Automatic commentators

Master's Thesis
University of Turku
Department of Information Technology
Computer Science
2004
Antti Siira

Reviewers:
Harri Hakonen
Jouni Smed

UNIVERSITY OF TURKU
Department of Information Technology

SIIRA, ANTTI: Automatic commentators

Master's Thesis, 49 p., 7 app. p.
Computer Science
March 2004

Automatic commentators and augmented reality are two new research topics that so far have been separate. This thesis creates a link between the two topics and explains the benefits of examining them together. The general structure of an automatic commentator system is described and each part of it is examined in detail.

So far the research on automatic commentators has concentrated into bringing an enhanced experience to sports events. This thesis takes a more general view of the problem by aiming at producing objective commentator systems making the flavor-generation part a separate subsystem. The presented methods and ideas are tested and verified using a case study of an implemented commentator system.

Keywords: automatic commentator, pattern recognition, artificial intelligence, human-computer interaction, augmented reality

TURUN YLIOPISTO
Informaatioteknologian laitos

SIIRA, ANTTI: Automaattiset kommentaattorit

Pro gradu –tutkielma, 49 s., 7 liitesivua.
Tietojenkäsittelytiede
Maaliskuu 2004

Automaattiset kommentaattorit ja lisätty todellisuus ovat kaksi uutta tutkimusalaa, joita on tähän asti käsitelty erikseen. Tämä pro gradu -tutkielma yhdistää nämä tutkimusalat, sekä esittää mitä hyötyä alojen tutkimisella yhdessä voidaan saavuttaa. Lisäksi tutkitaan automaattisten kommentaattorien yleistä rakennetta, sekä rakenteen eri osien toimintaa.

Tähänastinen automaattisten kommentaattorien tutkimus on keskittynyt tunnelman luomiseen urheilutapahtumiin. Tämä tutkielma tarkastelee automaattisia kommentaattoreita objektiivisina järjestelminä, joissa tunnelman luominen on eriytetty omaksi osajärjestelmäkseen. Esitetyt ideat on verifioitu toteuttamalla automaattinen kommentaattorijärjestelmä.

Asiasanat: automaattinen kommentaattori, hahmontunnistus, tekoäly, ihminen–tietokone vuorovaikutus, lisätty todellisuus

Table of Contents

List of Figures

List of Tables

Acronyms and Symbols

VR	Virtual reality
AR	Augmented reality
AV	Augmented virtuality
RV-continuum	Reality-Virtuality-continuum
AI	Artificial intelligence
PR	Pattern recognition
MVC	Model-View-Controller
FFEN	Feed-forward expert network
DGSM	Deterministic generalized sequential machines
FT	Finite transducer
DAG	Directed acyclic graph
NLP	Natural Language Processing
e	Event
d	Event description in internal form
m	Message
o	Explanation of an event intended for output
c	Confidence level
i	Importance value
t	Time
t_e	Time stamp
l	Cognitive load
θ_r	Repeat threshold

tb	Theme bonus

Chapter 1

Introduction

This thesis focuses on two new research topics, namely *augmented reality* (AR) and *automatic commentators*. AR serves as a background for the rest of the thesis since it sets the direction of approach to automatic commentators. The automatic commentators are first studied with providing a theoretical background in Chapter 4 and then by examining a case study of RoboComm in Chapter 5, an automatic commentator for the RoboCup Soccer Simulator (see Section 1.2).

1.1 Augmented Reality

Augmented reality belongs to a set of sub-disciplines of *virtual reality* (VR). As a research area of its own it is fairly new as the technology required in the development has not existed or has been too expensive. However, recent developments in VR specific hardware have made more complex tasks possible.

The definition of augmented reality has not yet been coined, but this thesis uses the following definition:

Definition 1. *A augmented reality system is a real-time system that modifies the input to the user's sensory system with artificially created stimuli.*

The user's sensory system is clearly in a central position of Definition 1 as it is to most VR research, and it will be further discussed in Sections 2.1.2 and 2.1.3. The input

modification requirement is clarified in Section 1.1.1, where it is used to partition VR into several distinct sub-disciplines. The need for a real-time system is not as obvious as the other parts and it can be relaxed in some applications depending on how strictly the reality is defined.

1.1.1 From virtual to augmented reality

In order to understand the differences between sub-areas of virtual reality and to put augmented reality into its place within this context, the *Reality-Virtuality-continuum* (RV-continuum)[MTUK94] can be used to classify the approaches (see Figure 1.1).

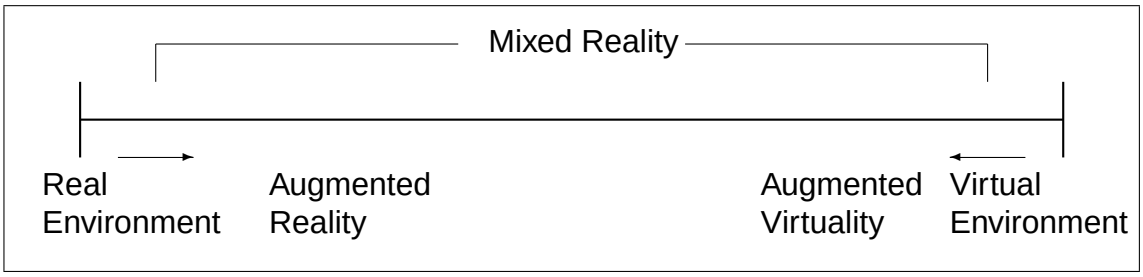

Figure 1.1. Milgrams RV-continuum [MTUK94]

Virtual reality is defined as an artificial environment provided by a computer in which ones actions partially determine what happens in the environment consisting solely of computer generated sensory stimuli. VR is the most widely studied part of RV-continuum along with telepresence as they are the easiest ones to implement.

Augmented virtuality (AV) is virtual reality with some real-world objects. For example, a chat room with avatars that allow users to communicate via microphones is a completely virtual environment that is augmented with real voices generated by the chatters.

Augmented reality is fundamentally the same as AV, the only difference being the ratio between real/virtual environment. The distinction between augmented virtuality and augmented reality is not a clear one. To clarify we can think about the film *Who framed Roger Rabbit* where half of the film happens in real world with cartoon characters (AR) while the other half takes place in cartoon world with embedded human actors (AV). AR equipment usually includes wearable computers or desktop computers that alter and display incom-

ing data stream from video camera. This is usually referred to as a *Windows-on-World* concept. AR enhances the users senses by overlaying a perceptible virtual layer on the physical world [Azu97].

Diminished reality is a special case of AR. In diminished reality nothing is added to user's perception but instead some elements are removed from it. Examples of such are ear-plugs that reduce the amount of audio input to the person wearing them or colored glasses that filter out certain colors from user. More complex examples include a video stream where objects are removed from images and the background is extrapolated to fill the void or an audio environment where vocals are removed from the music.

The *real environment* in Milgrams RV-continuum (Figure 1.1) refers to telepresence. Thus, it is actually real world perceived by external sensors, transfered through a medium and put unaltered to the user's field of perception. Telepresence is used, for example, in controlling vehicles in hazardous environments.

1.2 RoboCup Soccer Simulator

The RoboCup is a international joint project to promote *artificial intelligence* (AI), *robotics*, and related field [Fed]. The ultimate goal of the RoboCup is to create a fully autonomous humanoid robots that can win against the human world champion team in soccer by the year 2050. The RoboCup Soccer Simulator [ROB], a part of the RoboCup project, is a software simulator created to study the AI problems involved in RoboCup.

The RoboCup Soccer Simulator consists of a server software, that runs the simulation, and a monitoring tool, that provides the visualization (see Figure 1.2). The players for teams are created as separate software agents that may not communicate with each other, except via the methods provided by the server. These software agents compete as teams in a game of soccer.

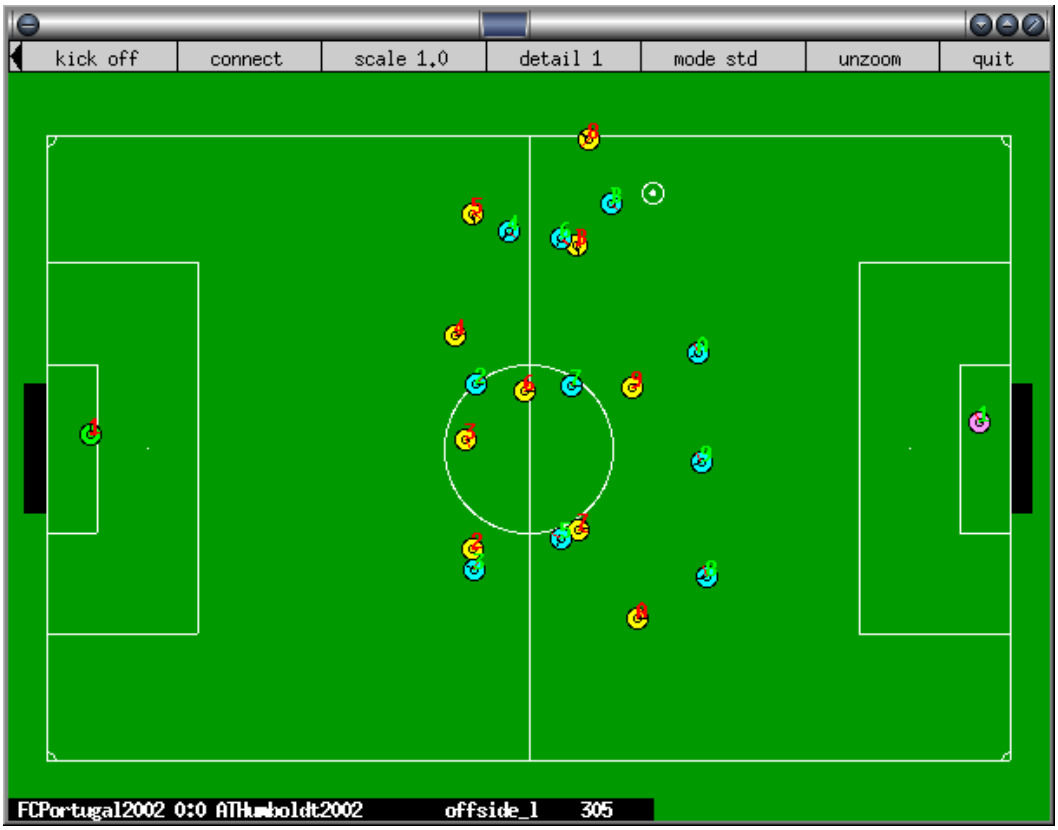

Figure 1.2. RoboCup Soccer Simulator.

1.3 The problem and used research methods

The research on automatic commentators has so far been extremely limited and exploring the topic is like mapping an uncharted territory. Although there are some implemented systems in computer games, documentation about their details has not been published and their performance seems to be somewhat limited.

The main focus of this thesis is to find problems that a commentator designer will face as well as possible solutions for them. Because many of the concrete issues are application specific, this thesis concentrates on the general internal structure of a commentator. Also, tools for dealing with specific types of concrete problems are presented in this work.

Most of the results presented in this thesis come from an empirical study of creating a commentator system, RoboComm, which is presented as a case study. An exception to this is the Chapter 2, that gives a review of examples and available literature. This is

mainly because of the lack of available AR hardware.

1.4 Anticipated results and the boundaries of thesis

As the commentators have been explored only by a limited amount by academic studies, there is obviously room for improvement. Therefore the goal of this work is to find some general guidelines that can be used in design and as a basis of further research. Another goal is to show that automatic commentators can be seen as augmented reality systems and can be studied as such.

This thesis mainly concentrates on the internals of a commentator system. Hence the problems of input and output receive only a cursory look. As most of the actual pattern recognition and artificial intelligence methods are highly application dependent, only a few of them are presented in this work.

Chapter 2

Literature Review of Augmented Reality

Augmented reality is a fairly new area of research about the synthesis of real and virtual world, and, as such, it provides an interesting background for automatic commentators.

2.1 Applications of augmented reality

AR has a variety of practical applications ranging from entertainment to military uses. These applications are the driving force behind AR, because they set the goals for future studies.

2.1.1 Virtual space

The virtual objects inserted to the user's sensory input in AR can be divided to two distinct categories called *supplementing* and *supporting* depending on their natural role in the environment. The user should not be able to identify artificial supplementing objects from real ones. Objects in supporting category do not have to fool the user into believing in their existence but they are normally used to present some additional information to user such as time or temperature.

2.1.2 Enchanting the reality

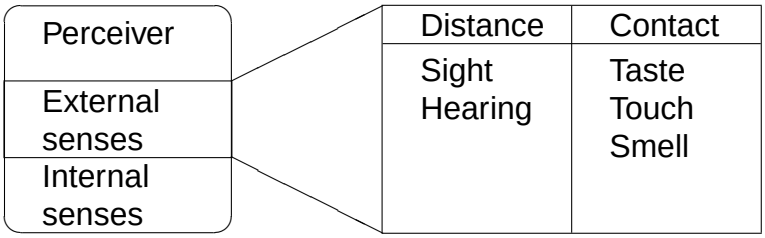

Figure 2.1. Human sensory system

All virtual reality systems are based on human senses, so understanding the way humans perceive their environment and understanding the limitations of natural sensory system are important in AR research. As shown in Figure 2.1, senses can be separated to internal and external senses. Internal senses such as hunger, balance, fatigue and pain are currently out of our reach because they cannot be altered directly from the outside.[1] On the other hand, external senses are relatively easy to access. The external senses category can be further subdivided to long range senses and contact senses based on the means of delivering them. Long range senses are used to receive signals through a medium, and, as they are the easiest ones to alter, they are the primary area in VR research. Due to technical limitations, contact senses are currently considered only a secondary priority.

2.1.3 Fooling the senses

To be able to create supplementing objects, the AR system is required to the present realistic physical events to the sensory system. In other words, *suspension of disbelief*[2] should not be necessary. Humans have a highly sophisticated visual system and it is not an easy task to fool a person into believing the existence artificially created objects. In general there are several issues concerning this:

- registration of virtual objects in 3D,

[1] Inflicting pain might be possible but it is not necessarily a desirable feature.

[2] The willingness in the reader or viewer to suspend their critical faculties in order to enjoy the experiences.

- photo-realistic images,
- low latencies, and
- natural response of virtual objects.

Studies [PSLJ99, Azu95] show that human visual system can detect errors in the alignment of objects up to a fraction of degree. To illustrate, the width of a full moon is about 0.5° when observed from the Earth. The low latency requirement is connected to the registration problem. Because the peak velocities of head angular movement can reach $370^\circ/s$ (although typically maximum peak value in normal use is $120^\circ/s$), hence slow rendering can cause objects to lag behind. On the other hand, virtual objects that react to user's actions need to respond fairly quickly to maintain realistic appearance. It has been suggested that to keep the displacement of objects below 0.25° the maximal lag should not exceed 5 ms, although many authors claim that 10ms would be sufficient in order to maintain sufficient realism.

Visual augmentation is the most widely studied method[3] in AR field, because it is relatively easy to produce and because the results are more impressive than audio augmentation. Although the creation of basic realistic audio environment is easier than visual one, a more sophisticated virtual audio output should take user's surroundings into account and be able to produce 3D audio environment that matches the acoustics of the user's whereabouts. This will clearly be a more complex task than matching the lightning and theme of visual cues.

The other external senses presented in Figure 2.1 are not widely used in virtual reality systems, because of implementation difficulties. For instance, creating a device that produces taste would probably need obtrusive gadgets placed in the user's mouth, making the acceptance of such features quite hard. There has been some successful implementations of touch producing devices, but they are usually limited to user's hands. One of these implementations is a palmtop-system with force-feedback, which is used to study placement and moving of artificial objects in real environment. When compared

[3] In particular registration and rendering problems.

to visual-only system, force-feedback leads to faster alignment and smaller placement errors [NMK96]. This suggests that research on integration of different output methods into multisensory systems will become more important in the future.

2.1.4 Improving the senses

One of the main uses of an AR system is its ability to improve the senses of the user, which is a task that historically has been mainly studied by military organizations. One example of recent military studies includes widening users vision to infrared bands [SWS00]. Civilian uses could include thermograph mounted on cars that detect pedestrians and animals even in darkness. One commercial civilian system, *vOICe* [Mei02], transforms visual information from a head-mounted camera into audio, enabling blind persons to “see” the environment.

Range	Example
Internal	heart rate monitor, body thermometer
Near	thermometer
Location based	other people, building information
Long	camera, microphone
Remote	satellite image, webcams

Table 2.1. External sensor examples

2.1.5 Classical systems

There are numerous examples that could be categorized in augmented reality, even if they have been invented before AR itself. Although devices such as hearing aids and eyeglasses could be included in loosely-defined AR, they are excluded in favor of the definition given at the beginning of this chapter, since hearing aids do not introduce any new information to user or modify them significantly. On the other hand, systems such as thermograph can be included in classical AR systems, because they display new objects that humans cannot normally perceive.

2.1.6 Entertainment

Entertainment is one of the most obvious uses for augmented reality. One of the key elements in computer games is to immerse the player completely in a world created by game designers, which requires that the surroundings are as realistic as possible. Since there can hardly be no more realistic world than the real world itself, AR offers great promises for the future of games. Augmented reality can also allow a total control over player's senses, which enables to reduce the distracting. Virtual objects used in entertainment usually fall into supplementing category.

One example is of using AR in entertainment is *ARQuake* [TCD+02] created at the University of South Australia. ARQuake is based on popular computer game *Quake* and it uses wearable computer *Tinmith* (see Section 2.2) to overlay virtual objects to the player's vision, by superimposing a world of monsters and buildings to the real world.

2.1.7 User support systems

User support systems in augmented reality provide information and analysis to users that help in accomplishing the tasks at hand. Classical examples presented in the literature include various repairing tasks ranging from blueprints presented into the view of a mechanic to providing doctors with real-time ultrasound images of patient's bodies. When user support systems are combined with AR the result is called an *intelligence amplifying system*. [Bro96]

Remembrance Agent acts as an automatic associative memory. The software package contains back-end tool *Savant* that parses through files in various formats storing keywords. The front-end Remembrance Agent monitors users activities and provides list of files containing information that it thinks is relevant to current task [Rho96].

2.1.8 Collaborative environments

Adding communication units to augmented reality systems allows the building of collaborative environments where the users may share their virtual space. Collaborative en-

vironments require a 3D registration framework to prevent misunderstandings resulting from different views. In collaborative AR environments, the users are presented the same virtual object possibly from different viewpoints and they are allowed to experiment and modify the models. Example systems ranging from doctors performing cardiology diagnostic to geometry learning environments have shown promising results [Ber98] [SH02]. Collaborative environments can also be implemented in VR, but the process can take advantage of ARs capability of having face-to-face conversations as well as arbitrary real world props.

The shared space technologies are classified based on their *transportation*, *artificiality* and *spatiality* [BGR+98]. The dimension of transportation determines whether the persons and objects are bound by their location. If the transportation allows remote connections, the objects can leave their local space. Face-to-face meetings and immersive telepresence are the opposite ends of this dimension. The degree of common coordinate system among participants determines the spatiality dimension. Obviously this dimension applies only to shared space systems that have a concept of coordinates. The artificiality is already discussed in Section 1.1.1.

2.2 Hardware platforms

Although augmented reality environments can be explored in normal desktop computers (the *windows-on-world* concept), the real power of these systems is achieved with so called *wearable computers* and AR systems embedded in various vehicles. Wearable computers are basically portable computing platforms that are worn by users instead of just carrying them around. Usually such systems include *head-mounded displays*, some sort of keyboard for giving input and a main processing unit.

MIThril is a wearable computing platform developed by MIT Media Lab [oT]. It is intended to provide a development and prototyping platform for researching human-computer interaction for body-worn applications. MIThril consists mainly of hand-made hardware. Detailed instructions are distributed in open source manner.

Tinmith is a wearable computer developed at the University of Southern Australia, which is built from commodity hardware gathered around a generic laptop. It is a general purpose platform with primary usage of exploring real environments modified with artificial buildings, plants and other such objects. The user can build and change the surroundings using data gloves while still inside AR. Tinmith also is the platform that provides the basis for ARQuake (see Section 2.1.6).

Both MIThril and Tinmith are based on Linux and X Window System. Their difference is that MIThril seems to be heading towards elegant hardware platform, while the Tinmith-project is concentrating on prototyping with applications.

Chapter 3

Commentator Overview

The current automatic commentator research started with attempts to create an automatic software soccer commentator for the *RoboCup* [KTS+97]. Because these systems are sports commentator systems, the ongoing research is concentrating on recognizing events and bringing an enhanced experience to games [ABTI+00]. Although sports commentators are a very useful tools for discovering problems and solutions for a commentator system, the uses for a commentator system are much more far-reaching.

The commentaries create so intense experience and additional content to real world sporting events that it is almost impossible to imagine a sports broadcast without it. This is the main reason why computer gaming industry has developed an interest of bringing commentaries to their games, which has been recently increasing [KSH03]. The problem with sports commentators is that they are a special case among commentators, because of their bias is on recreation instead of objective information.

Definition 2. *An automatic commentator examines an external system and produces output that*

1. *helps to understand of the occurring events,*
2. *enhances the perception or experience of the system,*
3. *presents other information relevant to the observed process, and*
4. *operates in real-time.*

The first item in Definition 2 defines objective commentators, whose function is to produce accurate description about the state of world. Objective commentators can be seen as a basis for all other types of commentators as the generated messages can later be biased or transformed to more entertaining form.

The second item allows the inclusion of commentators to be based on their recreational value alone. The commentator may show replays of previous events and highlight or amplify things that are otherwise hard to notice. This part of definition also binds commentator systems closely to augmented reality and its goal of improving the senses (see Section 2.1.4).

The third item describes commentators that give background information for occurred events, compare the current state against history and try to predict the tendencies in the system. To achieve this the commentator should have an inside model about observed system on which it can base the comparisons and predictions.

The last item, requirement of real-time response, separates commentators from general expert systems and narrows down the amount of available methods. Without this requirement the commentator could simply look at the future and do analysis based on that information. For example, a sports commentator could check the final result and bias its comments to favor the winning team. In other words, the real-time requirement preserves the integrity of a commentator. On the other hand, if the commentary lags too much behind of the actual events, the observations do not provide additional value to the user.

As we can see in Definition 2, commentators are *user-centric* systems, which will become more evident in Section 4.4 that deals with problems in message selection. The focus on user is naturally shown also in Section 4.5, when the issues concerning output is discussed briefly.

3.1 A commentator as an augmented reality system

Describing a general commentator as an augmented reality system may seem a little bit unorthodox, because the underlying world can actually be artificially created. This is, however, possible, because the origin of input is not defined in Definition 1.

Because a commentator works by augmenting the user's perception in real time, it fulfills the Definition 1 and, therefore, it can be classified as an AR system. The similarity of these systems becomes more evident when inspecting the general structure of a commentator (see Figure 3.1) and that of an AR system.

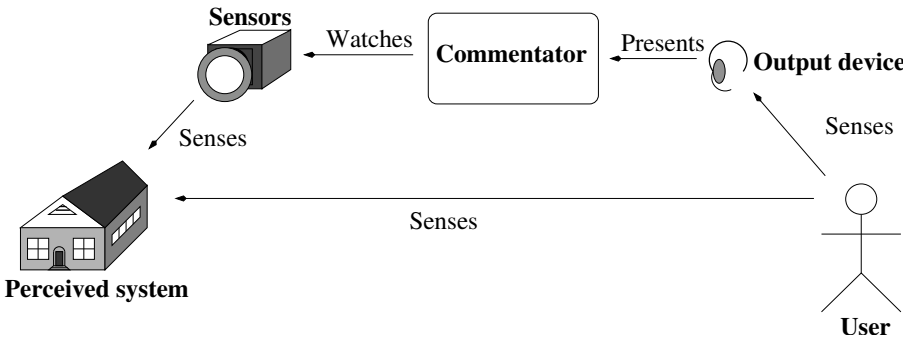

Figure 3.1. A overview of a commentator system

So far the AR research has concentrated on the possible hardware platforms, and the amount and type of content created on these platforms has been quite limited. By relaxing the definition of AR to include *augmented virtual realities*, the experiments for creating content and uses for AR systems becomes easier. This has been one of the reasons why the *RoboCup Initiative* created the separate *RoboCup Soccer Simulator* in order to research multi-agent architectures without having to produce often expensive hardware.

Although commentators and AR systems have much in common, there are few separating issues. Firstly, a commentator is not allowed to invent new entities as most of current AR implementations do. Secondly, the relaxation of the definition of reality is not necessary for commentators as a commentator working on artificial world is completely valid one. Neither commentator nor AR is a subset of the other, but the intersection between them is quite large.

3.2 A commentator as an artificial intelligence and pattern recognition system

A commentator can be seen as an artificial intelligence system, because although it does not have to think about consequences of its actions, it does have to make intelligent reasoning about current and possibly future events and also it has to be able to determine the consequences of actions performed by the detected entities.[1] These properties make commentator a predicting and explaining AI system.

For the purpose of making informative and intelligent comments about perceived system, the commentator has to have some expert knowledge about it. The expert system created for analyzing events can use normal AI techniques as it does not have any special requirements beside the need for real-time response.

The fields of pattern recognition (PR) and artificial intelligence have always been closely connected to each other as the former aims at recognizing entities from the sensory data and the latter tries to understand their meaning. The division of duties is quite similar in commentators. PR methods are used to process input into more computationally feasible form and to recognize events from sequences of states.

Wearable computers and augmented reality provide a challenging and difficult environment for PR and AI because of the limited computational resources, real-time requirements and unconstrained environment.

3.3 High-level structure of a commentator

Examining the high-level structure of a commentator (see Figure 3.1) reveals the classic *model-view-controller* (MVC) pattern. The set of sensors watching over the perceived system controls the changes in commentator. The view part is obviously provided by the output layer, while the pattern recognition and the artificial intelligence part are responsi-

[1] For example, if a commentator perceiving user's surroundings notices a car heading towards the user, it might issue a warning so the danger could be avoided.

ble for the model part. Naturally the parts of the system depend somewhat on each other, for example, the recognized events can depend on the available input to great extent.

This separation of functionality offers many advantages for the design of commentators. Looking from the classical augmented reality perspective the controller and the view are highly hardware-dependant parts of a system. On the other hand, the model can be taken out from the rest of the system and studied separately. When examined separately, the view to a commentator changes from “what can we say with this input” to “what input do we need in order to achieve this”.

Figure 3.1 illustrates that in any commentator system there must be at least three operative parts present: A way to gather input, a module that generates a description of input, and a method of presenting the description to the user. The IO part is hardly unique to the commentator systems, which leads to conclusion that what goes inside the middle layer is the defining part. Indeed, as the structure of the commentator is looked at more closely in Chapter 4, its components are used to classify different types of commentators.

Chapter 4

Structure of a Commentator

A commentator should describe its input to the user in a way that maximizes the usefulness of commentary (as described by a utility function). The utility function associated with a commentator can be quite complex, because its value is ultimately determined by an end-user. This chapter examines general issues for producing high quality output.

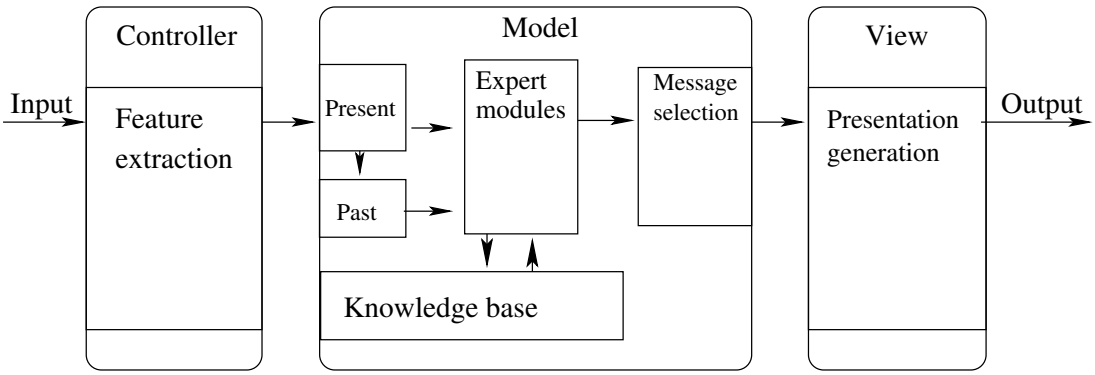

Figure 4.1. A general structure of a commentator system

4.1 Input

The input to a commentator can come from a variety of sources (see Table 2.1 for examples in augmented reality). Usually the amount of input is too large or susceptible to errors, and inputs have to be preprocessed (see Section 4.2) before further analysis. Since commentator is a kind of a pattern recognition system, the preprocessing methods

originate from pattern recognition.

4.1.1 Feature selection

One of the most difficult aspects of pattern recognition is to select a *functional closure* (the smallest subset with desired properties) of the available input. As many PR methods have complexity greater than linear, the amount of input has tremendous affect on performance. In PR research this is usually referred as *the curse of dimensionality*. Since in PR there is no such thing as negative information, the removal of features cannot lead to a better classification. The worst case is completely overlapping classes, which results in zero gain.

The selected features should have as much discrimination power as possible. To achieve this the selected features should have *a large between-class distance and small within-class variance* in feature vector space [TK99]. Figure 4.2 provides an example of a feature space.

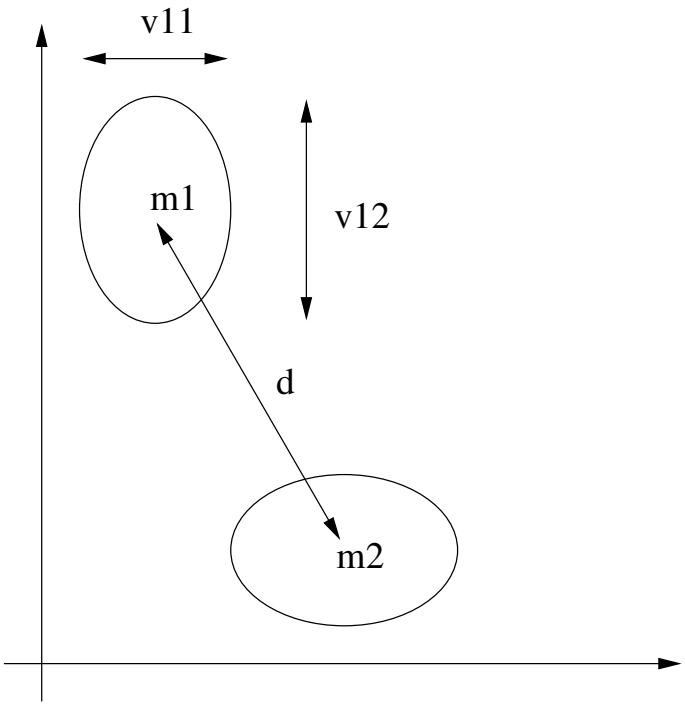

Figure 4.2. Example of a feature vector space. A large *d* (distance between class means *m1*, *m2*) and small *v11*, *v12* (variances within-class) bring more discrimination power.

4.1.2 Data representation

Selected values that represent measured values from sensors are grouped in a *feature vector* $x = [x_1, x_2, \ldots, x_N]$. Each of the features in vector x belongs to a corresponding set $X_1, X_2, \ldots, X_N$ that can be either numerical (quantity) or symbolic (label). Any complex measurements (combination of numerical and symbolic values) can be flattened to a feature vector. For example, pixels from $I \times J$ sized image can be inserted to feature vector so that $x_{k+1}, x_{k+2}, \ldots, x_{k+I*J}$ represent the corresponding values.

Features can be categorized into four different basic types based on their relative significance (see Table 4.1 for examples). Each successive class is based on the previous classes by attaching some new meaningful property. *Nominal* features are labels for different categories and they cannot be ordered in any meaningful way. They form a simple equivalence class with no other property than '$=$'. *Ordinal* features are nominal features that have introduced operator '$<$' between the classes thus forming a ordered set. *Interval-scaled* features are ordinal features with operator '$+$' dividing the sets in some intervals. With interval-scaled features it is meaningful to speak about the amount of difference between classes. The last and the most expressive class of features is the *ratio-scaled* features. Ratio-scaled features have defined operator '$*$' to class and one can measure the absolute difference thus giving meaning to sentences such as *twice as much* or *half the amount* [TK99].

Feature type	Example
Nominal	Male–female, true–false, motorcycle–car-truck
Ordinal	Very small–small–normal–large-very large–shuge
Interval-scaled	Degrees Celsius
Ratio-scaled	Weight, pressure

Table 4.1. Feature types and examples

4.1.3 Static vs. dynamic input

The type of input can be used to classify commentators to two element types. *Static commentators* (for example, commentator describing paintings) do not have the concept of time so there is no need to refer to the previous states (history). On the other hand, *dynamic commentators* (for example, sports commentators) have to take the previous states into account, which is why they need to look at the incoming feature vectors as time-series data $x(t)$. Of course one can model static commentators as dynamic commentators by defining

$$x(t) = \begin{cases} [x_1, x_2, \ldots, x_N] & t = 0 \\ \text{zero vector} & \forall t \neq 0 \end{cases}$$

Static commentators are not interesting on their own, but they can be used as building blocks when constructing larger systems. The concept of time is discussed further in Section 4.3.1.

4.2 Preprocessing

An *outlier* is defined as a point that lies very far from the mean of the corresponding random variable [TK99]. Many pattern recognition fitness-functions, such as *least squares criterion*[1], are sensitive to outliers and therefore *outlier removal* is usually the first preprocessing method used. In outlier removal, the values outside some predefined range, usually determined by standard deviation, are simply dropped. If the outliers are not a result of errors or noisy data but instead belong to a distribution with long tails, the designer has to select a fitness-function that is relatively insensitive to the outliers.

Sometimes all values are not available for processing (this can be due to faulty sensors or removal of outliers. Usual methods of coping with missing data are dropping the entire vector, estimation of the value using class mean or, in the case of time series data, extrapolation of value using history. The fitness of the selected method depends greatly

[1]Selects x for which $min(\sum_{n=0}^{N} E(x_n))$, where $E(y)$ is the error function.

on the type of feature in question.

Data normalization is used to transform different types of input to more easily processed form. The most common normalization method is standard transform of random variable to unit normal distribution, see Equation 4.3, that shifts variable mean to 0 and variance to 1. Another common normalization is achieved by using *sigmoid*-function to produce so-called *softmax* (4.4) scaling.

$$\bar{x}_k = \frac{1}{N}\sum_{i=1}^{N} x_{ik} \quad \textit{(sample mean)} \tag{4.1}$$

$$\sigma_k^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_{ik}-\bar{x}_k)^2 \quad \textit{(sample variance)} \tag{4.2}$$

$$\hat{x}_{ik} = \frac{x_{ik}-\bar{x}_k}{\sigma_k} \quad \textit{(normalized variable)} \tag{4.3}$$

$$\hat{x}_{ik} = \frac{1}{1+e^{\frac{x_{ik}-\bar{x}_k}{\sigma_k}}} \quad \textit{(softmaxed variable)} \tag{4.4}$$

4.3 Analysis of events

The role of the analysis-layer is to find events and entities from the incoming data in a reliable manner. The analysis-layer is probably the most critical in a commentator system, because without reliable data to pass to message selection layer, the commentator has hardly anything but humor value.

4.3.1 Time

As mentioned in Section 4.1.3, time plays a crucial role in many commentator systems, because it enables the concept of events (i.e. changes in the system). The classical three-level design hierarchy divides time into *short-*, *medium-* and *long-term* events[2] [KSH03]. Of course the division between categories is completely arbitrary, since no clear boundaries can be given. Nevertheless, the definitions of different time frames can be useful in determining which methods to use in event recognition.

[2]Known also as operational, tactical and strategic levels.

Long-term events

Long-term events are the topmost view of the system and they generally provide information from a statistical point of view. As they are usually based on shorter term events, the amount of data can be large, which means that unnecessary details need to be filtered out. This filtering may cause quantization problems that the designer has to deal with by incorporating *fuzzy classification* or some similar method. Analyzer modules handling long-term events are usually the biggest utilizers of knowledge bases and they try to find the reasons behind the events and fundamental changes in the system. Analyzing long-term events does not have to be a real-time task and thus it can utilize more complex reasoning methods.

Medium-term events

Medium-term events provide a layer between long- and short-term events by grouping multiple short term events to provide information about relations between entities and basic tendencies in the system. The used methods come also from artificial intelligence and pattern recognition. It is much more difficult to define medium-term events than long- or short-term ones due to its dynamic nature as an interconnecting layer. On the other hand, middle-term events seem to be the most interesting ones from the implementor's point of view. Because middle-term events are recognized by order of magnitude more often than the long-term ones, puts limits to the set of the usable methods. Middle-term analyzers deal with relatively concrete events, and therefore they cannot resort to using mere statistical analysis as the long-term ones and they also do not have the luxury of dealing with primitive events of high confidence that the short-term events have.

Short-term events

Short-term events describe the basic changes in the perceived system, and their effects are usually concrete and easily distinguished. Because their effects vanish rapidly, the recognition methods must be fast but not necessarily highly accurate. The diminished

need for accurate recognition is due to the small scale effect of each individual event, and because their number is usually large, the errors tend to average out. Short-term events are normally handled with various pattern recognition methods and they rarely provide predictive information about the system.

Atomic events

It is also useful to introduce *atomic* events that differ from the three other categories in that they are not inferred from other events. Atomic events happen instantaneously from the commentator's point of view. Interestingly, analyzers of such events are actually static commentators. In practice, atomic events are usually grouped together with short-term events, but in commentator systems this division of events is beneficial.

4.3.2 Modeling the analyzer as an expert network

An analyzer can effectively be modeled with a *feed-forward expert network*[3] (FFEN). An FFEN pushes knowledge forward and gets more and more refined results after each node. Each node is considered a sole expert in its domain, although internally it can contain several sub-experts. Because the node is the sole expert for a given topic, there are no conflicting events passed around the network. The absence of conflicting events provides a nice property of not having to implement probabilistic reasoning methods at the network level, which simplifies the overall structure. It is quite easy to incorporate new expert nodes in FFEN, since a new node cannot break any existing functionality. The cost of this property is that the complexity is pushed inside each node, which can lead to difficult implementation issues. One particular drawback in FFENs is that there is no clear way to determine the order of arriving events beforehand. This also leads to complex implementation issues as the node has to deal with all possible combinations of incoming events.

A node in FFEN is defined by its inputs, a set of possible outputs, internal reasoning algorithm and connections to outside knowledge sources. The knowledge sources are

[3] A *pipes and filters* architecture with additional constraints.

discussed further in Section 4.3.4. The topology of FFEN is *directed acyclic graph* (DAG).

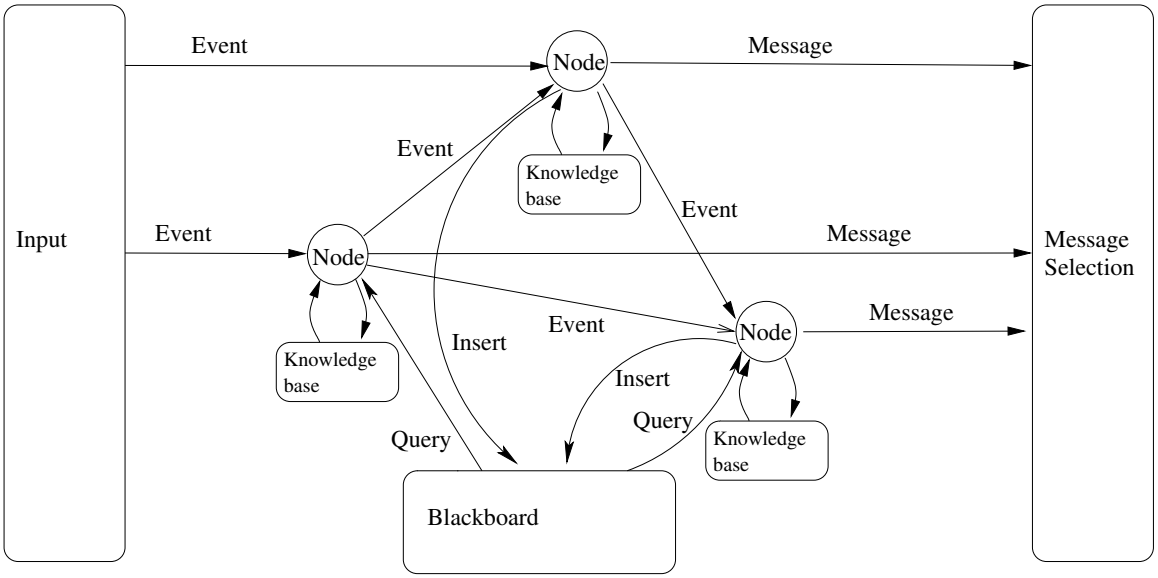

Figure 4.3. Example of an FFEN

An event in FFEN is a triplet consisting of *event description*, *confidence* and *time stamp* (event $e = (d, c, t_e)$). The event description is the actual event the node claims to have detected, confidence is a value from the range $[0, 1]$, where 1 indicates absolute certainty and 0 a blind guess, and time stamp is the moment in time this event has occurred. In particular, *null event*[4] belongs to the set of messages any node can output. The network assumes that the confidence level associated with a given event is correct because it is the responsibility of a sole expert. In FFEN there can not exist a path from any node to itself, which means that a node is not allowed to change its own input directly. This is because an FFEN is a *push-network*, and if we allowed such cycles it could easily lead to unstable system where events would amplify themselves or infinite loops.

The method of calculating confidence levels plays an important role in the creation of a stable FFEN. Although the implemented method depends on the problem domain, there are desired properties all methods should have. The confidence value of an event should have a lower bound of 0 and an upper bound of 1 or the maximum confidence of its inputs. These requirements follow from that the confidence is based on the previous information,

[4] A null event is a way for a node to say “I did not detect anything.”

and, thus, it cannot be more certain than its predecessors. Also, the confidence level should not decrease, if more sources of outside information are added. Because the new connections cannot provide conflicting information concerning the event in question, additional information can only strengthen recognition confidence. This can also be seen as a result of the absence of negative information in pattern recognition as discussed in Section 4.1.1.

One general pattern of a recognizing node is the *trigger-node* type. A trigger node requires certain amount of events to arrive to its input before it sends its own message forward. Before the node sends a recognition event forward, it can send null events or simply abstain from sending any events depending on the implementation. The calculation of the node's confidence can be done, for example, by simply counting the product of confidence level in all required events and discarding the rest. The confidence level of incoming events can be weighted by a constant a_i, but usually this is not necessary that is $a_i = 1$. It is the responsibility of designer to ensure that the final result does not exceed maximum confidence of 1.

$$c_{out} = \prod_{i=0}^{N} a_i c_i \quad \forall c_i = \{c | e = (d, c, t_e), e \textit{ is relevant to recognition}\}$$

Another common pattern is a *time-triggered node*, that is activated at predefined intervals or by otherwise defined moments in time. This method should generally be used as rarely as possible, because the time intervals tend to group together causing the output messages to occlude each other. Therefore, if time-triggered nodes are widely used in a single commentator, the designer has to ensure that the moments are well distributed, for example, by selecting intervals that are relatively primes. The time-triggering also seems an artificial way to determine the need for output, unless the problem domain includes some special moments in time, for example, half-time or the end of game in a sports match.

The last trigger-pattern is a *cumulative node* pattern that accumulates certainty over time and sends an event forward at each clock tick with the current confidence. If subsequent events do not agree with the ongoing recognition, they can have a negative effect on

the confidence. This does not conflict with the requirement that more information should not decrease confidence, because the internally saved information can be allowed to behave as it wants. Cumulative certainty can be modeled, for example, by adding all input events (possibly over time) together and squashing the sum with a function that limits the output to the interval $[0, 1]$. Such a function is of course the sigmoid-function used in calculation of softmax (see Equation 4.4).

The final problem with confidence levels is the assignment of confidence to atomic and internally reasoned events. One possibility is to assign these values or methods by hand, and hence in this case the confidence is based on the system designer's authority. Other possibilities include *supervised learning methods* and estimating performance values from the data.

FFEN is a good way to *push* events forward, but the creation of analyzers becomes easier if there is a way to *pull* events. The pull model can be realized with a blackboard that every analyzer is allowed to use for inserting or requesting data. It is also a good place to gather history of events, should some analyzer require the previous events.

4.3.3 Methods

Because the methods for recognizing atomic events (see Section 4.3.1) are closely related to the observed domain, and it is quite difficult to give general guidelines for creating them. Dependencies between the problem domain and recognition methods also affect other layers, but there are some useful generalized methods that can be applied here.

Deterministic generalized sequential machine (DGSM) is a useful method for recognizing short- and medium-term events. A DGSM is a special form of a *finite transducer* (FT) that reads its input one character (event) at a time, does not read empty word from its input (but may print one), and has only one possible transition for each state-input pair.

Definition 3. *A deterministic generalized sequential machine is a six-tuple*

$$T = (Q, \Sigma, \Delta, \sigma, q_T, F)$$

where

- Q *is a finite set of states,*
- Σ *is the input alphabet,*
- Δ *is the output alphabet,*
- σ *is a mapping* $\sigma : Q \times \Sigma \rightarrow P(\Delta^* \times Q)$ *such that* $\sigma(q_i, \Sigma_i) \rightarrow (\Delta_n, q_n)$ *and* $\sigma(q_i, \Sigma_i) \rightarrow (\Delta_m, q_m) \Leftrightarrow \Delta_n = \Delta_m \wedge q_n = q_m$
- q_T *is the starting state, and*
- F *is the set of end states.*

The inputs to event recognizing DGSM (see Figure 4.4 for graphical representation) can be divided to four basic categories:

- *F-events*, that move forward and output nothing,
- *N-events*, that have no effect on the internal state and output nothing,
- *I-events*, that invalidate the current event and output nothing, and
- *A-events*, that move forward and print recognized event.

DGSM structure suits for medium-term and short-term recognition because it is simple, deterministic, and its implementation is easy to understand. Of course a DGSM can be expanded to recognize multiple or even successive events.

4.3.4 Sources for outside information

A variety of *knowledge bases* can provide additional intelligence to a commentator and allow, for instance, a comparison of current events against history. In addition to historical comparison, the knowledge bases can be used to store information about entities and relations between them. Such information would be highly beneficial in the creation of context aware commentator.

A knowledge base provides similar function as the blackboard concept, which was introduced earlier. Expert nodes can insert or update information residing in a knowledge

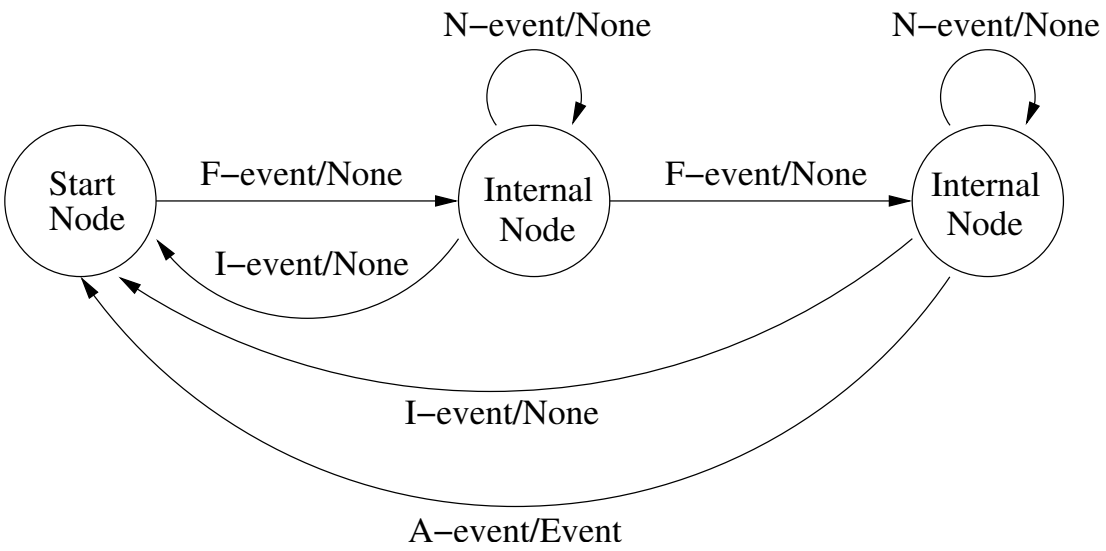

Figure 4.4. Simplified representation of a deterministic generalized sequential machine

base as well as query information from it. The difference between them is that the main utility of a knowledge base is not solely to relay data between nodes but also between designer and nodes. Furthermore, the information stored in such base is meant to carry persistent information about more general patterns of the problem domain and not only data that is relevant to the task at hand.

4.3.5 Problems with incorrect recognition

The performance of FFEN can slowly deteriorate, if false premises created by incorrect analysis are stored and referred later or used in cumulative calculations. Therefore, the system should be created so that the effect of historical values decrease as time progresses, thus making the system more resilient against errors. The effect of a false recognition is already somewhat reduced in confidence level calculation, if method presented in Section 4.3.2 is used.

4.4 Message selection

The role of the message selector is to decide *what to say* and *when to say* it by limiting the number of messages to the output, thus preventing the user from being overwhelmed

by information. The message selection is effectively the one separating factor between commentator systems and more general expert systems. Message selection has close ties with the output part of *Natural Language Processing* (NLP), as the goal of NLP is to make a computer communicate with humans.

4.4.1 Message internals

A message is fundamentally the same as the events passed around between nodes, but because its intended target audience is a person instead of another program it is slightly modified.

Definition 4. *A message is a five-tuple*

$$m = (o, c, i(t_\Delta), l, t_e)$$

where

- o *is the textual explanation of an event,*
- c *is the confidence level,*
- $i(t_\Delta)$ *is the importance function describing the vitality of event at time* $t_e + t_\Delta$*,*
- l *is the amount of time needed by the user to interpret event* $l \in]0, \infty[$*, and*
- t_e *is the time when the message was generated.*

The c and t_e parts of an message are obvious since they are the same as the corresponding properties of an event. Term o is also the same as d in an event but with the requirement that it is open for a human to interpret. The properties $i(t_\Delta)$ and l can be described as follows: As seen in Definition 4, the importance function gives the value of event to the user(assuming that the recognition is correct). The value depends on the difference of time between the event recognition and the current time, because the importance of events tends to decrease gradually as time passes. For example, telling the spectator that player X had the ball a second ago is usually more informative than telling that he had the ball an hour ago. Simplest way to implement $i(t_\Delta)$ is to give a constant

value for the time-frame $[t_e, t_e + n]$ and after that drop the value to 0. Another common technique is to use linear or exponential drop, but the actual function can be arbitrarily complex.

The l value represents the cognitive load of interpreting the message, giving the amount of time needed to wait before putting new message to the output layer. Of course this value is only a suggestion and if important enough value arrives, system should force a priority interrupt [ABTI+00].

4.4.2 Message fitness function

Since message selection is essentially about selecting the best candidate available, the system needs some means for determining the fitness of each message. From the structure of messages (see Definition 4) it follows that the fitness-value can be based on the confidence, importance and the cognitive load.

Creation of a proper fitness function is an optimization process that aims at maximizing the confidence and importance, and at the same time, minimizing the cognitive load of messages presented to the user. The basic fitness function that has these properties is $c * i(t_\Delta)/l$. If a strong correlation exists between the cognitive load and importance (more important messages are usually harder to interpret), term l can be removed from the equation.

4.4.3 Message selection strategies

The fitness function gives an indicator about each message in isolation, but when many messages are grouped together, the commentator system needs to form a selection strategy to optimize the cumulative fitness, average fitness or some other desired characteristic of output messages. The message selection strategy is actually one of the most fundamental factors in commentator system design, because it is responsible for creating a seamless series of messages to output. In a way, it transforms a series of comments to a *commentary*.

A naive message selection strategy is to look at the queue and to take the first message in the queue, if it exists. This is analogous to having no selector strategy at all. The naive selection strategy is included in the comparison (see Figure 4.5) for a rough estimate of how much improvement can be achieved by incorporating more advanced strategy.

Basically all selection strategies should incorporate some sort of *cutoff point* for the process. This point represents a minimum level usefulness. Any message with fitness below the cutoff point is considered so unimportant, that it should not be output even if no other message is waiting in the queue.Unimportant message means one with low confidence and/or importance properties.

Possibly one of the simplest strategies is to evaluate the fitness value for all messages waiting in the queue, drop all messages with a value below some cutoff point, and send message with the highest fitness to the output layer. This strategy is clearly a *greedy algorithm*.

The greedy strategy can easily lead to a suboptimal selection. Consider the case presented in Figure 4.5, where m_1 gets initially chosen instead of m_2. By the time m_1 has been interpreted, the fitness of m_2 has fallen below the cutoff line and is dropped. If m_2 had been chosen first, m_1 would still be above cutoff and they both would be selected because they would yield higher cumulative fitness.

A more advanced strategy could form tree from the possible selections at each point and select the path with the highest cumulative fitness. This is a *dynamic programming* strategy (see Algorithm 1) and it gives the optimal solution for local problem and it solves easily the case where the greedy alternative fails. The problem with this strategy is that it does not take possible future events into account. If we consider the case in Figure 4.6, we notice that dynamic programming algorithm would make suboptimal selection, because at t_0 it would choose m_2 as before, but at $t_{0.5}$ it would choose m_3 yielding worse solution than the optimal sequence $\{m_1, m_3\}$. It is important to notice that algorithms based on dynamic programming are susceptible to the size of the message queue and to the type of the messages in it. Short cognitive loads and long periods of time before

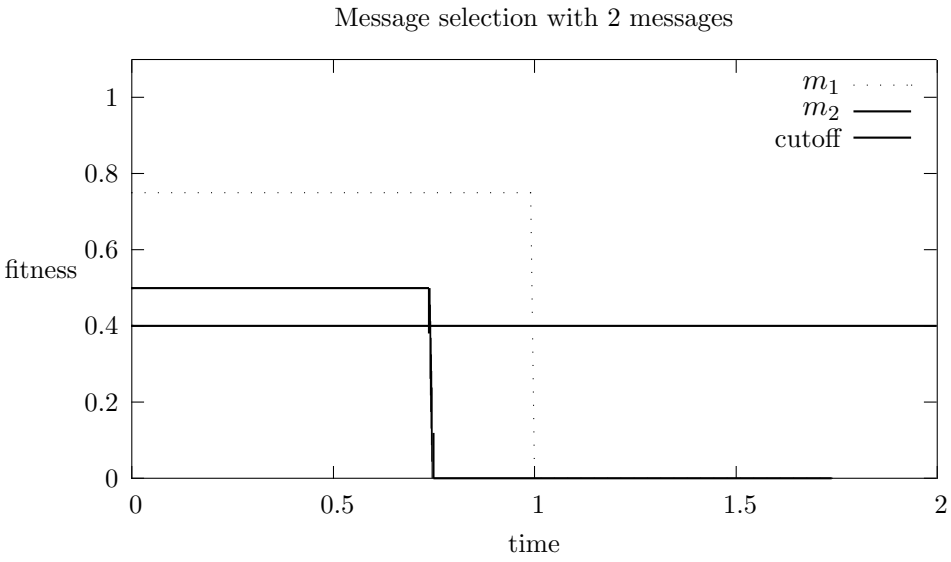

Figure 4.5. Message selection case with $l_{m_1} = 1.5$ and $l_{m_2} = 0.5$

dropping below cutoff in particular can easily cause a combinatorial explosion. This algorithm gets its results by favoring messages with small cognitive load over more important ones.

The greedy algorithm and dynamic programming algorithm base their decisions on local information, but if one has access to information about the distribution of the properties of events by either estimating them from data or by some knowledge about the process of generating messages, these algorithms can be improved. For instance, counting the probability of missing important messages while outputting events and multiplying it with the average maximum message arriving during that interval, the fitness-value can be decreased accordingly. In other words, we try to anticipate the messages that might be missed during the output of a message. As we can see in Table 4.2, the dynamic selection strategies outperform the greedy strategy both in the sum of fitness and number of output messages. On the other hand, incorporating probabilistic reasoning to these algorithms does not create enough improvement to justify more complex implementation. In greedy algorithms, the effect is actually negative. When compared against the naive strategy, the benefits of other selection strategies become evident: Even the worst selection strategy achieves 160% gain over the naive one in the total sum of fitness. The

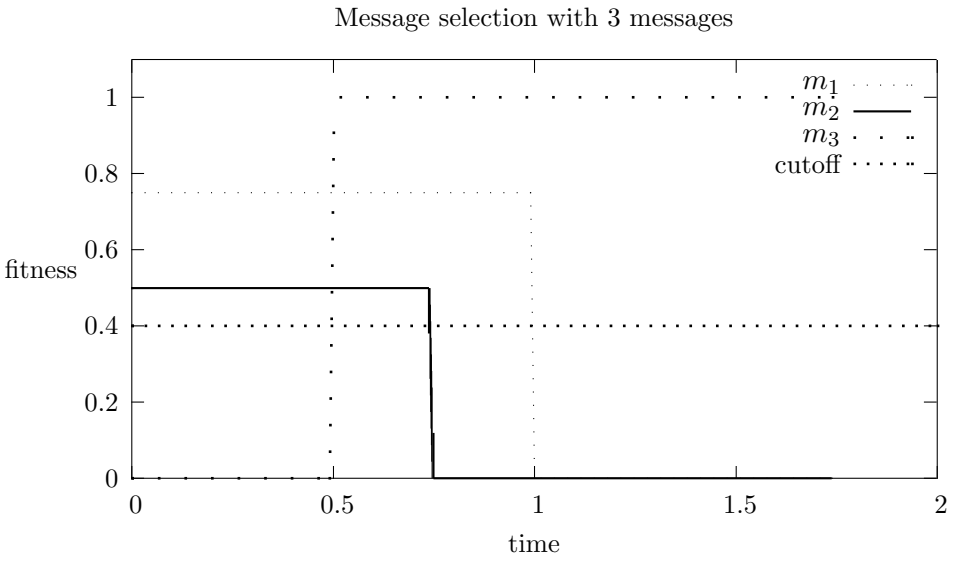

Figure 4.6. Message selection case with $l_{m_1} = 1.5$, $l_{m_2} = 0.5$ and $l_{m_3} = 1.25$

details of the simulation are described in Appendix A.

Algorithm	Sum of fitness	Average fitness	Number of messages	Declines
Naive	84	0.21	404	0
Greedy	137	0.40	346	0
Probabilistic greedy	134	0.38	355	1
Dynamic	208	0.26	802	0
Probabilistic dynamic	212	0.26	828	1

Table 4.2. Results from simulation runs (described in Appendix A) with 10000 messages. Declines indicates the number of times the selector decided not to output messages, even if there were suitable candidates in the queue.

4.4.4 Theme augmented selection

A theme based selection strategy tries to create meaningful chains of outputs by favoring consecutive outputs belonging to the same logical group. This effect can be easily incorporated into an existing commentator by giving a fitness bonus to all messages in the same group as the last output message. The benefit of this strategy is that the output

Algorithm 1. Selector strategy based on dynamic programming

1: DYNAMIC-SELECT(M,t,c)
2: **In:** available messages M; time t; current node c
3: **Out:** fitness value assigned to c
4: $M \leftarrow M - c$
5: $N \leftarrow \emptyset$
6: **for all** $i \in M$ **do**
7: **if** FITNESS(i) $>$ cutoff **then**
8: $N \leftarrow N \cup i$
9: **end if**
10: **end for**
11: **if** $N = \emptyset$ **then**
12: return FITNESS(c)
13: **end if**
14: $m_{current} \leftarrow 0$
15: **for all** $j \in N$ **do**
16: $m_{current} \leftarrow$ MAX($m_{current}$, DYNAMIC-SELECT(N,$t + d_i$,i))
17: **end for**
18: return FITNESS(C) $+ m_{current}$

becomes easier to follow, and the cognitive load can also be reduced.

Theme based selection is not exactly a stand-alone message selection strategy but rather an improvement to existing ones. Adding such feature to a commentator means that the designer must place different messages to groups by hand and decide how closely related they are (i.e., how much bonus do they get). It could also be useful to incorporate context awareness to the message selector that would prevent nonessential messages if important messages, such as a possible goal, are anticipated in the near future.

4.4.5 Repeated message problem

One of the goals of the commentator is to produce a high-quality description of the ongoing events in the observed system, and the same messages output multiple times in succession reduce the usefulness of a commentary. For example, sports commentator that tells viewers that player X controls the ball once every second would be quite annoying. Even if the commentator waits for the time indicated by the cognitive load, the delay between similar messages can be too low.

One way to deal with repeated messages is to build the commentator so that it produces only output about changes in the system and not about the state of the system itself. In other words, this means that the commentator does not produce a message if it has already done it in the recent past and the situation has not changed in the meantime.

Other way to deal with repeated messages is to implement a *repeat threshold* $\theta_r(m)$ for messages by branding a message as *taboo* or by giving a fitness penalty for a fixed period of time. This effectively means that the system skips similar messages for $\theta_r(m)$ clock ticks. The threshold method is quite simple, but it produces sufficient results for our purpose.

4.4.6 Preferences

The preference of messages can vary from one person to another, which is quite important to realize in the creation of a commentator. One easy way to implement user

preferences into a system is to parameterize the *importance* and *cognitive load* values as well as the message selection strategy. Although as the size of possibly recognized events grows, this may soon become infeasible to let the user do it manually.

Incorporating a method of recognizing user's interests and conveying them back to the system allows an automatic adaptation of parameters to each individual user. However, discussion of such system is beyond the scope of this thesis.

4.5 Output

The output layer is the final part of the commentator system and the problems in this section mainly concentrate on presentation techniques. This layer concerns mostly with psychology, but there are also some computer science related problems.

4.5.1 Natural language generation

One of the biggest problems in the output layer is the production of natural language. Although template-based, short sentences can be enough for some systems, humans can quickly grow weary of hearing or reading the same sentences over and over again. The field of natural language generation has a long history of research, which mostly deals with the creation of grammars.

Resorting to textual commentary is probably the most easiest approaches to implement. If, however, most of the unaltered information about world is based on visual cues, the need to switch focus between world and text can cause too much strain on the user. This problem can be overcome by transforming the text to audio via *text-to-speech* synthesizers.

4.5.2 Flavor generation

Flavor generation is one of the main functions of traditional television sports commentators (see Chapter 3). Because the viewers can actually perceive most of the events occurring in the competition as well as the commentator himself, a detailed explanation

of the basic events should not be necessary. The reason why commentators still vocally describe even petty details is that even if the viewer is looking at the game alone, he *feels* like he is watching it with an expert friend. The reason why international level games have more passion involved is that the *expert friend* is even on the same side as you are.

The output messages can be made to have more flavor by creating a set of more emotionally charged messages or by setting the bias to favor the side the user is supporting. The commentator can also be created so that it tries to avoid long periods of silence by producing meaningless talk or background information to fill the void.

One important factor affecting the flavor of a message is of course the medium used to present messages. For example, audiovisual presentations generally carry more emotional content than plain written text. However, these issues are outside of this thesis.

Chapter 5

Case Study of a Commentator System

Many of the ideas and issues presented in this thesis come from a project of creating an actual commentator system for RoboCup Soccer Simulator. Although it is a sports commentator, the aim is to produce an objective commentator instead of a recreational one. The project, called RoboComm, is used as a test environment for ideas and as a source for problems that needed to be solved. For a technical documentation of the implementation details of RoboComm, see [Sii04].

This chapter explains the higher level decisions and goes through some performance analysis. The RoboComm system is used as a basis of this thesis, because if offers a *proof-of-concept* of many presented ideas.

5.1 Input

The RoboComm system connects to the soccer server (see Section 1.2 for details) using protocol intended for the soccer monitor, a visualization tool for the soccer server. The *monitor communication protocol* provides information about the current play mode[1], the location of ball and each player and the current time.

[1] Possible play modes include: Play on, offside left-right, after goal left-right, time over, ...

RoboComm does only primitive preprocessing in order to remove outliers (see Section 4.2). If a player or ball is situated outside field, its position is replaced with the previous value. This amount of preprocessing seems to be enough, because the number of errors is usually quite small and as the data source is artificial the data is preprocessed at the server end.

5.2 Analysis

The analysis is done in an FFEN consisting of several recognizing nodes. There is also a blackboard available which can be used to store previous states such as the last ball controller and a successful pass. When a node has recognized an event, it is forwarded to the successor nodes in the network. In addition, a message is sent to the message selector.

5.2.1 Expert nodes

Currently the RoboComm system includes the following recognizing nodes:

- *Ball control* analyzer for recognizing the player controlling the ball
- *Ball kick* analyzer for recognizing the player kicking the ball
- *Pass* analyzer for recognizing passes between players, pass interceptions and failed attempts to score a goal
- *Goal attack* analyzer for recognizing the direction of attacks to the goal
- *Goal* analyzer for recognizing the scored goals
- *Medium term ball control* analyzer for analyzing the ball control percentages over time
- *Long term ball control* analyzer for analyzing the changes in medium term ball control

- *Play mode* analyzer for recognizing events such as offside, goal kick, etc.
- *Field control* analyzer for recognizing the focus areas on the field for the teams

Most of the expert nodes deal with short-term or even atomic events as they are the essential building blocks for more advanced analyzers.[2] Medium-term analyzers *goal attack* and *medium-term ball control* as well as long-term analyzers *long-term ball control* and *field control* utilize and refine the data in a flowing from the more primitive analyzers.

5.3 Message selection

The choice for a message selection strategy is not a straightforward task. A simple simulation (see Appendix A) can be used to examine what kind of impact different approaches have. Probabilistic models were discarded because they did not show enough improvement over the basic strategies. The greedy algorithm is chosen instead of dynamic one, because taboo branding and theme bonuses are incorporated to selection strategy. Implementing a dynamic selection strategy with these additional features would have been too complex in this case.

The amount of messages in the queue is kept small by the cutoff level and the number and distribution of important messages. The greedy algorithm is augmented with theme bonus (tb), although it seems that the system does not have enough connected events to fully realize the benefits of this refinement. The repeated message problem (see Section 4.4.5) becomes evident with the ball control analyzer and it is dealt with *taboo marking* strategy.

A simple fitness function $i * c * tb$ seems to provide an adequate method for separating the more descriptive messages from the less interesting ones, and because the cognitive load of message is highly correlated with importance, there is no need to include it in fitness. When a message type is branded taboo, the fitness value for all messages of

[2] Even many of the short-term analyzer nodes rely on other atomic and short-term analyzers.

same type is 0.

$$f(x) = \begin{cases} 0 & type(x) \in taboo \\ i(x) * c(x) * tb(x) & type(x) \notin taboo \end{cases}$$

5.4 Output

The output layer of RoboComm is simple: It prints one row for each selected message with the following structure. A row consists of a time stamp, 1–10 '!' characters indicating the importance of the message, 1–10 '*' characters indicating the confidence of the message, and the actual text. For example, a successful ball stealing might produce the following output.

```
001562 : !!         : ***         : Ball stolen by right 6
001575 : !          : **********  : Ball controlled by right 4
```

The first column of commentary rows above tells the viewer that the events took place at $t = 1546$ and $t = 1562$, and since the time unit is 100ms, it means the events occurred about 2.5 minutes after the initial kickoff. The messages are not very important as they are rated only 1/10 and 2/10, but the system seems to be quite sure that the latter event is true because it is given the confidence of 10/10.

5.5 Test results of RoboComm

The tests of RoboComm functionality were performed using four different teams that participated in RoboCup simulation league 2002: FC Portugal 2002, ATHumboldt 2002, United-2002 and Wright Eagle 2002, which are publicly available in the web [oA, Ber, oTaA, oSoC]. Each team played against all others four times and the output messages were logged. The tables and figures related to the output message analysis are found in Appendix B.

5.5.1 Analysis of output messages

The total amount of input messages for the RoboComm system was 7634 from the 24 game matches. In other words, approximately 318 messages per match were generated. The minimum number of messages was 251 (0.7 messages/s) and the maximum 371 (1.0 messages/s). Not all possible messages were output during matches, because many of the different play modes did not occur.

As we can see from Table B.1, there are two fundamentally different categories of messages. The first category consists of messages with absolute confidence, which are mainly play mode messages.[3] These messages are either an atomic notification from the server or they have no natural way of determining confidence. The second category of messages seems to form more or less skewed Gaussian distributions. This separation is even more obvious in Figure B.3. Almost half of the messages (3429 out of 7634) have confidence level of 1.0, while the rest of the messages have confidence level with mean of 0.32 and variance of 0.02. The number of messages with constant confidence value is high in the current system, because the higher level analysis modules depend on the atomic events.

The analysis on the distribution of importance levels does not give as much insight into the underlying process as the analysis of confidence level. This is because the importance values attached to the messages were handpicked. Nevertheless, figures B.1 and B.4 show that the current implementation of RoboComm outputs mostly messages commentating the normal flow of games. Over 60% of the messages (4739 out of 7634) reside in the importance interval $[0.00, 0.15)$.

The examination of confidence level against different importance levels (see Figure B.2) shows that there is no correlation between these two values. This observation is backed by that the correlation coefficient is only 0.05.

[3] As these are the decisions made by game referee, there is no need to consider their credibility.

5.5.2 Example of a recognition of an event sequence

In this section we give a more detailed look at a concrete sequence of events and how they are formed. This example is taken from a simulation run played between ATHumboldt and FC Portugal. It is a commentary of events ending in a goal by left team (FC Portugal). The output sequence is shown below.

```
000965 : !!!        : **          : Left 6 moves forward with ball
000982 : !!         : ***         : Ball kicked by left 6
000989 : !!         : **          : Ball stolen by right 7
000993 : !          : *******     : Ball controlled by left 6
001000 : !!!!!!!!!  : ********** : Left strikes from middle
001006 : !!!!!!!!!! : ********** : Goal by left B
001006 : !!!!!!!    : *           : Goal was set up by left 6
```

The initial state shown in Figure 5.1 is taken at $t = 947$. The important entities *Left B*, *Left 6*, *Right 7* and *Ball* are labeled for easier interpretation.

Left 6, who is controlling the ball, starts moving towards the goal while kicking the ball along. The *pass analyzer* notices that the receiver of these kicks is the same as the originator, *Left 6*, and as there are multiple such “passes”, the analyzer determines that player is moving forward with the ball.

About three seconds later at $t = 982$ *Left 6* is still moving forward, but as the taboo status of “move forward with ball” message is not yet lifted, the *ball kick analyzers* message about plain ball kick is outputted. This incidentally happens at the same time as *Left 6* makes a sharp turn right. At $t = 989$ *Right 7* gains a momentary control of the ball (see Figure 5.3) and this is duly noted by the commentator. *Left 6* takes back the ball only 0.4 seconds later at $t = 993$, but this is merely noted as “Ball controlled by *Left 6*” as the message about ball stealing is still taboo.

At $t = 1000$ a time-triggered event about the direction of attack is received by the message selection layer, and it is output because of its high fitness value. This output message prevents the message about pass from *Left 6* to *Left B* that occurs at $t =$

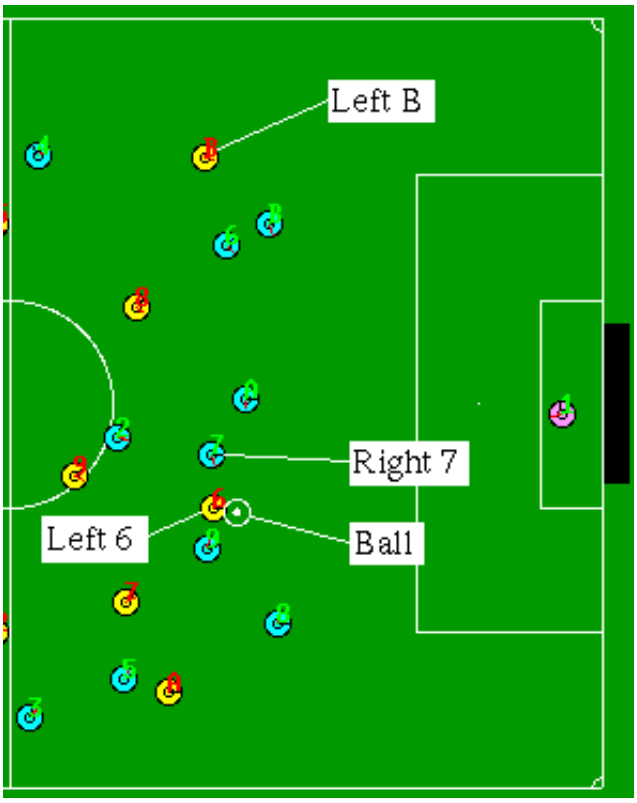

Figure 5.1. Initial state showing the important players and the ball.

1002, because the system is still waiting for the user to read and understand the previous message.

The goal occurs at $t = 1006$ and the system presents a message noting this, and shortly afterwards the system also gives credit to *Left 6* for setting up the goal as it made the pass leading to goal. The low confidence level for the final pass can be explained with the amount of players within the kicking range when the pass originated. The ball movement and game state 0.3 seconds before goal is shown in Figure 5.4.

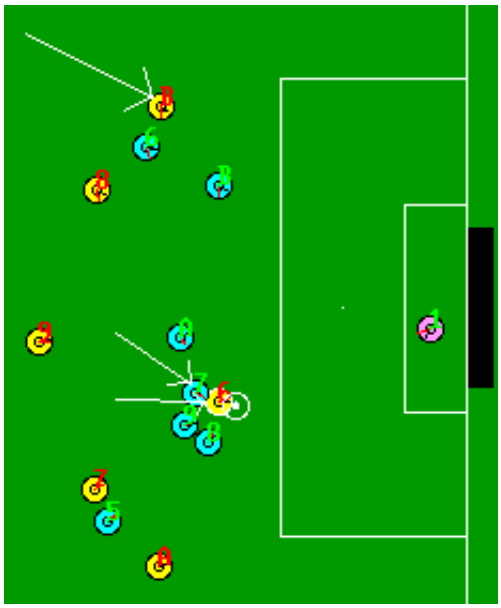

Figure 5.2. *Left 6* moves forward. Movement vectors for the players are added for clarity.

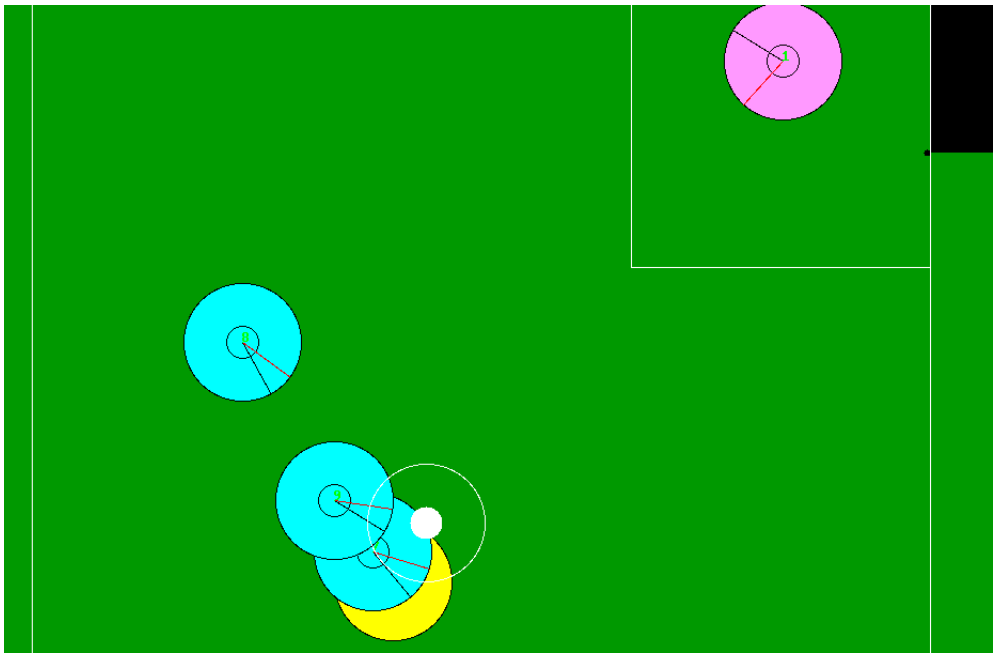

Figure 5.3. Ball is stolen momentarily by *Right 7*.

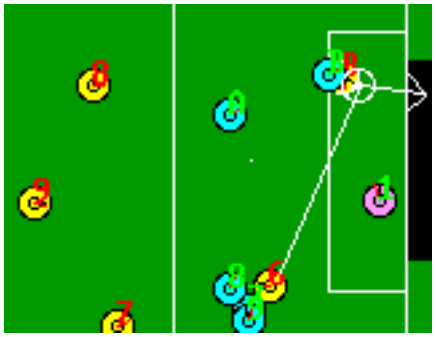

Figure 5.4. *Left 6* passes the ball to *Left B* who scores a goal. Line indicating the ball movement is added.

Chapter 6

Conclusions

Only a few articles about automatic commentator systems have been written. This work described a new and general structure for implementing commentator systems. The presented ideas were tested and verified by implementing the RoboComm system on the RoboCup Soccer Simulator.

The field of automatic commentator systems can be linked to various other research areas. The definitions presented for augmented reality and commentator systems enables to create a link between these two fields. This link provides commentator systems a with background to build upon and also a new approach for augmented reality research.

The FFEN structure for event analysis provides an implementor a flexible structure to build a collection of collaborating analyzers. It effectively pushes the complex parts of analysis inside each node making the reasoning about the overall structure easier.

Message selection generally assumes that the user's ability to interpret messages is less than the systems ability to produce them. If this was not the case, there would not be any need to filter out messages except those below a cutoff point. This makes the message selection effectively a resource optimization in an uncertain environment. However, selecting the best messages for output naively yields far worse results than utilizing a more intelligent method. The *theme-based selection* and *repeated messages problem* further highlight the special needs of message selection by showing that the commentary is more than just the sum of comments.

The optimal message selection strategy depends heavily on the type and distribution of incoming messages. Therefore classifying these different types of message streams would be beneficial in further studies about automatic commentators. Overall the topic of message selection seemed to be the most defining part of a commentator system, and, as such, the most interesting area for subsequent research.

References

[ABTI+00] Elisabeth André, Kim Binsted, Kumiko Tanaka-Ishii, Sean Luke, Gerd Herzog, and Thomas Rist. Three robocup simulation league commentator systems. *AI Magazine*, Spring 2000.

[Azu95] Ronald Azuma. *Predictive Tracking for Augmented Reality*. PhD thesis, UNC-Chapel Hill, 1995.

[Azu97] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[Ber] Humboldt University Berlin. Athumboldt. http://www.ki.informatik.hu-berlin.de/RoboCup/. Page visited on March, 2004.

[Ber98] T. Berlage. Augmented reality communication for diagnostic tasks in cardiology. *IEEE Transactions on Information Technology in Biomedicine*, 2:169–173, 1998.

[BGR+98] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleve. Understanding and constructing shared spaces with mixed-reality boundaries. *ACM Transactions on Computer-Human Interaction*, 5(3):185–223, 1998.

[Bro96] Frederik P. Jr. Brooks. The computer scientist as toolsmith ii. *Communications of the ACM*, 39(3):61–68, March 1996.

[Fed] The RoboCup Federation. Robocup. http://www.robocup.org/. Page visited on March, 2004.

[KSH03] Timo Kaukoranta, Jouni Smed, and Harri Hakonen. Role of pattern recognition in computer games. In Loo Wai Sing, Wan Hak Man, and Wong Wai, editors, *Proceedings of the 2nd International Conference on Application and Development of Computer Games*, pages 189–194, Jan 2003.

[KTS+97] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The robocup synthetic agent challenge,97. In *International Joint Conference on Artificial Intelligence (IJCAI97)*, 1997.

[Mei02] Peter Meijer. Seeing with sound for the blind: Is it vision? The Center for Consciousness Studies at The University of Arizona, April 2002.

[MTUK94] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. volume 2351 of *Telemanipulator and Telepresence Technologies*. SPIE, 1994.

[NMK96] H. Noma, T. Miyasato, and F. Kishino. A palmtop display for dextrous manipulation with haptic sensation. In *CHI'96*, pages 126–133. ATR Communication Systems Research Laboratories, April 1996.

[oA] IEETA/University of Aveiro. Fcportugal. http://www.ieeta.pt/robocup/. Page visited on March, 2004.

[oSoC] University of Science and Technology of China. Wrighteagle. http://wrighteagle.org/. Page visited on March, 2004.

[oT] Massachusetts Institute of Technology. Mithril construction and assembly. http://www.media.mit.edu/wearables/mithril/ hardware/index.html. Page visited on March, 2004.

[oTaA] University of Texas at Austin. United. http://www.cs.utexas.edu/users/pstone/RoboCup/United2002-sim.html. Page visited on March, 2004.

[PSLJ99] W. Pasman, A. van der Schaaf, R.L. Lagendijk, and F.W. Jansen. Low latency rendering for mobile augmented reality. In *Proceedings of the ASCI'99*, pages 372–376, June 1999.

[Rho96] Bradley James Rhodes. *Just-In-Time Information Retrieval*. PhD thesis, Massachusettes Institute of Technology, 1996.

[ROB] The robocup soccer simulator. http://sserver.sourceforge.net/. Page visited on March, 2004.

[SH02] Dieter Schmalstieg and Gerd Hesina. Distributed applications for collaborative augmented reality. IEEE Virtual Reality, pages 59–66, Orlando, Florida, March 2002.

[Sii04] Antti Siira. Robocomm - automatic commentator for robocup soccer server. Technical report, University of Turku, Department of Information Technology, 2004.

[SWS00] Dean Scribner, Penny Warren, and Jonathan Schuler. Extending color vision methods to bands beyond the visible. *Machine Vision and Applications*, 11:306–312, 2000.

[TCD+02] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, and Wayne Piekarski. First person indoor/outdoor augmented reality application: Arquake. *Personal and Uniquitous Computing*, (6):75–86, 2002.

[TK99] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 1999.

Appendix A

Selector Simulation

This appendix describes the test environment for measuring the properties of different message selection strategies. The results of this test are used to determine appropriate message selection strategy for the RoboComm system (see Chapter 5). Because the actual RoboComm system was not built at this phase, the source of test messages was not the actual RoboCup Soccer Server but instead an artificial random process.

A.1 Data generation and fitness-function

The importance value of each message is generated using random variable with exponential distribution parameterized by $\lambda = 10.0$. This should give a quite good estimate of real-life importance distribution, because there is usually numerous events with small importance, while the number decreases rapidly as importance increases. The importance is constant over a period of time 1–10 with uniform distribution between range 1–10. The confidence value is set to constant 1.0 as it does not play a crucial role in this experiment. The cognitive load is generated at random to the range 1–10 with a uniform distribution. The three values (importance, time frame, cognitive load) are not correlated in any way. A test sequence of 10,000 messages is recorded to be used with the message selection strategies.

The fitness function is simplified to return the importance value of the message, since

the confidence level is set to a constant value and the cognitive load is not used in RoboComm. The cutoff value is set to 0.1 for all selectors.

A.2 Test structure

The messages arrival rate is set to uniform 5 messages for a time tick. Other possibilities are random or periodic, but they would complicate the probabilistic algorithms somewhat.

The messages arriving at each tick are inserted to the message queue and the selector is activated. The selector can choose one or zero messages at a time, and if a message is selected the selector has to wait until time indicated by cognitive load value has passed before it can select another message. During this waiting time the messages continue to arrive. After the messages are exhausted the system keeps running the selector until the importance values of messages in the queue have dropped below cutoff line.

The probability of single message arriving with a time frame smaller than t is modeled with function $f(t) = 1-0.9^{t-1}$, which is multiplied with `avgMissed[t]` that represents the expected maximum of missed messages fitness in time t (see Table A.1). In *probabilistic greedy* and *probabilistic dynamic* message selection strategies, the value corresponding to the cognitive load of each message is subtracted from the fitness value of that message and the resulting value is used when comparing the relative fitness of messages. Furthermore, if the resulting value of the best available message is less than zero, the system abstains from outputting any messages, because the likelihood of missing an important message is too large.

1	2	3	4	5	6	7	8	9	10
0.0	0.29	0.33	0.36	0.38	0.40	0.42	0.43	0.44	0.45

Table A.1. Average maximum fitness of missed messages during cognitive load wait. The values are estimated from data of 100,000 messages generated separately of the test data of 10,000 messages.

Appendix B

Message Analysis

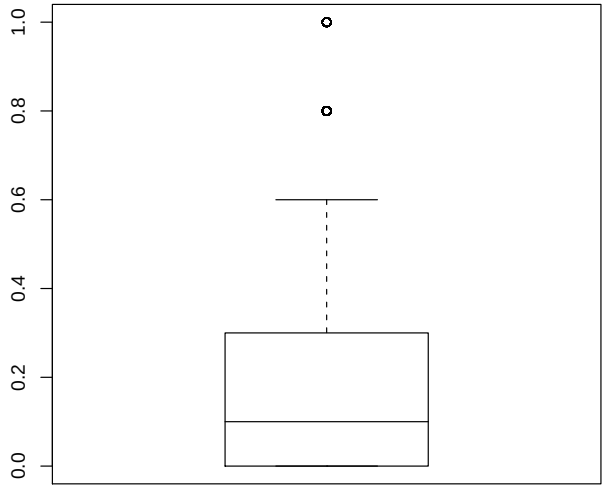

Figure B.1. Importance boxplot representing the distribution of importance values.

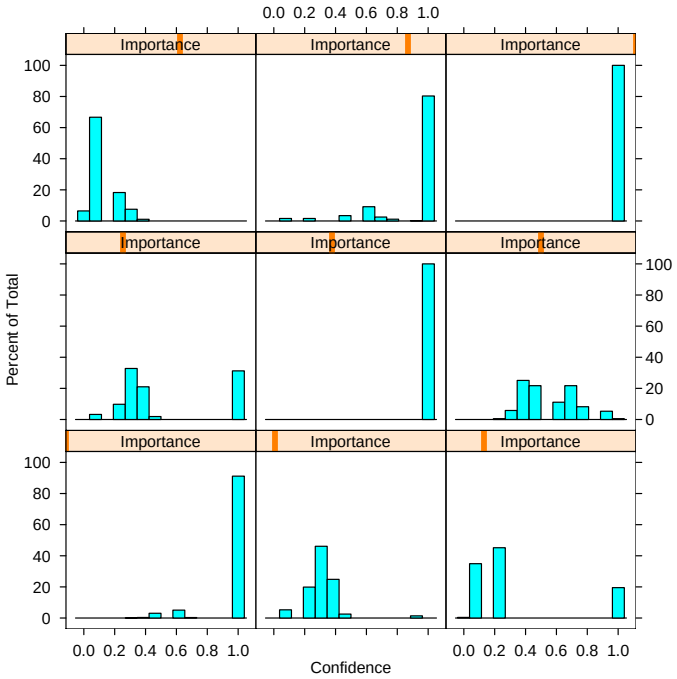

Figure B.2. Confidence-importance histogram. Messages are divided to 9 groups based on their importance value. The range of importance is equal for each group. Bars represent the percentage of messages with the given confidence value.

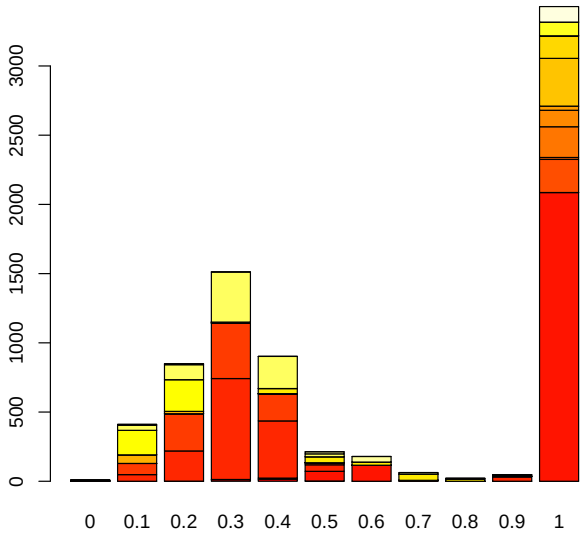

Figure B.3. Confidence level barplot. Bars represent the number of messages with the given confidence.

Message type	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
attack goal	0.00	0.00	0.03	0.40	0.50	0.07	0.00	0.00	0.00	0.00	0.00
ball control	0.00	0.00	0.00	0.00	0.00	0.03	0.05	0.00	0.00	0.00	0.91
ball kick	0.00	0.03	0.15	0.49	0.28	0.03	0.00	0.00	0.00	0.02	0.00
ball stolen	0.00	0.08	0.28	0.42	0.20	0.02	0.00	0.00	0.00	0.00	0.00
concentrating	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
corner kick	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
free kick	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
goal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
goal kick	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
goal setup	0.06	0.67	0.18	0.08	0.01	0.00	0.00	0.00	0.00	0.00	0.00
kick in	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
kickoff	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
medium ball control	0.00	0.00	0.00	0.00	0.21	0.24	0.13	0.25	0.10	0.06	0.01
moves forward	0.00	0.43	0.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
offside	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
pass	0.00	0.05	0.14	0.48	0.31	0.03	0.00	0.00	0.00	0.00	0.00
performance	0.00	0.08	0.08	0.00	0.00	0.17	0.47	0.13	0.06	0.01	0.00
strikes	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Table B.1. Message type versus confidence level. Numbers represent the proportion of messages with the given confidence level.

Message type	frequency	proportion
attack goal	30.00	0.00
ball control	2288.00	0.30
ball kick	1489.00	0.20
ball stolen	962.00	0.13
concentrating	239.00	0.03
corner kick	14.00	0.00
free kick	222.00	0.03
goal	119.00	0.02
goal kick	30.00	0.00
goal setup	93.00	0.01
kick in	345.00	0.05
kickoff	162.00	0.02
medium ball control	177.00	0.02
moves forward	408.00	0.05
offside	99.00	0.01
pass	759.00	0.10
performance	86.00	0.01
strikes	112.00	0.01

Table B.2. Message type frequencies and proportions

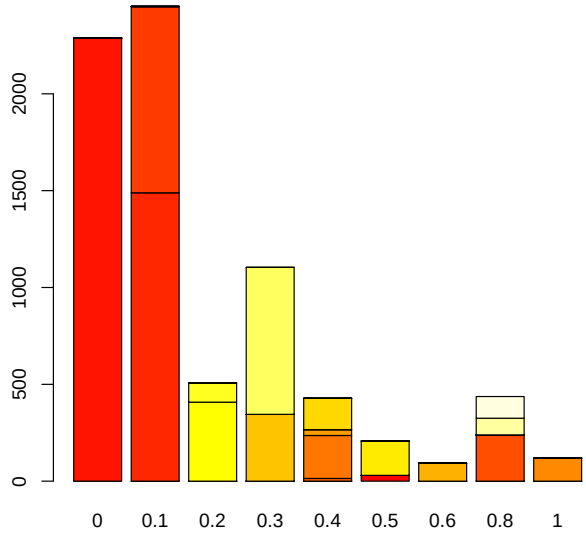

Figure B.4. Importance level barplot. Bars represent the number of messages with the given importance.