FINAL REPORT

ARGE
Institute für Praktische Informatik (Prof. Ferscha)
und für Technische Informatik (Prof. Volkert)
an der Johannes Kepler Universität Linz
Altenberger Straße 69
4040 Linz

A. Ferscha, J. Volkert, M. Pitra, S. Vogl, D. Kranzlmüller,
W. Narzt, V. Christian, P. Heinzreiter

February 24rd, 2003
Abstract

VRIO (Virtual Reality Input/Output) is a framework for ...
# Contents

1 Motivation ................................................................. 6
   1.1 Problem Statement .................................................. 6
      1.1.1 Natural Interaction .......................................... 6
      1.1.2 Mobile and Remote Interaction ............................ 7
      1.1.3 Synchronous and Delayed Interaction ..................... 7
   1.2 The VRIO Project .................................................... 7
      1.2.1 Framework Characteristics .................................. 8
      1.2.2 Scenarios .......................................................... 9
   1.3 Mobile Interfaces For Ubiquitous Interaction .................... 9
   1.4 Introduction to Speech Based Interaction ........................ 11
      1.4.1 Speech Application Structure ............................... 12
      1.4.2 Speech and Application Domains ............................ 12
   1.5 Limits of Speech Recognition ..................................... 13
   1.6 Related Work .......................................................... 13
      1.6.1 Nomadic Radio .................................................. 14
      1.6.2 Impromptu ....................................................... 14
      1.6.3 PUC ............................................................... 15
      1.6.4 How May I Help You? .......................................... 15
      1.6.5 SpeechWear ..................................................... 16
      1.6.6 The eSleeve .................................................... 16
      1.6.7 Discussion ........................................................ 16
   1.7 Structure of this document ........................................ 17

2 Architecture ............................................................ 18
   2.1 The VRIO Framework ................................................. 18
      2.1.1 Architecture ................................................... 18
      2.1.2 Components .................................................... 18
Contents

2.1.3 Hardware and Software Requirements ........................................ 20
2.1.4 Data Structures ........................................................................... 22
2.1.5 Server Database .......................................................................... 24
2.1.6 Demo Applications ....................................................................... 24
2.2 Using the VRIO Framework Applications Programmer’s Interface .............. 25
  2.2.1 XML Requests ........................................................................... 25
  2.2.2 The C++ API ............................................................................. 27
2.3 The Embedded ViaVoice SDK ............................................................ 30
  2.3.1 Architecture ............................................................................. 30
  2.3.2 Hardware and Software Requirements ....................................... 32
  2.3.3 Design Guidelines ...................................................................... 33
  2.3.4 Decisions According to VRIO Design ....................................... 35
2.4 Using the Embedded ViaVoice SDK Application Programmer’s Interface ..... 35
  2.4.1 Static Vocabularies .................................................................... 36
  2.4.2 Creating Run-Time (Dynamic) Vocabularies ............................... 38
  2.4.3 Speech Recognition and Synthesis .......................................... 39

3 Scenario 1 ................................................................................................. 41
  3.1 Structure of the Virtual Reality environment .................................. 41
    3.1.1 Hardware Components of the Holodeck Application .................. 41
  3.2 Operation of the Holodeck Application .......................................... 42
    3.2.1 Initialization ............................................................................ 42
    3.2.2 Activities within the Virtual World ......................................... 42
    3.2.3 Object Modifications ............................................................... 44
    3.2.4 Example of Holodeck Application .......................................... 45

4 Scenario 2 - Multiuser Interaction via VoiceWalls .................................... 47
  4.1 WebWall: Multi-User Interaction on Wall Displays ........................... 47
    4.1.1 Wall Computing ....................................................................... 47
    4.1.2 Limitations of Public Communication ..................................... 49
    4.1.3 Public Communication on Digital Walls ................................... 50
    4.1.4 The WebWall Framework ......................................................... 52
  4.2 Service Classes ............................................................................... 56
    4.2.1 Note ....................................................................................... 56
    4.2.2 Gallery .................................................................................. 57
  4.3 The VRIO—WebWall Interface ......................................................... 58
  4.4 Interactive Walls ............................................................................. 59
## Contents

4.4.1 Why Do Walls Need Voice? ........................................... 59
4.4.2 State Diagram ......................................................... 60
4.4.3 Commands in Detail ................................................... 60
4.4.4 Command Input Modes ............................................... 64
4.4.5 Embedding the application in the VRIO framework ................. 64
4.5 Demonstration .......................................................... 65

5 Achieved Goals ............................................................. 72
5.1 Software Goals .......................................................... 72
5.2 Application-oriented Goals ............................................. 72
5.3 Papers ..................................................................... 73

6 Summary and Future ....................................................... 74
6.1 Future Enhancements .................................................... 74
   6.1.1 Future Research ................................................... 74
   6.1.2 Voice Applications ................................................. 75
   6.1.3 Extensions to the Server ......................................... 76

A Glossary ................................................................. 78

List of Figure ............................................................. 80
List of Tables ............................................................... 81
Bibliography ................................................................. 82
1 Motivation

Traditional user interfaces like mouse and keyboard are input devices that have been optimized for people who manipulate 2D data like texts, images, spread sheets. For other people, like mobile workers, as well as for different tasks, these tools provide only sub-optimal support for interaction. Many research projects in Human Computer Interaction (HCI) have investigated alternative interface techniques and tools that are better adapted to the situation of moving people that need to control digital devices.

This interaction problem has resulted in a number of research efforts and alternative input devices, like wands, trackers, space-mice, gesture recognition among others. These tools share one common property: at least one hand is necessary for input, a resource that is often not available easily.

The VRIO framework tries to provide a better mode of interaction through a flexible system for speech-based interaction that enables users to control any device hands-free with a digital interface.

1.1 Problem Statement

Mobile workers often need to interact with digital devices while they are moving around. It is often impractical, if not impossible, to interrupt the current task, walk to a computer terminal, enter a request and continue with the original work. This project aims at relieving workers from these tasks by enabling them to control devices using voice input while on the move.

1.1.1 Natural Interaction

Depending on the type of task, people use highly specialized tools for effective solutions. While performing physical operations, switching to a computer display and keyboard is often a tedious and time-consuming operation. Nevertheless, computers can provide important information that aid in solving these tasks or can provide additional information.
1 Motivation

To aid users in such situations, adapted input devices need to be developed that keep the user’s hand free for their core accomplishments. One approach to this problem is to replace traditional input devices like mouse and keyboard with other artifacts that are better suited to the problem domain like tangible interfaces [IU97]. Interaction is enabled through digitally connected physical artifacts that fit seamlessly into the problem domain.

Graspable interfaces are nonetheless impractical in many situations, especially when the user is moving about or is in need of both hands. For these situations, other input channels have to be used, like voice interfaces. Speech recognition can be used for input, and text-to-speech for user notification, which eliminates the need for (physical) input devices and displays.

1.1.2 Mobile and Remote Interaction

Devices that are connected to a network can not only be controlled locally, but also from remote locations, if the right hardware and software infrastructure has been deployed. Building on the Internet and associated communication standards, many devices already are or can be connected so they are controllable at a distance.

Using mobile networked input devices, appliances and software components may not only be influenced at a distance, but also while the user himself is moving. The availability of wireless networks in a growing number of places frees people from staying at a terminal, making people stay connected as long as they are in the coverage area of a network.

1.1.3 Synchronous and Delayed Interaction

While many tasks need to be completed instantly, it is often desirable to schedule actions so that they are executed at a later time. Relieving users from the necessity to remember actions and recall them sometime in the future, it is highly desirable, as forgetting about them is a common source of error. This can be overcome by enabling users to schedule requests that are executed at a future point in time.

As an additional benefit of delayed execution, users need not stay connected but may still influence devices. This store-and-forward behavior is especially useful for moving people that have only sporadic network coverage.

1.2 The VRIO Project

The results presented in this paper can be summarized as follows: We have implemented a flexible framework, the VRIO framework, that enables speech control of arbitrary devices for
mobile users, and implemented several scenarios that demonstrate the use and features of the VRIO framework.

1.2.1 Framework Characteristics

The VRIO framework enables mobile users with the help of a mobile client to control devices at a distance by issuing voice commands. The clients send requests to the VRIO server which is responsible for managing users and devices. The devices are represented as actuators, software components that expose identification and public commands of a device. Besides enabling remote and immediate and delayed execution, the framework characterized by the following properties:

System Structure The framework is built around three major classes of entities that are involved in the communication process: Clients enable implement the user interface for controlling devices. The controlled devices are represented by actuators, which enable control over the given piece of hard- or software. A server manages status information for both clients and actuators and routes requests between the two sets of entities.

Portability & Device Independence The framework incorporates machine-independant data structures for communication using XML structures. This enables programmers to easily add new components to the framework regardless of their system architecture or communication infrastructure. While the standard information exchange uses TCP/IP, HTTP and other Internet-protocols, the required data packets could also be exchanged using other communication methods. This is especially important due to the wide range of different mobile clients and different devices that are available.

Extensibility The communication between mobile clients and the devices they shall control is managed by a server that is responsible for routing the requests. Extending the system with new clients or devices is easily feasible as possible commands are defined in a description language that the server reads dynamically.

Scheduling A scheduling mechanism in the server enables requests to be executed at a given time. Requests can be executed at arrival time or postponed to a latter time. Besides this feature, the system has been prepared for real-time scheduling by adding a deadline for each request.
Dynamic Configuration  Components in the framework (clients and actuators) register at the server dynamically to request or provide system services. As requests address the devices they want to interact with using symbolic names, the requests can dynamically be rerouted when the addressed device has changed its network identification.

1.2.2 Scenarios

Using the framework, two scenarios have been implemented to demonstrate the features of the framework in two application areas.

GUP Scenario: VR Editor  The GUP scenario illustrates the use of the framework in immersive virtual reality settings, where a mode-switch between virtual and traditional input devices (from tracker to keyboard, for example), is a very costly operation.

IPI Scenario: VoiceWalls  The second scenario demonstrates the integration of VRIO with WebWall, a complex communication framework which enables access to multimedia content on large public interaction areas (walls). In combination, the two frameworks provide speech-based interaction on wall displays, large interactive areas.

1.3 Mobile Interfaces For Ubiquitous Interaction

This projects deals with the situation of mobile users who need to interact with devices, both physical as well as pure software artifacts. An analysis of the working environment of these users leads to a number of assumptions that are essential for the system design of the VRIO framework:

1. mobility means that a user interface has to be reachable for the user, so interaction becomes possible.

2. which means that a) the user has to be always in reach of an interface or b) he carries a device that enables interaction.

3. 'interact with devices' means that devices must be controllable (using some intermediary software).

The decision upon a wearable device in favor of a static user interface in the environment is based on the following thoughts: Providing interfaces in the environment, in the form of kiosks, interactive walls, PCs etc., limits users to the region where these interfaces have been
1 Motivation

installed. These access points are shared between a number of users, resulting in possible bottle-necks, as this resource needs to be shared.

By keeping the user interface close to the user (by means of a cell phone, PDA or wearable computer) users are only limited by the reach of the underlying wireless network infrastructure, which can be extended far more easily than deploying new static devices. The user may interact with devices at any time and place, as the interface is always in reach and available.

Keeping the interface within reach of the user means that he needs to carry some digital, networked device for interfacing with the rest of the world. As far as these mobile device are concerned, a wide range of options is available with highly diverse system properties, from small-screen, low-computational power smart phones up to full-fledged wearable PCs with high-resolution screens and fast CPUs.

A look at the effects of Moore’s law on different system components (cf. Fig. 1.1) shows that CPU speed and memory are growing at exponential speeds, while wireless network bandwidth is lagging behind. As a logical consequence of these trends, personal computing devices are gaining on local computational power while network throughput and latency is limiting the connectivity to the outside world.

![Figure 1.1: Effects of Moore’s Law. Graph courtesy Thad Starner [Sta02b].](image)

An important exception to this trend of increasing performance is battery capacity - it has increased only linearly the last years, and thus limits the amount of mobile processing: The more computation the more battery weight. Therefore, powerful wearable computers are rather heavy pieces of equipment due to their power needs.
As a ‘golden middle’, using high-end PDAs seems a viable solution between these restrictions: They do not use too much battery, resulting in lightweight devices that can be easily carried along, but bring along enough resources for sophisticated local computation like speech recognition or dynamic interface creation. This lets us minimize the effects of the slower intermediary network while maximizing functionality and availability of the mobile system.

1.4 Introduction to Speech Based Interaction

For humans, speech is a natural exchange of information that happens effortlessly. Nonetheless, it is a very complex process that incorporates many of our abilities like the recognition of a very large vocabulary and making sense out of the transmitted words using nonverbal information as well as one’s own history and knowledge.

Unlike traditional interfaces, where user input appears as a discrete event in a system, voice recognition is a stochastic process with varying recognition rates, as the same word spoken twice will never sound (exactly) the same, caused by varies intonation due to stress or emotion, background noise, etc. This complexity poses severe limitations on current speech recognition algorithms for computer systems.

McTear [McT02] provides a comprehensive introduction to speech recognition and speech interfaces. He defines three classes for speech applications based on rising interaction complexity:

**Dictation systems** that are only responsible for transcribing voice input into written text without further interaction

**Command & Control systems** enable users to interact with computers using simple commands. System reaction is limited to executing the action or reporting that the command was not recognized. This simple interaction strategy has already entered consumer-devices like voice-activated mobile phones.

**Spoken dialogue systems** provide user interaction using natural language dialogues. They can operate with a limited vocabulary that can consist of a small set of words and digits up to large vocabularies, involving increasingly complex dialogue management.

An earlier introduction to speech based systems is given by Bernsen [NOBD97], who describes a layer model of system components and different concepts for dealing with user input.
1 Motivation

1.4.1 Speech Application Structure

Dialogue-based applications are based on one of three main strategies to deal with user input:

**Graph based systems** control user input by a set of dialogue states. The transitions denote the various paths the user can take through the application states. The system prompts at each state and waits for one input at a time, which is used to change the state of the system.

**Frame based systems** keep a template with ‘slots’ - data the user has to provide. Dialogue flow is not predetermined, but depends on the missing data, which the user may provide in any order. These systems can offer flexibility for information services, where more than one data item needs to be known before the request can be fulfilled.

**Agent based systems** employ AI systems to determine possible interaction by incorporating rules that operate on a (possibly large) knowledge base. It incorporates a complex user model for effective dialogue creation and analysis.

Applications involving dialogue systems need to be carefully designed to deal with possible errors, as speech input is highly error prone compared to traditional input devices. Supporting “Undo” wherever possible and assuring the user that the system has understood are vital aspects of speech interface design.

1.4.2 Speech and Application Domains

The complexity classes listed in the previous section describe only ‘classical’ applications, where a user interacts with an application in a specific application context. It concentrates on speech recognition and navigation strategies alone, neglecting the necessities of different application domains.

**Pure Audio Interfaces** Interaction using audio as the only way of interaction can be found in a number of application domains: Telephone based information and booking systems are a very common solution. Wearable computing projects also make use of audio in cases when no display is available or to provide non-intrusive interaction possibilities.

**Multimodal Interfaces** Besides systems that use audio as their only interaction channel, there exist many application areas where voice is used as one of many ways of interacting with a system. These multimodal interfaces enable natural interaction by letting users freely choose the input modality, whatever is best adapted to their situation.
Non-Speech Audio Interfaces  Besides interaction using speech (recognition or text-to-speech systems), non-speech audio interfaces are an area of vivid research. Especially sound output where data sets or events are mapped to different audio events can provide important system state information. This process is called sonification [Son03], and provides an additional output dimension when interacting with complex data sets.

Non-voice user feedback is also a valuable source of information for wearable and mobile computing in general [SS00, PBH02].

1.5 Limits of Speech Recognition

Despite the advances in speech recognition over the past years, this mode of interaction is still highly error-prone and limited. The limits are either caused by technical shortcomings or by physiological factors.

Shneiderman [Shn00] describes an important cognitive limitation: Problem solving and speech compete over the same resource, the short-term memory, while problem solving and motoric activities like pointing and clicking do not interfere. This causes difficulties when users are forced to recall data while they have to issue voice commands.

Another gap in the voice-based human-computer interface is prosody, or speech melody of whole sentences, which does not only carry important semantical data, like the rising tone for questions, but does also carry important emotional information.

Another problem described by Starner [Sta02a] are the effects of ambient noise levels that cause humans to automatically speak louder in the presence of growing background noise (‘Lombard speech’), another burden for recognition algorithms. As Starner points out, speech recognition interfaces need to be designed in a way to gracefully react to errors:

When the accuracy is low, the penalty to the user for errors should also be low.
[Sta02a]

Consequently, speech interfaces can be used best in applications with a clear structure and limited vocabularies. User interfaces need to be designed so the user can be certain that the system has succeeded in understanding his commands.

1.6 Related Work

Besides a number of research projects that deal with various aspects of voice recognition and its application, standardization efforts have started as well: The Voice Browser Working Group [W3C03] is defining a set of (XML based) markup languages that address dialog,
speech synthesis, speech recognition, (telephone) call control and other aspects of interactive voice response applications.

1.6.1 Nomadic Radio

The nomadic radio project [SS00] at the MIT Media Lab is a wearable computing project that explores purely audio-based interaction for mobile workers. Nomadic radio uses a limited grammar for voice-based browsing of personal messages, email and calendar and access to news broadcasts. Spatialized audio output is used to provide the user with additional cues about importance and nature of the transmitted data. The hardware setup shown in Fig. 1.2 incorporates a Nortel prototype of a shoulder-worn stereo speaker and microphone combination.

![Nomadic Radio hardware setup](image)

The software system is built upon the Watson speech recognition library by ATT [Wat02], and uses different capture modes depending on the environment: In quiet environments, continuous audio monitoring constantly listens for commands. For noisier surroundings, a push-to-talk interaction technique is needed for acceptable recognition rates: Users push a button which activates the recognition process for a number of seconds.

1.6.2 Impromptu

Impromptu [Lee01] is a project of the Speech Interface Group at the MIT Media Lab dealing with audio services in wireless networks. A networked mobile device (a PDA, Fig. 1.3) is enriched with an audio interface able to control a number of applications like radio, mp3 playback and voice telephony. It incorporates a user notification system where events from the different applications compete with each other to gain the user’s attention.

Impromptu acts as a thin client with respect to speech processing: The audio signal is recorded at a mobile client (a PDA), and sent via a WLAN connection to a server responsible
for speech recognition. For other applications, like telephony, audio streams are routed directly from one mobile client to another.

1.6.3 PUC

The personal universal controller (PUC, [NMH+02]) is an approach for improving the interfaces to complex appliances by introducing an intermediary graphical or speech interface. A PUC engages in two-way communication with everyday appliances, first downloading a specification of the appliance’s functions, and then automatically creating an interface for controlling that appliance. The specification of each appliance includes a high-level description of every function, a hierarchical grouping of those functions, and dependency information, which relates the availability of each function to the appliance state. Dependency information is used for automatic interface generation, both graphical and speech versions.

1.6.4 How May I Help You?

The ATT project ‘How May I Help You?’ [GAA+02, GRW97] focuses on providing natural-language dialogue systems. It enables telephone services to use a near-to-natural language as user input, in contrast to the commonly-used ‘for help, dial 1’-style of acoustic menus.
While this project operates with thin clients (mobile phones), it enables a very reactive, flexible approach to interaction with electronic services.

### 1.6.5 SpeechWear

The SpeechWear project [RRT96] at CMU aids mobile users at technical inspection tasks. It consists of a notebook with a speech recognizer (the CMU Sphinx library) that is used to navigate HTML documents. The documents are enriched with special tags that define the possible voice input to dynamically create the input language grammar for each page.

### 1.6.6 The eSleeve

The Bristol eSleeve [RM02] is an arm-worn wearable computer. The platform is built around a wristwatch computer (the Matsuch com onHandPC) and reads data from various context sensors like a GPS receiver, a compass and a speech recognition module. After first attempts to use the speech-recognition module for menu navigation, it has been abandoned due to high error rates, both from environmental noise but also because of different user intonation, which has grave effects on recognition performance.

![Figure 1.4: Bristol eSleeve arm-worn wearable computer. [RM02]](image)

### 1.6.7 Discussion

Impromptu uses a remote server for voice recognition and supports only audio applications. The personal universal controller provides only direct control of devices without an intermediary server, which heavily limits its flexibility. Although mobile, SpeechWear is a standalone application used to browse (static) documentation. The eSleeve has stopped speech input, and is not connected to the outside world.
Motivation

The VRIO framework, in comparison, provides device control with speech recognition on the mobile clients using a flexible client-server architecture that supports scheduled requests and dynamic reconfiguration of the involved components.

1.7 Structure of this document

The rest of this document is structured as follows: Chapter 2 introduces the architecture of the framework, its major components, and its APIs.

Chapter 3 describes an application scenario developed at GUP, where the VRIO framework is used to control immersive environments.

Chapter 4 describes the integration of VRIO with the WebWall framework, which enables multi-user interaction on large surfaces. VRIO is used to manipulate objects on these displays, enabling multiple users to interact simultaneously.

Chapter 5 sums up the achieved goals and chapter 6 provides an outlook on future research and application directions that have been found promising.
2 Architecture

2.1 The VRIO Framework

The VRIO framework enables the development of applications on arbitrary end devices and capsules data transport, the mapping of clients to actuators and the XML representation. Using the framework, the development process of new clients and actuators is very simple.

2.1.1 Architecture

The aims of the VRIO framework are:

- giving arbitrary client devices the possibility to talk to
- arbitrary actuators via
- an intelligent mapping stored on the VRIO server with
- no communication limits

Figure 2.1 on page 19 shows the architecture of the VRIO framework. There are three major components in this architecture:

- Clients provide a user interface to arbitrary actuators
- The VRIO Server manages data transport and maps commands to controls
- Actuators evaluate the given input

2.1.2 Components

Clients

The clients (shown in figure 2.1 on the left side) provide an interface for people to interact with the framework. This interaction can be done manually, like filling out webforms, using
The main drawbacks of existing systems are

- the user is limited to specific devices
- human-computer interaction is not intuitive
- inconsistent user interfaces
- special training is necessary to handle the devices

Because of these drawbacks there exists another way of interaction with the framework, namely a speech driven interface. Spoken words are easy to use without training the user.

**The VRIO Server**

The VRIO server (shown in figure 2.1 in the middle) has some complex tasks to perform, such as interpreting commands received from clients, transform them to controls, and delivering them to the correct actuators. Further it manages client and actuator devices to provide...
some framework internal namespace. Additionally it provides a mechanism to schedule the
delivery of controls, in case the server cannot reach the actuator device or a specific time for
evaluation of the control is set.

The cloud above the server block (shown in figure 2.1 is a semantic representation of
some network the server is connected to. This can be the Internet, an enterprise internal
intranet or something else. The server has a web interface to perform administrative tasks
in the server database.

**Actuators**

Actuators perform the tasks the user scheduled. They are controlled by the same API that
the client uses. They can be simple physical devices like a coffee machine or a light switch,
as well as more complex devices like a web server. Of course actuators can be the same
device as the client. In this case the device acts as sending and recieving station for VRIO
requests.

### 2.1.3 Hardware and Software Requirements

There are some hard- and software requirements for the involved components of the VRIO
system. Common requirements are

- A (wireless) network interface
- A (wireless) network infrastructure
- An OS that supports delivering applications in common (object oriented) languages
  like C++ or Java to embed the VRIO framework

**Clients**

The following properties declare a client device that can be used by the VRIO framework:

- a computer based (embedded) device that owns
- a microprocessor
- a non-trivial operating system
- a user interface
- a network interface with a TCP/IP stack
• nice-to-have: an audio interface for input via microphone and output via speaker or headphone

Common clients often have the form of handheld or embedded devices like the Compaq iPAQ with enough CPU power and an adequate amount of memory. To realize scenario 1 and 2 as described in chapters 3 and 4 we used HP/Compaq iPAQ 3870 with a 200 MHz intel strongARM microprocessor and 64 MB flash RAM. Additionally this device owns a microphone and a speaker for audio in-/output. With the optional PCMCIA expansion pack the device has a slot for additional hardware. In our setting we used the slot for a Symbol WirelessNetworker compact flash WLAN card. A typical VRIO iPAQ is shown in figure 2.2.

![Figure 2.2: A typical client device](image)

**Server**

The Server has to store lots of data, so the best way to keep backups and manage fast access to stored data is using a database management system (DBMS). The DBMS used should accept SQL-style queries to manipulate data.

To allow configuration and administration tasks from remote hosts the server should own a webserver and some scripting facility like PHP or Java servlets. The setting used for the ongoing project involves a Gericom Masterpiece notebook running Debian GNU Linux with Apache webserver and PostgreSQL DBMS. A picture of the server is shown in figure 2.3.

**Actuators**

There are barely any requirements for actuators except the common ones. Actuators represent the piece of hardware performing the user’s intention. So the actuator just has to recieve the control request and interprets a piece of code that acts in the way the user intended to.
Of course actuators can be much more complex devices, there are no limitations in what the actuator should do on receiving a control request he understands.

2.1.4 Data Structures

Data going from the client to the VRIO server, and also going from the server to some actuator is represented in XML, based on a XML schema (DTD). XML is the standard format for exchanging document-type information. Because it is lightweight, easy to understand and expandable, it is the first choice for the VRIO framework.

XML Representation

Both commands and controls fulfill the XML schema shown in figure 2.4 on page 23, the tags are explained in the following section.

**Type** Each XML request has a type to identify the meaning of the request. The different types are specified in chapter 2.2.1
2 Architecture

Source  Every XML request has a unique source (specified by name and URI). This is the name and the unique address of the client device.

Destination  The destination tag means the actuator the client wants to evaluate the control the server maps from the given command.

Owner  The owner is a representation of the user issuing the command. This may be a human being but it is not a must, embedded devices with sensors acting on some algorithm are also owner of the issued command.

ExecutionDateTime  The ExecutionDateTime tag specifies a timestamp at which time the command/control has to be evaluated at the actuator. This includes the need for delivering the control to the actuator before that timestamp.
2 Architecture

**Deadline**  This is a timestamp (not mandatory). The issued command sent through the XML request must be evaluated before the specified time, otherwise it will be dropped.

**Content**  The mandatory tag in the XML schema is the content tag, where the client stores its data. The data being transported can also contain some variables that are evaluated at runtime on the VRIO server.

2.1.5 Server Database

On the VRIO server resides a database that holds the commands that the framework understands and the controls that are sent to the actuators. Also there is the mapping table for definitions which command activates which control. Altering this table means changing the behaviour of clients and actuators without need for reconfiguration.

2.1.6 Demo Applications

On the CD shipped with the report there are some demo applications showing the major advantages of the framework. They are built for the Compaq iPAQ using a wireless network infrastructure via WLAN cards and access points. A detailed system architecture and installation instructions can be found in the mid-term project documentation \[FVP+02\].

**Simple SMS Application**

First, there is an application that sends and receives short messages to and from other clients via the VRIO framework. This is an example where client and actuator reside on one physical device, being combined in one single application.

The motivation for this application was to show the benefits from the VRIO architecture. Arbitrary SMS application clients can communicate with other client users. With conventional SMS techniques via cell phone the user has to type in the message using a small keyboard (rarely more than 10 keys). Using the SMS application and enabling voice input this means more flexibility.

**CAVE Navigation**

Second, there is an application that controls a 3D scenario shown in a CAVE environment. The conventional method of navigating through the scene is by using a 3D mouse that is hard to understand and gives no feedback. Second the 3D mouse is wired to the CAVE enviroment.
2 Architecture

With this application users can navigate through the landscape via commands that are transported to the display via the VRIO framework, entered in a wireless device such as the Compaq iPAQ.

2.2 Using the VRIO Framework Applications Programmer’s Interface

There exists an applications programmer’s interface for using the VRIO framework when developing computer programs. Because there exists only a semantic difference between client and actuator, the API is for both of these involved end devices. The API is platform independent, as there exists an implementation in Java, as well as some libraries written in C++. This C++ implementation compiles under various platforms, such as Pocket PC 2002 (Windows CE), Linux, IRIX. The Java version is preliminary and untested, but should work as well.

2.2.1 XML Requests

As shown in figure 2.1 on page 19 there is a data flow from client to server and from server to actuator (and of course vice versa). The server is not mandatory for data transport, in simple installations it is possible that VRIO requests go only from client to actuator. The VRIO framework distinguishes between command requests, that flow from client to server and control requests that go from server to actuator. A third class are general requests used for initialization, metadata handling and connection tracking.

There are examples of each request type in the following sections, showing a typical VRIO situation. The task is the following: Turn on lights in room P121.

We identify a client in form of a Compaq iPAQ handheld device that has a microphone that accepts speech. The server has a mapping from the iPAQ to a lightswitch panel that controls all the lights in a given building.

The command Turn on lights in room P121 will be decomposed by the application running on the client to the following components:

- lights is intended for the light switch panel actuator
- Turn on is a change of status of a given light switch
- room P121 is the location of the switch
Command Requests

A command request is data passed from the client to the server. The server has to know the command request to evaluate it and pass it to the correct actuator, all other command requests will be dropped. A typical command request from the explained setting would look like this:

```xml
<?xml version="1.0"?>
<Request version="1.0" id="4711" timestamp="2003-02-04 19:12:42">
  <Type>Command</Type>
  <Source>Ferscha iPAQ3</Source>
  <Destination>Light Switch Panel</Destination>
  <Owner>ferscha</Owner>
  <ExecutionDateTime />
  <Deadline />
  <Content>
    <Parsed>Turn on</Parsed>
    <Parameter name="room" value="string">P121</Parameter>
  </Content>
</Request>
```

Look that the base form of the sentence is split up into its major components for easier evaluation.

Control Requests

Control requests are XML representations of data going from the server to an actuator. In most cases the server rebuilds the received command request to a control request based on its stored mapping table and rewriting rules. Also the server decides to which actuator the data is passed. For our light switch panel example the control request would look like this:

```xml
<?xml version="1.0"?>
<Request version="1.0" id="4711" timestamp="2003-02-04 19:12:44">
  <Type>Control</Type>
  <Source>Ferscha iPAQ3</Source>
  <Destination>Light Switch Panel</Destination>
  <Owner>ferscha</Owner>
  <ExecutionDateTime />
</Request>
```
In this case the actuator does not accept commands of the form Turn on, as the client issued, but the server rewrites the given command using its stored rewriting rules. Note that now the control request looks like a RPC call.

**General Requests**

A general request is for instance a setup request when a client or actuator logs onto the VRIO namespace. In our given example a typical general request would look like this:

```xml
<?xml version="1.0"?>
<Request version="1.0" id="4711" timestamp="2003-02-04 19:12:44">
  <Type>Setup</Type>
  <Source>Ferscha iPAQ3</Source>
  <Destination>VRIO Server</Destination>
  <Owner>ferscha</Owner>
  <ExecutionDateTime />
  <Deadline />
  <Content>
    <Parsed />
  </Content>
</Request>
```

An additional general request would be an acknowledge request, when an actuator sends back a notification that he has received the appropriate control request.

### 2.2.2 The C++ API

The following diagram shows (shown in figure 2.5) the collaboration diagram of the VRIO Request class.
The main class for interacting with VRIO request messages is the VRIORequest class. The application developer builds up his own VRIO message via set-methods in the VRIORequest class.

The Source, Destination, Owner and Content members of the VRIORequest class have only get_* methods. The underlying C++ objects provide data access for the major tags that have to be filled out or read by client or actuator. To access the Parameter members the application developer uses a ParameterList object that stores all possible parameters in a list.

After all mandatory tags are set (if they are mandatory for the XML message they will appear anyway) the developer creates the XML message via VRIORequest->xml_cpy().

**Communication Layer**

To use the communication layer of the VRIO framework, the application developer only has to set some specific values (like host, port), issues a setup method and the communication parameters are set. To send and recieve messages the developer uses the VRIOSend and VRIORecieve classes and passes the message he wants to send to the framework via parameter. There are some return codes that can be evaluated due to error handling.
Using a VRIOCommReceive object the application developer has to choose if he puts the vrio_receive method into a thread, because as in all socket applications this command will block since a connection is established and terminated correctly.

After receiving a message from the communication layer the developer passes it to the VRIOParser->parse() method to get a correct VRIORequest object. If the XML document is valid and well-formed, the fields in the VRIORequest object are filled out correctly for further use.
2.3 The Embedded ViaVoice SDK

The Embedded ViaVoice SDK enables the development of speech-driven applications on handheld devices. Speech recognition, text-to-speech (synthesized output of textual data) and other audio functions are computed in real time on embedded devices such as the Compaq iPAQ.

2.3.1 Architecture

Embedded ViaVoice provides a component-based architecture. These components are compiled as part of an application to provide the APIs for deployment of speech technology applications. The API sets included with Embedded ViaVoice provide the following capabilities:

- Audio device management
- Speech recognition management
- Speech synthesis management
- Binary data management

The illustration in figure 2.6 on page 31 shows the relationship among the Embedded ViaVoice components, the abstraction of the audio device driver and hardware by the Embedded Audio Layer (EAL) of the audio device, and that the application links these components into its required image.

Embedded ViaVoice supports the following speech application components:

- Audio Optimizer (AOP)
- Embedded Audio Layer (EAL)
- Embedded Speech Recognition (ESR)
- Embedded speech synthesis (ECI)
- Unlimited vocabulary (VOCU)
Audio Optimizer (AOP)

The Audio Optimizer (AOP) component is an automatic gain control designed to increase speech recognition accuracy. It analyzes audio data it receives during a speech recognition session and adjusts the hardware input gain, as necessary. The operating parameters and adjustment levels are generally fixed. The AOP is described in Managing Audio.

The ECI and AOP components require the EAL component to function. Depending on the usage of the VOCU component, ECI is required for the unlimited mode (as opposed to the dynamic mode).

Embedded Audio Layer (EAL)

The Embedded Audio Layer (EAL) component provides an interface for audio recording and playback on embedded devices and shields an application from audio device driver details. The EAL is mandatory for speech synthesis (ECI).

Embedded Speech Recognition (ESR)

The Embedded Speech Recognition (ESR) component provides an API that allows applications to implement speech recognition commands.
Embedded Speech Synthesis (ECI)

The Embedded Speech Synthesis (ECI) component provides an API for speech synthesis on embedded devices. There exists language support for most common languages, like US English, UK English, German, French, Japanese, ... These language packs may be abbreviated by ecienus, ecienuk, ecidede.

Unlimited Vocabulary (VOCU)

The unlimited vocabulary (VOCU) component provides the capability to generate pronunciations using either static dictionaries or ECI to produce vocabulary sets at run time. The embedded application can use the ESR API with the new vocabulary set. The VOCU component may require ECI to be running on the embedded system.

2.3.2 Hardware and Software Requirements

Windows CE Memory Requirements

Table 2.1 on page 32 shows the component Flash, DRAM, and total estimated memory requirements for Windows CE on a typical handheld device.

<table>
<thead>
<tr>
<th>Component</th>
<th>Flash Memory</th>
<th>DRAM Memory</th>
<th>Total Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>esr</td>
<td>592 KB</td>
<td>477 KB</td>
<td>1069 KB</td>
</tr>
<tr>
<td>esr + eci + ecienus</td>
<td>2646 KB</td>
<td>750 KB</td>
<td>3396 KB</td>
</tr>
<tr>
<td>esr + vocd</td>
<td>1359 KB</td>
<td>747 KB</td>
<td>2106 KB</td>
</tr>
<tr>
<td>esr + vocu + ecienus</td>
<td>3359 KB</td>
<td>917 KB</td>
<td>4276 KB</td>
</tr>
<tr>
<td>esr + eci + vocd + ecienus</td>
<td>3357 KB</td>
<td>1018 KB</td>
<td>4375 KB</td>
</tr>
<tr>
<td>esr + eci + vocu + ecienus + bfm</td>
<td>3363 KB</td>
<td>1102 KB</td>
<td>4465 KB</td>
</tr>
<tr>
<td>esr + acbf</td>
<td>611 KB</td>
<td>522 KB</td>
<td>1133 KB</td>
</tr>
</tbody>
</table>

Table 2.1: Memory Requirements I [EVV01]

Table 2.2 on page 33 shows the ECI shared library Flash, DRAM, and total memory requirements for Windows CE on a typical handheld device for US English and German language.

Codecs

Table 2.3 on page 33 shows the minimum audio requirements that a user-supplied codec must meet.
## 2 Architecture

<table>
<thead>
<tr>
<th>ECI Shared Library</th>
<th>Flash Memory</th>
<th>DRAM Memory</th>
<th>Total Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecienus</td>
<td>2033 KB</td>
<td>286 KB</td>
<td>2319 KB</td>
</tr>
<tr>
<td>ecidede</td>
<td>1445 KB</td>
<td>274 KB</td>
<td>1719 KB</td>
</tr>
</tbody>
</table>

Table 2.2: Memory Requirements II [EVV01]

<table>
<thead>
<tr>
<th>Signal Parameters</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling-to-Noise ratio (SNR)</td>
<td>100 Hz to Nyquist frequency is ~ 60 dB</td>
</tr>
<tr>
<td>Total Harmonic Distortion plus noise</td>
<td>3 dB full scale</td>
</tr>
<tr>
<td>Sample width</td>
<td>Minimum of 13 bits of usable signal are required for speech recognition recording, 16 bits are required for ECI</td>
</tr>
<tr>
<td>Sampling frequency (speech recognition)</td>
<td>11 kHz is required</td>
</tr>
<tr>
<td>Sampling frequency (ECI)</td>
<td>8 kHz or 11 kHz is required</td>
</tr>
<tr>
<td>Number of channels</td>
<td>A mono signal is required</td>
</tr>
<tr>
<td>Signal encoding</td>
<td>Signed linear PCM</td>
</tr>
<tr>
<td>Input gain steps</td>
<td>Minimum 10 steps</td>
</tr>
</tbody>
</table>

Table 2.3: Codec Audio Requirements [EVV01]

### 2.3.3 Design Guidelines

When designing an application that uses speech as its primary interface, it is important to design the speech UI of the application early in the design process. The UI should undergo usability testing to ensure that it is intuitive and that the voice commands can be recognized accurately by the speech recognition engine.

#### Active Contexts

When designing a speech user interface, the set of phrases a user can say at a given time is typically called the context (or application state). When a speech command is recognized, the application can either switch to a different context or stay in the current context. Designing an application involves determining which contexts the application should have, which commands are available within each context, and which commands transition the application from context to context.

A context is not directly related to and should not be confused with a single vocabulary. Instead, a context represents all the phrases available at the current application state. Therefore, a context can consist of one or more vocabularies.
When creating a context, one should ensure that the set of active phrases is intuitive to the user. This means that the selected phrases are easy for users to match up with the task currently being performed. Also, it is important to ensure that the speech engine can accurately recognize the selected phrases. Having a vocabulary with too many choices after a single word or having similar-sounding words can cause the recognition performance and accuracy to degrade. One way to meet these goals is to use an iterative design process, for example:

1. The UI designer designs an initial set of phrases for each of the contexts.
2. The set of phrases is used in mock-up scenarios with users to determine if the phrases meet the needs of the tasks being performed.
3. The set of phrases for each context is updated based on the new data acquired in Step 2.

Steps 1, 2, and 3 are repeated until the users find the phrases easy to use and the speech engine can easily recognize the phrases with sufficient accuracy.

**Design Trade-Offs**

The following trade-offs should be considered when designing grammars:

- **Speed and accuracy.** Larger grammars are more capable of command variations, but increase command latency and reduce recognition accuracy. For example, adding alternative commands, such as

  is there mail from <name>

  and

  did <name> send me mail.

- **User expectations.** Larger grammars can set unrealistic user expectations because a user can say just about anything. Without the possibility of real natural language processing, the speech recognition system cannot anticipate all the possible ways a user can structure a command. Therefore, it is important to carefully limit and define the commands that the user can say. To maintain real-time performance of the recognition system, a grammar should contain no more than 500 words.
Speaking to an Application

When speaking to a speech recognition application, it is important to follow a few basic guidelines:

- Always speak clearly and in a direct line to the microphone.
- Do not hesitate or mumble while speaking.

It is important to understand the specifications for the Signal-To-Noise Ratio (SNR) with respect to the microphone and the acoustic engine data provided to the engine. Reducing noise and/or matching the noise aspects of the engine data improve recognition accuracy.

Speech Synthesis During Recognition

When developing a robust speech application that includes speech synthesis and speech recognition, it is important to design the application so that speech synthesis does not occur while speech recognition is in progress. To prevent this from happening, ensure that all speech synthesis output has completed or has been paused before recording audio. In addition, it can be verified that the application is in a non-speaking state.

2.3.4 Decisions According to VRIO Design

The decisions made how to use the Embedded ViaVoice SDK in the VRIO context were the following:

- building an application with a few states
- building basic static vocabularies for the major states
- using dynamic vocabularies that can be changed at runtime
- showing proof of concept with small static vocabularies and a dynamic vocabulary built during run-time

2.4 Using the Embedded ViaVoice SDK Application Programmer’s Interface

To enable spoken input in an application the developer has to divide the commands the application must understand into static and run-time vocabularies.
2.4.1 Static Vocabularies

Static Vocabularies are a set of commands that never change in the application, like End to end the application. Of course there can be more than one spoken command for the action the application shall perform, like End, Quit, Bye. To create such static vocabularies the developer first writes grammar files and builds the vocabulary sets out of the files.

Grammar Files

The Speech Recognition Command Language (SRCL) is a file format that Embedded Via-Voice uses to create grammars that define vocabularies. A grammar is a set of allowable phrases that a speech recognition engine can accept, where a phrase consists of one or more words. The following example shows a grammar that accepts a four-digit object number:

"Objekt 0 0 0 0"
"Objekt 0 0 0 1"
"Objekt 0 0 0 2"
....
"Objekt 9 9 9 9"

Using SCRL, this grammar can be reproduced much more concisely and in a more maintainable format. In SCRL, the same four-digit object number would be defined as:

<root> = Objekt <ObjectID>
:ObjectID> = <digit> <digit> <digit> <digit>
<digit> = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

The format is based upon the Backus-Naur Form (BNF) specifically adapted to speech recognition.

Building Static Vocabularies

After an application writer has designed the application grammars and written the corresponding files to represent the grammars, a vocabulary set must be built from these files. A vocabulary set is a combined grouping of the binary representations of grammars in a format usable by the speech engine.

The application writer specifies the set of files that represent the grammars, a set of pool files that define additional words contained in the vocabulary, and a specification of the acoustic model that the vocabulary sets can use. These acoustic models must match the
engine initialization data that the application passes to the speech engine. The illustration in figure 2.7 on page 37 shows this process in detail.

Figure 2.7: Building Vocabularies

These created vocabulary sets are referred to as static because the grammars they represent cannot be changed at run time on the embedded device.

The following procedure is required to create a static vocabulary set:

1. Write one or more grammar files that define the vocabulary grammars required by an application. There should be one grammar file for each vocabulary.

2. Write one or more files containing the baseform combinations for words used in a vocabulary that are not contained in the default dictionary.

3. Add these words to a special pool file (if required).

4. Create the binary vocabulary set files from the grammar files, the pool file (if created).

5. Create a binary image consisting of all vocabulary sets used by the application and send this to the target.

VRIO Scenario2 Grammars

The main grammar file used in the scenario 2 setting looks like this:

```
<root>= Erzeuge:50001
   | Objekt:50002 <ObjectID>
   | Befehle:70000
   | Kürzel aktualisieren:70001
```
The <root> node is the main entry for the grammar. As one can see there are several commands possible, Erzeuge, Objekt <ObjectID>, Befehle, Kürzel aktualisieren and Ende. The annotation info for the command Objekt contains four digits. After the application recognizes the command (in fact the numeric ID of the command) it is able to switch between application states, enabling other vocabularies.

### 2.4.2 Creating Run-Time (Dynamic) Vocabularies

One limitation of vocabularies is that the grammars they represent cannot be modified after they are built on a host development system and stored into memory on the embedded device. This limits the function of applications because many applications require vocabularies to change at run time. An example of such an application is the one presented in chapter 4 on page 47 where a user might want to switch between different shortcut lists. Therefore, it is not be practical for an application to store all possible shortcuts into a static vocabulary.

**External Lists**

One way an application can overcome this limitation is to use an external list. An external list provides a way to add words to a vocabulary at run time instead of requiring a static list of names that are statically linked. An external list is a placeholder that a developer defines in a grammar file where words are added to a vocabulary at run time.

**Acoustic Baseform Service**

The acoustic baseform service converts speech in the form of PCM data into baseforms. An application can then add the created baseforms to extend a vocabulary at run time. The acoustic baseform service functions in a manner similar to the recognition service in that it takes as its input PCM data. However, instead of converting the PCM data into text (as the recognition service does), the acoustic baseform service converts the PCM data into baseforms.
The acoustic baseform service operates in Push-To-Talk mode only, regardless of how the engine is configured. Therefore, the application must know when the user indicates that speech has terminated.

**Dynamic / Unlimited Vocabularies**

A vocabulary generated by an application at run time is called a dynamic vocabulary. To use dynamic vocabularies the grammar must be stored in the form of memory buffers that define the vocabularies in the dynamic vocabulary set and a collection of application-specific pool files.

An unlimited vocabulary is an extension of a dynamic vocabulary in that it is not restricted to a fixed set of words. The process of creating an unlimited vocabulary uses the speech synthesis engine internally to generate pronunciations for words that are not in the dictionary or pool files.

An application creates unlimited vocabularies in the same way as dynamic vocabularies. There is a performance overhead with respect to generating unlimited vocabularies versus dynamic vocabularies. The speech synthesis engine (which is quite sizable) must be available on the embedded device. Also, generating unlimited vocabularies takes longer than generating dynamic vocabularies.

### 2.4.3 Speech Recognition and Synthesis

**Processing Speech**

Processing speech is accomplished by performing the followings tasks:

1. Initialize the speech engine by selecting the acoustic environment and engine operating parameters.

2. Initialize one or both of the speech engine services: the recognition service to convert PCM data into phrases for speech recognition, and/or the acoustic baseform service to convert PCM data into baseforms to extend vocabularies.

3. Register one or more vocabulary sets that speech applications use to perform speech recognition.

4. Register and enable one or more vocabularies contained in the registered vocabulary sets.

5. Start the speech engine so that the speech engine can generate recognition results.
6. Acquire and enqueue PCM data.

The application uses vocabulary binary data (created from grammar files) and the language specific binary file for dynamic vocabularies as well as audio data as input. This process is shown in figure 2.8 on page 40.

![Diagram of input files for speech driven applications]

Figure 2.8: Input Files for Speech Driven Applications

**Synthesizing Speech**

The ECI API library provides an interface between applications and the same speech synthesis technology employed in IBM’s ViaVoice Outloud text-to-speech (TTS) system. This embedded version of ECI has been redesigned to provide support for multiple, concurrent speech synthesis threads, and a consistent interface on all supported platforms.

Text to be synchronized is appended to the input buffer. Each word takes its voice definition from the active voice. Speech is synthesized from the input buffer according to the associated voice parameters, placed in the output audio buffer, and then sent to the appropriate destination. The active voice can be set from a number of built-in voices. The language, dialect, and voice parameters can be modified individually. As text is added to the input buffer, the active voice definition is stored with it so that changes to the active voice do not affect text already in the input buffer.
3 Scenario 1

The first scenario of VRIO’s application is placed in the CAVE Automatic Virtual Environment [CNSD+92]. The target application is Holodeck, a simple 3-dimensional editor for arbitrary graphical objects. With Holodeck, the user creates a 3D scene by placing arbitrary objects within the virtual world. User interaction is required for placing and manipulating the objects.

3.1 Structure of the Virtual Reality environment

3.1.1 Hardware Components of the Holodeck Application

The Holodeck Application consists of three parts:

- The CAVE Automatic Virtual Environment
- The 3D Wand Input Device
- The VRIO Virtual Reality Input Output system

The CAVE is a room-sized, 3-D projection-based environment for real-time visualization of interactive, synthetic computer-generated contents. The display environment consists of 3 projection walls (2.5 x 2 meters) and a projection floor. The graphics contents of the wall are generated on the SGI Origin 3800 supercomputer, which allows to process even large amounts of data in reasonable time.

The application of the CAVE requires stereo shutter-glasses, which provide the user with a 3-dimensional view of the scenario visualization in the CAVE. The correct perspective according to the users position in the CAVE is computed based on the tracking information of the shutter-glasses. This is expected to increase the immersion of the user and thus the experience of being “inside” the virtual world.

The primary device for navigation and interaction in the CAVE is the 3-D wand, a 6 degrees-of-freedom pointing device. In our case, the wand is a 3 button track-ball device.
equipped with a tracker, which provides the position of the tracker within the the simulated scenario.

While the wand is a suitable device for positioning and navigation in the CAVE, its 3 buttons limit the flexibility in terms of control mechanisms. In the past, most Virtual Reality application required additional input from a keyboard, which was placed on a table close to the CAVE.

The VRIO system is expected to relieve the user from moving between the Virtual World and the keyboard. Instead, commands can be spoken into the microphone, and are translated by the VRIO server in the background into corresponding actions regarding the Virtual Reality application. Different commands for modification of the virtual objects are entered via voice, the movement and positioning of the different geometries is controlled by the wand. Those, the only input devices the user needs anymore, are used closely coupled.

For the application of VRIO in the CAVE, the original VRIO prototype 0 [KRV01b] has been replaced by the latest version as described in this document. This comprises the Compaq iPAQ, which features a head-set with a microphone, and a Wireless-LAN card to connect to the network and the VRIO server respectively. For simplicity, the server is running on the SGI Origin 3800 supercomputer, although any other available machine could be used.

Figure 3.1 displays a user in the CAVE equipped with the shutter-glasses, the head-set, the wand in the left hand, and the Compaq iPAQ in the right hand.

### 3.2 Operation of the Holodeck Application

#### 3.2.1 Initialization

The Holodeck application is only one CAVE application, that can be controlled by VRIO. Other examples are described in [KRV01a, KRHV01, KH01b, KH01a, RKV01, KRV00]. However, since it is our most recent work we decided to take it as an example for the usefulness of speech processing within Virtual Reality.

Upon initialization, the Holodeck actuator connects to the VRIO server with the provided port number and IP address. A empty Holodeck world is generated, waiting for further commands from the user. Additionally, the VRIO client on the Compaq iPAQ connects to the VRIO server and initializes the audio capturing software.

#### 3.2.2 Activities within the Virtual World

Within the VR world, the user is able to perform the following activities:

- Manipulation of 3D Objects
Figure 3.1: User Wearing Shutter Glasses
3 Scenario 1

- File operations of 3D Scenes

Manipulation operations of 3D objects are used to compose the virtual scene. The user is able to generate a series of different 3-dimensional objects or additional items such as light sources by using the command

```plaintext
new <object>
```

followed by the particular `<object>`. The following objects are available within the Holodeck application:

```plaintext
cone|cube|cylinder|pyramid|sphere|light
```

The object `light` represents a light source, which can be used to illuminate the objects in the current scene.

3.2.3 Object Modifications

Once an object has been generated with `new`, its position is determined by the position of the wand in the CAVE. As long as an object is attached to the wand, the user is able to move its position to arbitrary locations in the cave. With the command

```plaintext
stop
```

the position of an object can be fixed to the current location. With

```plaintext
move
```

the object can be re-positioned again. Once an object’s position has been fixed, new objects can be generated using `new`. To select another object already available in the virtual world, the user enters the command:

```plaintext
select [previous|next]
```

The next or previous object in the order of creation will be selected. The objects are ordered round robin, such that the first object follows the last in the queue. Objects that have been selected can be modified in terms of shape and appearance. With

```plaintext
rotate
```

an object can be rotated according to the movement of the wand. With
scale [up|down]

the size of the object can be adapted as required. With the command

color <Color>

an object’s color <Color> can be modified. The following colors are available:

red|green|blue|yellow|magenta|cyan|white|black

In addition, the three components of each color (red, green, and blue) can be modified with the corresponding commands and the requested color code:

[no|less|more|full] [red|green|blue]

3.2.4 Example of Holodeck Application

An example of the user inside the CAVE is shown in Figure 3.2. A set of objects has already been generated and is displayed on the right wall. The current scene includes a white cube with a red cone on top of it, a green pyramid to its left and a blue cylinder in the background. Please notice, that the 3-dimensional imagination is lost due to the 2-dimensional projection of the picture.
Figure 3.2: User in the CAVE - Holodeck Scenario
4 Scenario 2 - Multiuser Interaction via VoiceWalls

This scenario describes the use of the VRIO framework in conjunction with a coordination framework that enables multi-user interaction on large displays (walls), the WebWall framework. The combination of the two frameworks results in interactive walls that can be controlled using voice commands by multiple speakers.

4.1 WebWall: Multi-User Interaction on Wall Displays

In the past, computer screens have dominated our office desks. Over the last years, digital displays have moved away from the tables to our office walls in the form of digital whiteboards, and out into our everyday environments: Video walls can be found in almost every city by now, they appear in city centers, shopping malls, sports arenas, train stations and other places that are highly frequented [Put02]. Due to their different physical properties, other interaction paradigms than those associated with the classical computing device, the desktop PC, have to be employed.

4.1.1 Wall Computing

While PCs are private, or at least used by one person at a time, wall displays are shared among a number of users. Not only can many people perceive the displayed information, many can interact simultaneously on the same screen. These displays are constantly present can may react to input from various devices and sensor channels to provide proactive, context-based behavior.

The related research area deals with effective interaction with these shared artifacts that enable communication between people and interaction with digital components. Wall computing systems provide a mainly visual model of interaction, but unlike traditional computer science, when talking about shared displays, we refer not only to classical CRT displays,
4 Scenario 2 - Multiuser Interaction via VoiceWalls

but to wall-sized, high-resolution displays, ambient displays, augmented reality displays and other systems that can render information on different physical surfaces in the environment like walls, ceilings, floors, tables, or even people [YNM+02]. Wall computing is thus not restricted to an existing technology, but is an interaction metaphor or coordination model:

Wall Computing coordinates the interaction between users, objects and shared display artifacts.

Fig. 4.1 shows some examples of walls, one in a public setting at Zurich station, and one in a the private area of an office. The presents the digital pendant to normal whiteboards, and has been an active area of research over the last ten years. Closed-user group walls, like digital whiteboards, have a long research history (starting in 1992 with the Xerox Parc Liveboard [EBG+92]). Interaction on public artifacts, on the other side, is a rather new field of research.

Traditional wall communication, like sticking posters to poles or kiosks, imposes severe limitations on the interactivity of the communication process as it involves a passive medium for transporting information. Typical problems that may occur with paper-based public communication is that often some notes are hidden by others, and many are already out of date when they are read. Furthermore, interaction is a multi-step task involving writing down the address or number to call and the actual process of getting into contact.

Besides the stated problems, wall communication can be characterized by the following properties:

Public access  Walls are a public medium as the information posted on them can be read by anyone, and any person can publish information on them. Some exceptions exist, of course, where permission needs to be acquired first.
Being public, posted items address many people at once. Any person in reading distance can view its contents and eventually contact one of the authors, if an item sparks their interest. This contrasts private communication where data is exchanged via one to one connections, like phone calls, or is kept in small, well-defined groups.

**Anonymity**  Items published on public surfaces do not expose the identity of the author a-priori. This enables people to post information without publishing private information. Publishers do need to do so when they explicitly wish interaction. In that case, they have to provide some kind of contact information on the note, like a phone number or mail address.

When publishing information items in public spaces, the publisher does not know the readers in advance, in general. This can be limited so smaller groups, in some cases, when only certain people have access to information that is present in one place, or by watching the people passing by for example, but these groups are more or less open and not to determine a-priori.

**Asynchronous**  Synchronous communication describes a way of data exchange where both communicating parties must be ready to participate at the same time for interaction. Posting notes in public places, is asynchronous by nature: After pinning a note somewhere, eventually people will go by and read it.

The time it could take between the publishing and an eventual use of the information depends on the amount of readers that access this item and can range from minutes to weeks. As paper is static, the information stays available until the note is removed or hidden by another item.

### 4.1.2 Limitations of Public Communication

Traditional wall communication uses static infrastructure and objects for transporting information. The problems associated with the medium used and the usage patterns involved in the communication process results in a set of problem areas that cannot be solved easily without digital support:

**Context awareness**  Traditional notice boards involve several tasks to be accomplished until an item can be published: It has to be written down or printed, the author has to physically move to the wall where the information shall appear and pin it to that surface.

This physical movement is also involved when the posted item is not needed any more: The author has to go there and remove the note from the board again. Therefore, most (if not all) items will stay on a wall, even if they are not needed any more for a long time.
physical ‘garbage collector’ needs to take care of these left-overs. The whole process seems therefore not only inconvenient, but involves also a cost-factor on the administration side.

**Occlusion** New notes tend to hide older ones, making them gradually disappear. This way, important information might be hidden, and eventually lost under newer notes. While this is presents a crude form of ‘garbage collection’, as old items vanish, there is no control over the process available, as it is uncertain where the next poster will pin his paper and eventually hide important information.

**Topicality** Keeping the information up-to-date that is on display is the main problem of paper-based notice boards. Especially data that depends on dynamic information, like event times, sale prices, booking information and the like, cannot be updated easily. As timed-out information keeps being posted, it uses up space and distracts from newer information.

### 4.1.3 Public Communication on Digital Walls

Communication using a digital medium should support the properties of traditional public communication while eliminating its limitations. We have identified a number of requirements that a framework needs to fulfill in order to support this casual type of interaction, dealing with managing output, managing input, and supporting different types of interaction.

**Managing Screen Resources** Managing the available space of normal walls is an ad-hoc task that is executed by every author when a note is posted. Most of the problems reported above have their roots in this distributed, uncoordinated actions. A digital wall should provide a better solution to overcome the visibility problems we have described.

**Rendering & Layout** Like their analog counterparts, digital walls should be able to display a differently sized items, and support different media types and formats. To avoid occlusion problems like on the analog boards, an automatic layout mechanism is needed to arrange items in a way so that no important information is hidden.

**Information Overload & Eviction Policy** When posting an item on a board, the intention is normally to buy, sell or inform about perceive something until the item has been found, sold or the event is over. After that time, these items are not relevant for display any more and should be removed to avoid cluttering the display area. To enable this behavior, we propose a lifetime for every object. After their time to live has expired, an element should be automatically removed from a wall.
At times, there may be more items waiting to be posted than can be rendered at once without occluding other items. When such a condition occurs, a queueing mechanism shall provide a temporal order for waiting objects, which are displayed when older ones expire and are removed from the wall.

A visualization of this queue can acts as a visual feedback for users. Every action should result in a reaction a user is able to perceive and understand. In this case, a representation of the size of the queue could give an estimate when the next item will be displayed. That way, potential posters may estimate the ‘load’ on a wall and move to another if needed.

**Enabling wall access**  Granting access to public walls involves three different aspects of tasks. First, the system needs to cope with different types of input devices and technologies. Users should be able to interact spontaneously, resulting in specific requirements to the user management which shall enable the ad-hoc creation of new users. Finally, access to the wall needs to be regulated in a way to cope with offensive content or users.

**Flexible Access**  Walls should be easy to access with whatever access technology or access device is available to a user. While reading is easy - each item has a visual representation, users may interact with objects in different ways. This leads to the necessity of supporting different access technologies, as well.

Accessing information on a wall should be easy and intuitive, adapted to the personal tools a user is used to work with. It should not be restricted to the input modalities provided by a specific wall display.

**User Management**  Notice boards and walls are in general publicly accessible, which should be equally true for their digital pendants. With the enhanced possibilities of interactivity, flexible user management becomes a vital issue. To reflect the natural access to paper-based boards, interaction with digital walls should not need a setup or registration step, but should be immediately possible at first contact.

Another issue is how to define access rules for different situations, to restrict who is allowed to remove items, for example. In any case, users have to be identified, logged, and billed, in case of commercial systems. Furthermore, we have found several issues dealing with jurisdiction in conjunction with public displays that demand us to log who posted which item.

**Content Control**  Taking into account the sensitive situation of creating a public communication medium, access needs to be controlled in a way. Posting offensive content can be
at least annoying but can also mean legal consequences for the providers of a wall.

**Fulfilling Different Communication Needs: Services**  We have identified different intentions of people publishing information - informational items like notes, ads, poems, and others. Like applications in the traditional desktop computing environment, we map these different communication needs to service classes.

Service classes regulate the creation of and interaction with such items. Services implement the applications that fulfill different coordination and interaction functions.

In the following, service class addresses the properties of a whole service, while the term service instance refers to an instance of a service class, an item that is visible on a wall.

### 4.1.4 The WebWall Framework

The WebWall framework enables interaction by means of various service classes that offer different modes of interaction, comparable to applications on desktop computers. It fulfills the requirements for public wall communication defined in the previous subsection to provide controlled access to shared display spaces using mobile devices while guaranteeing a maximum of flexibility and extensibility.

In this section, we present the system at module level for a general impression of how the processing is partitioned. The WebWall framework can be seen as separated into a number of layers with defined communication channels between the entities in these layers. Figure 4.2 provides an overview over the framework modules that are relevant for integrating it with the VRIO framework. A detailed description of the implementation of the system have already been published in several theses and conference papers [FKV02, FV02, Vog02].

The system architecture of the framework is organized along two major concepts:

- **The separation of creation from the use of information.** The request generator and covers the creation side, where different access technologies are used to access the system. Requests are processed in the backend and the output transformed and displayed on a wall display using the display engine.

- **The separation of access and display technologies from the core processing layers.** The access technologies as well as the wall display make up the technology-dependent parts of the system (lower half). The processing itself is independent of access and display technology (upper half).
Figure 4.2: WebWall framework architecture.
Interacting with a WebWall

The Request Generator listens for input device dependent request data from the access modules, parses the incoming data and uses any technology-dependent identification data to authenticate his request the user. Service-specific code handles the content of the request to create a well-defined, device-independent request document that is forwarded to the backend for further processing.

In the original WebWall system, requests could only be created using cell phones via SMS, by sending an eMail, or via a Web interface. The combination with the VRIO framework opens a new dimension of interactivity, as items can be created and interacted with spontaneously using voice commands (cf. Fig. 4.3). This eliminates the tedious and time-consuming task of typing on mobile phone keypads and enables mobile users to spontaneously interact with walls.

Backend properties

The Backend System accepts the requests generated by the Request Generator and performs the requested action on the addressed WebWall, if the user is allowed to do so. Any changes such a request might cause (creation of new instances, deleting existing ones or interacting with running instances) are forwarded to the render engine.

The backend manages all active WebWalls along with all related objects (service instances) and user data. It incorporates the following mechanisms:

- A layout mechanism that is responsible for automatically positioning objects so others are not occluded. Users have no direct (physical) influence on the WebWall, so a proper
Scenario 2 - Multiuser Interaction via VoiceWalls

layout strategy is needed for maximising information on the display while avoiding clutter.

- A priority-queue for incoming requests. It enables ordered display of incoming requests and using a set of priority levels, important objects may override unimportant information on the display. In cases of overload, items are queued for later processing. This queue can also be visualized, so users have a good impression of current system load.

- An aging mechanism automatically removes old items. Each service class has a pre-defined maximum time-to-life, after which the items are removed from the WebWall. This reduces surface clutter and keeps the contents up to date, as old items do not need to be removed by hand.

- A user management module and user access control system regulates access to the instances. Every object on a WebWall has an explicit owner who may manipulate it, while other users have only restricted access.

Services

The applications that can be used on WebWalls are implemented as 'Service Classes'. Each service class enables a certain type of interaction or communication on a wall, like posting notes, or photo galleries, viewing videos, performing auctions, polls etc. New services can integrated into the system easily by subclassing a common service superclass and a dynamic loading mechanism.

Rendering Engine

The rendering engine is responsible for translating the (abstract) information about service instances into displayable content for a given display client, the physical WebWall. A collection of rendering modules adapt transform the content for different representations, like WebWall clients, the WWW interface, or a mail gateway.

Web Interface and Shortcuts

Users can access WebWalls also via a Web interface and interact with service instances via a normal browser. Besides this interaction path, the web interface serves as an administrative gateway, and a communication plattform for users.

In addition to these functions, users can save templates of service instances, shortcuts, that can be recalled at any time. Shortcuts are used to set up complex services like galleries,
polls etc. where complex data needs to be entered, like the locations of the images that should be displayed.

4.2 Service Classes

We have picked two service classes as a demonstrational subset of functionality for voice interaction: Posting notes and browsing through image galleries, two functionalities that involve complex input tasks when used with the existing input devices.

4.2.1 Note

Note is a service class intended to display short notices on a WebWall, as a form of a 1:n communication. There it acts as a digital analogy of the well-known stickies or Post-It notes used in everyday life. Users can post short text messages on a WebWall simply by sending some text to the wall. As note is the default service for all WebWalls, a new instance will appear containing the text.

Notes enable anonymous communication: People may send a reply to a service instance. Instead of displaying the passed information in a new item, the contents is routed to the author of the note. This behavior offers a higher degree of anonymity than paper-based board communication: The author does not need to publish any personal contact information, like writing a phone number on the note. It is sufficient to know the number of the WebWall item to communicate.

Notes can be created easily by sending a text string to the WebWall. As Note is the default service class for WebWalls, it is not necessary to prepend the command to create a new instance (‘!note’), resulting in a note with the default settings of the WebWall to appear. When the command is specified, notes can be made repliable, their lifetime (in minutes) and their priority can be set.

Using different styles, people can customize notes to their personal preferences. To prevent improper use of CSS or HTML code, the system offers a number of pre-defined
4 Scenario 2 - Multiuser Interaction via VoiceWalls

styles the users may choose from. These styles have a distinct name that can be passed as an argument in the command string (see Fig. 4.4).

Interaction Grammar for Note Notes should be created by using the command ”Neue Notiz” which creates and selects a blank note. Now, the system should be in text entry mode, letting the user append arbitrary text to the selected instance until the command “Fertig” exits this mode.

In practice, the speech recognizer on the iPAQ has to deal with very limited computing resources, which makes free-form text entry virtually impossible - system response becomes too slow and error rates rise. Therefore, text input needs to be limited to a useful set of keywords and sentences.

For the scenario, we have picked the following situation: A person wants to leave a note for a friend, or wants to buy/sell an item (a motorbike, car or table), resulting a set of sentences that can be used as an input for the speech recognizer:

- Suche / Biete (Motorrad / Tisch / Auto)
- Notiz an (Simon / Michael)

4.2.2 Gallery

Gallery provides a sort of ‘personal multimedia memory’. It displays a slideshow of pictures on a WebWall, that runs through automatically for a given time. The slideshow can be stopped at any moment and navigation commands can be used (‘next’, ‘prev’) to browse through the contents.

Figure 4.5: Service class Gallery.

Galleries can be predefined over the web interface, as it is highly impractical to include all these parameters (the URLs of all images, for example) in an SMS. Users may apply individual timings to pictures, re-arrange or delete them or upload new ones over the web interface. We currently limit the gallery to contain only up to 12 pictures to prevent system overloading.
Running galleries use the time values defined for each picture to automatically change pictures. Using can be influenced to go stop the ‘stop’ command, the automatic forwarding can be turned off. The commands ‘first’ resp. ‘last’ can be used to jump to the beginning or end of the presentation, while ‘next’ and ‘prev’ (or its alias ‘previous’) make it possible to advance or go back in the gallery by one page.

An experimental setup creates galleries from emails on the fly: A mail filter extracts attached JPEG images from MIME-messages to create a gallery (timings are set to reasonable default values). The gallery data is saved as a shortcut, as if it was manually created, using a unique name (the timestamp) as the name. A subsequent request with the name of the newly-created shortcut displays the gallery on the addressed WebWall. As this leads to a growing number of shortcuts in the system, an aging and pruning mechanism for old galleries might be needed for large-scale use.

**Interaction Grammar for Gallery** For efficient interaction with galleries, the speech recognition system should be able to understand all commands supported by the gallery instance:

- *Erstes* to jump to the first picture of a gallery
- *Letztes* to navigate to the last picture
- *Vor* to go forward
- *Zurück* for viewing the picture before the current one
- *Bild* `<nummer>` (where `nummer` is in the range of 1-12) can be used to go directly to the requested image

Galleries can be created using the shortcut name under which the images have been saved. After a shortcut has been instantiated, it is automatically selected, so the voice commands can be used as soon as the gallery appears.

### 4.3 The VRIO—WebWall Interface

For this application there exists a special interface on the WebWall server. The program has to get more information from WebWall objects than the conventional interaction with the Webwall would provide, so the application communicates with the WebWall through HTTP requests sent to a special VRIO WebWall servlet. The servlet returns status information in the HTTP answer, such as
4. Scenario 2 - Multiuser Interaction via VoiceWalls

- Service Class
- Object Numbers
- User defined Shortcuts
- Error messages

Because the user has the possibility to control existing objects on a WebWall, the different vocabularies depending on the service class of the object must be enabled at run time. So the application requires the information

- does the object with the specified number exist
- if so, who is the owner of the object
- of which service class is it
- how long does the object exist on the WebWall

4.4 Interactive Walls

The application described here is a sample application showing both the benefits of the Embedded ViaVoice SDK and the VRIO framework. With this application the user can control objects on an arbitrary WebWall. This is done through spoken commands that the application understands.

The application only understands german language. This is no limit, there is the possibility to exchange the vocabulary sets and grammars without modifying the application. The Embedded ViaVoice SDK provides most languages like US English, UK English, German, French, Japanese and so on.

4.4.1 Why Do Walls Need Voice?

Looking at the wall metaphor and the behaviour of people interacting with public walls it comes out that in everyday life the usage of embedded device often poses a problem to the user. Using cell phones for interaction via GSM or using more powerful embedded devices with integrated web browser means using badly designed interfaces for quick input of data. With a device that uses voice as input users

- can speak freely
Scenario 2 - Multiuser Interaction via VoiceWalls

- have no need for pulling out the device (pocket, bag, briefcase)
- lose rarely time using the interface

4.4.2 State Diagram

First approach

The statechart diagram shown in figure 4.6 shows the first version of the application, enabling the user to control objects on a given WebWall.

In the first attempt of creating a statechart of the application there are only a few commands possible, some for creating notes, hardcoded shortcuts (galleries), some for removing objects and a command to end the application. Also the user has the possibility to get an audio information of in which state he resides at the moment.

On top of the image there is some classification of the states. There are initialization states, states for object selection and class type specific commands.

To initialize the application the user has to enter the numeric ID of the WebWall (or the person can use some predefined shortcuts, like 'rail station', 'airport', and so on). In the initial release of the application this is done hardcoded for testing purpose.

Final Version

The final version of the application is a little bit more user friendly. He can create objects in a more intuitive way. There is a new state in which new objects can be created, to avoid confusion. The final statechart diagram is shown in figure 4.7. It is oriented in a hierarchical way to show that going down the hierarchy the user gets more and more commands he can enter.

4.4.3 Commands in Detail

The application aims at users who want to control objects on a WebWall, like creating notes, switching to other pictures on a gallery object or removing an object.

There are some basic commands the user always can enter (or at least in most of the states) and some commands that are bound to the class type the selected object belongs to.

Basic Commands

First, the user has the choice between two ways of working with objects, creating a new object or selecting an existing one. To create a new object the user issues the command
Scenario 2 - Multiuser Interaction via VoiceWalls

Erzeuge

To select an existing object the user has to enter the command

Objekt <ObjID>

where <ObjID> is the four digit object number of the WebWall object the user wants to manipulate.

Of course creating new or selecting other existing objects can be done out of any state the user is in. To do so, the person only has to issue Erzeuge or Objekt <ObjID>.

After creating a new or selecting an existing object there are a few commands that the user always can enter. With the command

Ende

the application is stopped without changing the object. To remove the active (or selected) object the user has to enter the command

61
After removing an object only new objects can be created or existing objects can be selected (as in the state when the user starts the application).

At last there is always the possibility to get the information which commands the user can enter at the moment. To get a spoken representation of the commands the user just issues the command

Befehle

There is a command, named

Kürzel aktualisieren

which rereads the shortcut list stored on the WebWall server, builds up a new vocabulary set for the shortcuts (this is dynamically created) and enables it. After issuing this command the user can use his own shortcuts to create objects on the WebWall. So it is possible to add shortcuts (on the WebWall server via the web interface) during runtime of the application. The command Kürzel aktualisieren is for situations when the user changes (enabling the new users shortcuts) or when the user adds a new shortcut to his list.
Object Creation Commands

The application initializes itself with a list of shortcuts the user has stored on the WebWall server. This has to be done because the creation of gallery objects is a more complex exercise. For gallery objects each of the pictures that should be visualized on the WebWall have to be named and every picture has additional flags to set (like the visualization time). To ease this process the user can redefine such picture sets and store these definitions under a common name (called shortcut). Creating a shortcut object means creating the defined picture set on a given WebWall.

Two different service classes of objects can be created with the application, notes and galleries. Getting a new note object means issuing the command

Neue Notiz

To create gallery objects the user has to enter the command

Neue[n]er <shortcutID>

where <shortcutID> should be filled with a correct shortcut the user has stored on the WebWall server. Of course the application has an internal list of these shortcuts and can synthesize them as well as putting a corresponding message on the handheld devices screen.

Class Type Specific Commands - Note

After a note is created on the WebWall the user can append text to it by simply speaking text into the handheld device, sentence after sentence.

In fact, the note object is a perfect example for using an unlimited vocabulary set, but this causes many problems in categories like memory usage, calculation time to recognize words and recognition ratio.

Class Type Specific Commands - Gallery

Like gallery objects are sets of pictures there are a few commands to control the visualization flow of these sets.

Erstes
Letztes

jumps to the first respectively last image of the set. With the Commands

Vor
Zurück
the user goes forward and backward in the picture set. Finally, he can go directly to a specific picture number by issuing the command

\texttt{Bild <picID>}

**Class Type Specific Commands - Other**

Including new WebWall services classes is no problem because the application is very easy extensible with new grammars, vocabulary sets and so on. The application developer only has to introduce new states in which the new commands are active. With this it is possible to build a new service class like \textit{chess board} where two or more users interactively can enter commands like \texttt{e2 - e4}.

4.4.4 Command Input Modes

There are two ways of speaking commands and recognizing them via the application. The first mode is Push-To-Talk (P2T), where the user must tell the program when a command is spoken (begin and end of audio recording sequence). This can be done either via pressing a button on the physical device or via mouse click on the user interface. This mode is an approach to userfriendliness because people can associate this mode with a walkie-talkie like process, where one has to press a button during the speech.

The second mode is Push-To-Activate (P2A). Here the application loops an audio recording sequence (given by a special length) and tries to recognize the spoken commands. This mode is more user friendly because the user only starts the application and enters commands via spoken input. But of course, it is highly resource consuming, because the application always records audio samples and always tries to recognize the given input, even if the user does not say a word for minutes.

4.4.5 Embedding the application in the VRIO framework

The application communicates with the WebWall through HTTP requests. It is possible to embed the application in the VRIO framework, giving it the meaning of a client and developing an actuator that communicates directly with the WebWall servlet. This setting provides the following advantage, namely that commands can be scheduled. For example, the command to create a note object at 19:00 with a given text can be issued at 13:00.

But there are also some disadvantages. For instance, the communication process slows down rapidly, because there are now three requests, from client to the VRIO server, from the server to the actuator and from actuator to the WebWall servlet. Because the client needs
information on WebWall objects the answer has to go the reverse way. This gives a total of 
6 single TCP transmissions instead of one bidirectional TCP flow. Second, there are more 
possible error sources, because now there are much more components involved.

4.5 Demonstration

This section shows some pictures taken out of the demo video “interactive walls” delivered on 
the CD. They show which interaction forms exist to communicate with an existing WebWall 
and how spoken commands perform action on the WebWall.

![Image of a phone with a message being typed.]

The user has to type the message he wants 
to send by hand. This is very exhausting 
because of the limited user interface (key-
board).

![Image of a phone with a message being sent via SMS.]

Sending the message via SMS gateway by 
typing in the number of the WebWall where 
the note object should be created.

Table 4.1: Interaction via SMS through GSM
The user types the message into a webform. This can be done on the iPAQ using the keyboard or via handstrokes on the display, a better interface than the SMS through GSM version.

Table 4.2: Interaction via iPAQ (Pocket Internet Explorer)
The user wants to create a note object and therefore issues the command *Neue Notiz* into his iPAQ.

The empty message appears. The application tells the user via TTS: *Neue Notiz, Objekt Nummer 0023*. The user issues the command *Notiz an Simon*.

The message appears, the text put on the note is spoken to the user via TTS. The user completes his message.

The note on the WebWall shows the full text the user has entered via speech interface.

Table 4.3: Interaction via Voice Commands

67
Two people meet in front of a WebWall, one of them wants to create a gallery object and issues the command *Neu Schiurlaub* (this is a shortcut user A has stored).

The gallery object appears on the WebWall and the user explains the content of the image to the other visitor.

User A issues the command *Bild 4* to go to a specific picture in the set of images.

The picture #4 is shown and the visitor asks some questions on the details of the picture.
While talking user B wants to show user A some hintful slides, so he issues the command *Neu Vorlesung*.

The newly created gallery object appears on the WebWall.

User B explains some details on the picture shown on the WebWall.

Table 4.5: Interaction via Voice Commands, part 3
Being at slide number 3, user B wants to go forward in his slide set so he issues the command Vorwärts.

Slide number 4 appears. After explaining something on the content the user wants to jump directly to the last slide. He speaks the command Letztes.

The gallery object on the WebWall shows the last slide in the picture set.

Table 4.6: Interaction via Voice Commands, part 4
4 Scenario 2 - Multiuser Interaction via VoiceWalls

To end the application, user B issues the command *Ende*.

Table 4.7: Interaction via Voice Commands, part 5
5 Achieved Goals

This section provides a summary of the results this project has achieved. The

5.1 Software Goals

- **Genuine architecture**: The architecture of the framework has been designed to best fit the situation of mobile workers that need to control a (known) set of devices. The framework supports dynamic reconfiguration of actuator and client addresses to cope with components that move to different locations.

- **Platform-independent communication**: The communication infrastructure is based on standard Internet protocols, the data itself is encoded in XML structures. Using a textual description format enables platform-independent data exchange, a necessity for incorporating arbitrary mobile devices into the framework.

- **Device abstraction**: An abstract mapping mechanism has been implemented that routes client-generated commands to actuators. Actuators are the representation of a (physical) device, which react to a published set of operations. Different physical devices may support the same (named) operations, making them exchangeable.

- **Client abstraction**: The clients create input to the framework, which is translated into control messages in the server. Using a normed data structure for the (expressed in an XML-DTD that can be used for validating requests), different clients can be implemented and integrated into the framework at little development cost.

- **Scheduling**: Requests can be stored at the server for execution at a later time. This enables

- **Persistent architecture**: The server stores any information regarding clients, actuators and command mapping information in a persistent storage. This guarantees a higher value of fault tolerance in case the server fails. Persistent logs are kept for analysis of user-device interactions, which could be used to provide pro-active behaviour.
5.2 Application-oriented Goals

- **Hands-free device control**: The VRIO framework enables control of arbitrary devices.

- **Efficient 3D manipulation**: The GUP scenario shows how the VRIO framework can be efficiently be used to access and manipulate data sets available in immersive 3D environments.

- **Control of multimedia content**: VoiceWalls

5.3 Papers

A research paper describing the VRIO framework [DK03] has been accepted at the Human Computer Interaction International 2003 (HCII) Conference in Heraklion, Greece.
6 Summary and Future

6.1 Future Enhancements

Future directions of research and development of the VRIO framework can be divided into two major concerns: Future research projects and novel application areas for the VRIO framework.

6.1.1 Future Research

- **Peer-to-peer architecture**: Transforming the current architecture into a peer architecture, where every component contains server features can provide a better adaptation to problems often found in mobility: The need for spontaneous reconfiguration and the absence of a central server (due to lack of network connectivity). Ad-hoc network connections can be exploited to exchange information (due to the store-and-forward implementation of the server components).

- **Automatic grammar generation**: The current implementation keeps the application logic (system state) in the client along with the speech recognition. Moving the state logic into the server could provide more dynamics for the clients, as application logic could be changed at one place only (the server), instead of changing the code on all clients. On the downside, clients need to be able to dynamically generate and recognize vocabularies (maybe in even different languages) and provide the server with the needed information. The runtime-created structures need some time for setup, which needs to be compensated for by using caching strategies.

- **XML-DTDs to grammar translation**: An automatism for translating device descriptions into input grammar poses several challenges, like the representation and extraction of speech commands and state information (when is a command available?).

- **Context-based grammars**: The combination of the VRIO framework with additional context sensors could provide a better model of the state the user is in (his context).


This information can be used to anticipate a set of possible commands the user is likely to use.

- **Noisy environments**: Dealing with background noise is a problem not satisfactorily solved up to now. The diverse nature of noise sources, from engines, music to human speech, provide severe problems for signal processing algorithms. Providing good results despite even in noisy environments is needed in many work areas and provides a challenge which still requires much innovation.

- **Real-Time Execution Environment**: The current server architecture and data communication structures incorporate a field per request for the execution deadline. An evaluation of different scheduling strategies and appropriate error recovery attempts could provide a valuable extension to the system.

- **Multi-channel user input**: Gesture recognition and other input devices provide other interesting channels for user input besides voice commands. Integrating other input devices into the VRIO framework can provide an even more versatile framework for natural interaction.

### 6.1.2 Voice Applications

- **Complex vocabularies**: It is essential to support for complex vocabularies and states, for example in machine inspection and repair tasks. These tasks need both a custom vocabulary (adapted to the part names of an engine, for example), and automatic creation of interaction states (which part has been already inspected/removed/replaced, what to do next?).

- **Free-form text entry**: The limited computational resources available on mobile devices is a severe limitation. An alternative architecture that enables free-form speech recognition could provide a much more flexible approach. This could be accomplished by either changing the voice recognition software and client hardware, or by transparently forwarding the audio-stream to high-end recognition servers.

- **Voice control for everything**: The Pervasive Computing Labor is a perfect test area for diverse application scenarios of the VRIO framework. A broad range of different devices (video beamers, computers, a hi-fi-system, etc.) need to be controlled remotely, as the devices are out of reach, hidden in an engine room.

- **Voice controlled presentations**: Hands-free control of powerpoint presentations is an application that could become as wide-spread and comfortable as the omni-present
laserpointer. A limited set of possible commands, not unlike the VoiceWall galleries, guarantee robust recognition results.

- **Voice controlled websurfing**: Surfing the Web on Voicewalls is a challenging task. It requires link analysis of web pages and automatic generation of the grammar depending on link texts. Several heuristics are necessary to deal with the diverse nature of links and their descriptions in order to get a manageable set of commands.

- **Humanoid in 3D control**: Controlling a humanoid avatar in a 3D immersive environment is a complex task. Voice commands can provide an effective way to recall complex command sequences.

- **Augmented reality interaction**: Besides pure VR applications, voice control can be exploited in augmented reality settings, where digital information is combined with physical objects, using either see-through head-mounted displays.

### 6.1.3 Extensions to the Server

The following extensions are major enhancements to the server architecture.

- **More Complex Mappings**: The command-control mapping is intuitively a 1:1 match from one command to one control. This mapping could be enhanced to a m:n mapping, where multiple commands trigger multiple controls. For example, the command *it’s freezing cold* would trigger actuator 1 that closes the windows and actuator 2 that activates the heating.

- **Knowledgebased Action**: The server can store the ingoing commands (and the controls triggered by the command) in a separate database. After some time a typical user profile will emerge, because most of the users will use the system in a regular way with the same commands.

  Using this knowledge the server can act in a proactive way. Knowing that a user regularly gets home and issues the command *Turn on TV* the system can do this automatically if the sensor notices the user coming home the next day.

- **Service Discovery**: The client and the actuator have to know the physical address of the server (typically using its FQDN). It is a big enhancement if the server would provide a service discovery mechanism where the clients and actuators discover the (nearest) server.
This could be also used for servers that have reduced functionality (because they are integrated in a handheld device, even if the device represents the client and the actuator).

- **Context Based Services:**

  The server can use additional parameters from the client or information he can require based on the clients ID from other sources like GSM providers (to get the cell info where the user physically is). Using this information would enable such a setting: the command *Turn on Light* would switch on the light in the room where the user stands and not in all rooms the light switch panel can control.

- **Server Clustering:** It is possible that in a large setting more servers are needed. This new designed setting would provide load balancing, location based services (for example different language settings). This involves more servers that have the possibility to roam clients when they move.
A Glossary

ACM Association for Computing Machinery
AI Artificial Intelligence
API Application Programmer’s Interface
ARGE ARbeitsGEmeinschaft
ATT American Telephone & Telegraph Company
CAVE Cave Automated Virtual Environment
CHI Computer-Human-Interaction
CMU Carnegie Mellon University
CPU Central Processing Unit
CRT Cathod Ray Tube
CSS Cascaded Style sheet
DBMS DataBase Management System
DRAM Dynamic RAM
DTD Document Type Definition
GNU Gnu’s Not Unix
GPS Global Positioning System
GSM Global System for Mobile Communications
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
IEEE Institute of Electrical and Electronics Engineers
MIT Massachusetts Institute of Technology
OCG Oesterreichische Computer Gesellschaft
OS Operating System
PCM Pulse-Coded Modulation
PDA Personal Digital Assistant
PHP PHP: Hypertext Preprocessor
PUC Personal Universal Controller
RPC Remote Procedure Call
SDK Software Developmenter’s Kit
SMS Short Message Service
SQL Simple Query Language
SRCL Language
TTS Text-To-Speech
UI User Interface
URI Universal Resource Identifier
URL Universal Resource Location
VR Virtual Reality
VRO Virtual Reality Input Output
WLAN Wireless LAN
WWW World Wide Web
XML eXtensible Markup Language
List of Figures

1.1 Effects of Moore’s Law. Graph courtesy Thad Starner [Sta02b] . . . . . . . 10
1.2 Nomadic Radio hardware setup [SS00] . . . . . . . . . . . . . . . . . . . . . . 14
1.3 Impromptu mobile user interface [Lee01] . . . . . . . . . . . . . . . . . . . . . 15
1.4 Bristol eSleeve arm-worn wearable computer. [RM02] . . . . . . . . . . . . 16

2.1 VRIO Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.2 A typical client device . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.3 The VRIO server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
2.4 XML Schema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.5 Collaboration Diagram for VRIORequest . . . . . . . . . . . . . . . . . . . . 28
2.6 Embedded ViaVoice components . . . . . . . . . . . . . . . . . . . . . . . . 31
2.7 Building Vocabularies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
2.8 Input Files for Speech Driven Applications . . . . . . . . . . . . . . . . . . . 40

3.1 User Wearing Shutter Glasses . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.2 User in the CAVE - Holodeck Scenario . . . . . . . . . . . . . . . . . . . . . 46

4.1 Public (video) wall at Zurich station; digital whiteboard in an office [MIEL00] 48
4.2 WebWall framework architecture. . . . . . . . . . . . . . . . . . . . . . . . . . 53
4.3 WebWall framework architecture. . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.4 Service Class Note: Styles yellow, blue, red, one. . . . . . . . . . . . . . . 56
4.5 Service class Gallery. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
4.6 Statechart Diagram, First Approach . . . . . . . . . . . . . . . . . . . . . . . 61
4.7 Statechart Diagram, Final Version . . . . . . . . . . . . . . . . . . . . . . . . 62
List of Tables

2.1 Memory Requirements I [EVV01] ........................................... 32
2.2 Memory Requirements II [EVV01] ........................................ 33
2.3 Codec Audio Requirements [EVV01] ...................................... 33

4.1 Interaction via SMS through GSM ....................................... 65
4.2 Interaction via iPAQ (Pocket Internet Explorer) ....................... 66
4.3 Interaction via Voice Commands ......................................... 67
4.4 Interaction via Voice Commands, part 2 ............................... 68
4.5 Interaction via Voice Commands, part 3 ............................... 69
4.6 Interaction via Voice Commands, part 4 ............................... 70
4.7 Interaction via Voice Commands, part 5 ............................... 71
Bibliography


Bibliography


[KRV00] Kranzlmüller, Dieter; Reitinger, Bernhard; Volkert, Jens: Experiencing a Program’s Execution in the CAVE. In: Proceedings PDCS ‘00, Conference on Parallel and Distributed Computing Systems, 2000. – Las Vegas, USA


[KRV01b] Kranzlmüller, Dieter; Reitinger, Bernhard; Volkert, Jens: VRIO - An Appliance for Natural Interaction with Computers. In: Bauknecht, K.
Bibliography


[RKV01] RETINGER, Bernhard; KRANZLMÜLLER, Dieter; VOLKERT, Jens: The MoSt Immersive Approach for Parallel and Distributed Program Analysis. In:
Bibliography


[Vog02] Vogl, S.: Coordination of Users and Services via Wall Interfaces, University of Linz, Austria, Diss., 2002


85