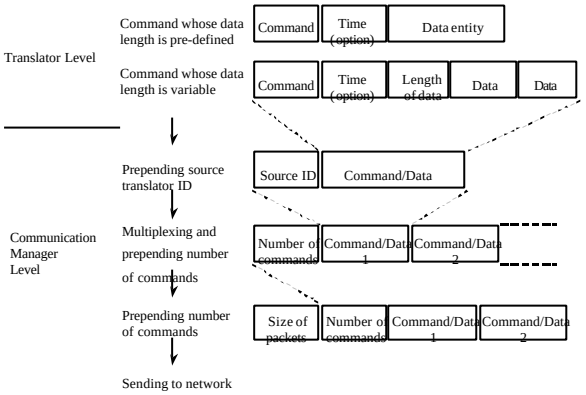

Fig.3 Multiple Layer Packet Formation

Another characteristic on the format of packet is the usage of data mask. Definition of Control object message can be written in BNF as:

Control = ObjectID [TimeStamp] [DataMask]
ControlData
ObjectID = OCTET
TimeStamp = 4OCTET
DataMask = *OCTET
ControlData = 1*OCTET

ObjectID is the ID number assigned in negotiation phase, and TimeStamp is the time stamp representing the length of time period after the live session starts. TimeStamp is an optional field and would not be used if the value of *time_stamp_size* field in RCML_ControlInput node is 0.

DataMask is an array of flags showing if the corresponding parameter be sent, with each parameter a bit. The unit of DataMask is byte, and the length will be the round off after dividing number of parameters by 8. So if there are 14 parameter in a object, the DataMask for that object would be 2 bytes. Data mask is used if the value of *datamask* field in RCML_ControlData node is TRUE.

Now consider controlling a 13 DOF robot arm with data mask used. If user change the angles of the 2nd and 7th articulation, the command sent to server is as below:

Control = ObjectID 0x0042 ControlData2 ControlData7

Size of packet is considered shorten by using data mask in multi-parameter control objects.

Finally ControlData means the set of parameters for that ControlData object.

5. Implementation example

An implementation based on RCML/RCTP specification is constructed and will be shown in the conference as a demo. The following is specification of the demo system:

- System architecture: Client/Server architecture.
- Hardware and computer OS:
  - Server: Toshiba Libretto 60 with Windows NT Workstation 4.0 installed
  - Client: IBM/PC AT compatible with Window 95/98 installed
  - Robot: the Minolta robot
- Development environment:
  - Server: Microsoft Visual C++ 4.0 or above
  - Client: JDK 1.1.6 and Visual C++ 4.0
- Helper programs:
  - Video player: CU-SeeMe v1.0
  - VRML Browser: Sony Community Place Browser 2.0

6. Conclusion

In this paper the overview of RCML and RCTP is stated. RCML and RCTP are designed to mutually cooperate in order to implement an low end and general purposed R-Cubed system, with basic concepts like translator and object be introduced.

Another characteristic of RCML and RCTP is the compatibility to the existent language and protocol. RCML is designed in the base of VRML97, and RCTP extends HTML/1.1 as the standard format in negotiation phase.

Finally, the complete specification can be refer by the following URL:
http://www.star.t.u-tokyo.ac.jp/projects/RCML

References

1. MITI of Japan, R-Cubed WG ed.: “R-Cubed”, Nikkan Kogyo Shimbun (1996)
2. S. Tachi, Real-time Remote Robotics – Toward Networked Telexistence, IEEE Computer Graphics and Applications (1998), pp. 6-9.
3. Y. Yanagida, N. Kawakami, and S. Tachi, Development of R-Cubed Manipulation Language - Access Real Worlds Over the Network, The 7th

Internaional Conference on Artificial Reality and Tele-existence (1997), pp.159-164.
4. http://www.vrml.org/Specifications/VRML97
5. http://www.robotic.dlr.de/STUDENTS/Martin.Rohrmeier/robot/robot.html
6. http://www.usc.edu/dept/raiders/story/index.html
7. http://www.cs.cmu.edu/~xavier/
8. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T., “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2068, UC Irvine, Digital Equipment Corporation, M.I.T., January, 1997.