Collected papers on the PITA project

Papers of the Workshop on
"*Intelligent Routing and Travel Information Systems*"

Held on June 28th, 2001 at TRAIL Research School, Delft

Mediamatics / Knowledge Based Systems group &
TRAIL Research School, Delft / Rotterdam / Groningen, June 2001

Program

9:00 - 9:10 Opening and welcome by Prof. Dr. Ir. P.H.L. Bovy

9:10 - 9:20 Introduction by Dr. Drs. L.J.M. Rothkrantz

9:20 - 9:50 Ir. Robert van Vark - *"The Personal Intelligent Travel Assistant"*

The Personal Intelligent Travel Assistant (PITA) is a handheld device with multi-media capabilities providing ubiquitous communication between travelers and service providers at any time before or during a trip. The PITA is targeted at a large group of customers demanding different dialogue styles based on the individual preferences and experience of users. Due to varying traveler surroundings, like trains and private cars, the PITA is intended to use multiple modalities, like spoken language and text. Current dialogue systems are usually targeted at a small and experienced user group or they apply fixed and formal strategies. Additionally, current systems are mostly unimodal. Dialogue management in PITA needs to go beyond the current state of the art in the following three areas, due to the wide variety in task complexity, user preferences and traveler surroundings. Moving beyond fixed and formal dialogues involves adaptive dialogue strategies depending on the task complexity and history of the dialogue so far.

9:50 - 10:20 Chen-Ke Yang - *"Speech interfacing in the Wireless Automotive Messaging Pilot"*

The presentation gives an overview the SWAMP project done at CMG Trade Transport & Industry. This project is an extension of the Wireless Automobile Messaging (WAM) pilot project, in which wireless services in the field of traffic and transport have been developed. The SWAMP pilot incorporates a dialogue-based speech user interface on top of the existing WAM application, thus giving speech access to the developed services.

10:20 - 10:40 Break

10:40 - 11:10 Gerritjan Eggenkamp - *"Dynamic multimodal route planning"*

Although the highway network in the Netherlands is flooded with cars and is subject to heavy congestion, no planner is available that finds the optimal route given all congestions in the network. Also, no planner is available that incorporates different travel modalities using these dynamic data. A prototype of such a dynamic multimodal route planner has been developed and will be presented. The route planner uses artificial intelligence to find the optimal route in the highway network. This approach was compared with a traditional shortest path approach and the results show great potential. Both approaches will be discussed and a comparison will be made.

11:10 - 11:40 Ronald Kroon - *"Dynamic vehicle routing using an Ant Based Control algorithm"*

Recently, agents algorithms based on the natural behavior of ants were successfully applied to route data in telecommunication networks. Mobile agents created the routing tables in the simulated network with behaviors modeled on the trail-laying abilities of agents. We applied this Ant Based Control (ABC) algorithm to route cars in a city. The cars send their position to a central system and from that data the actual congestion is modeled. The agents find the shortest route in time using the actual spare capacity along the links. The shortest route is sent to the car driver via a PITA device. A prototype is under development to route car drivers to a parking lot.

11:40 - 12:10 Ir. Patrick Ehlert - *"Intelligent driving agents"*

Computer traffic simulation is important for making new traffic-control strategies. Microscopic traffic simulators can model traffic flow in a realistic manner and are ideal for agent-based vehicle control. We have made a model of an intelligent agent that is used to control a simulated vehicle and is equipped with different driving styles. The agent is implemented and tested in a prototype traffic simulator. The simulator consists of an urban environment with multi-lane roads, intersections, traffic lights, and vehicles. In the simulator every vehicle is controlled by a separate driving agent and all agents have individual behavior settings. Experiments have shown that the agents exhibit human-like behavior ranging from slow and careful to fast and aggressive driving behavior.

12:10 - 12:30 Summary and closure

Introduction

The Personal Intelligent Travel Assistant (PITA) is a handheld device with multi-media capabilities providing ubiquitous communication between travelers and service providers at any time before or during a trip. Providing detailed and personal information will change public transport systems drastically. From impersonal systems aimed at large customer groups, they will change into personal systems where the customer gets the impression that the services have been designed for him personally.
Task complexity, user preferences and traveler surroundings diverse drastically, calling for communicating by multiple modalities. At the moment, spoken language and text interfaces using SMS and WAP are foreseen. The latter are interfaces developed especially for mobile telephony.
Using the PITA, the traveler can perform the complex task of planning a travel schedule for a future trip using different transport modalities. In the public transport system envisioned by the "Seamless Multimodal Mobility (SMM)" program, these travel schedules will also be used to plan transport capacity. The PITA will guide the traveler by signaling upcoming transfer points and providing information on transfer routes in nodes. En route, the PITA will also provide the traveler with dynamic information about delays, calamities and alternative routes, if applicable.

During the last year, researchers from the Knowledge Based Systems group, headed by Prof. Dr. H. Koppelaar and supervised by Drs. Dr. L.J.M. Rothkrantz, invested much time and effort in PITA related research. Complementary to research activities going on in the TRAIL research school, the group focused on artificial intelligence techniques to solve PITA related problems.
The PITA-client-interaction is based on different modalities such as speech, text, touch etc. To handle this multimodal input and output devices a dialogue-management module has been developed. The multimodal input to the system is converted to a uniform XML code. The coded information is processed in the dialog-management module and the module is connected to different databases. To retrieve information from the WWW, the PITA system is able to launch intelligent search agents to find the requested information in an automated way.
Search agents also play an important role in dynamic vehicle routing and route planning. Different service providers offer information about traffic congestion and timetables of public transport via the WWW. These web pages are designed for human-computer interaction. The search agents have to interact with the same pages, but they have to extract the information with less visual support.
In another project we used intelligent agents to model human driving. This way, we designed a traffic simulator with agent car drivers that can interact with each other and simulate various human driving styles.
To summarize, in this workshop we highlight some artificial intelligence tools and techniques used in research activities in the framework of the PITA project. More specifically, we will present the use of expert systems in dynamic route planning and the use of intelligent agents in information retrieval.

Ir. P.A.M. Ehlert
Drs. Dr. L.J.M. Rothkrantz

Delft, June 2001

Table of Contents

Adaptive dialog management using multiple modalities

R.J. van Vark & L.J.M. Rothkrantz

Department of Knowledge Based Systems
Delft University of Technology, The Netherlands
{R.J.vanVark,L.J.M.Rothkrantz}@cs.tudelft.nl

Abstract

This paper describes the extension of the Alparon dialog manager to a broader domain and multiple modalities. To apply the dialog manager in the multimodal environment of the Personal Intelligent Travel Assistant (PITA), a multimodal coding scheme has been designed using dialog acts and XML. Additionally, the rigid database interface of the Alparon system has been replaced by a flexible and adaptive information retrieval system based on communicating agents.

1 Introduction

Over the last years, information retrieval systems have shifted from pure voice access to multimodal access, due to the huge increase in Internet and wireless communications. Automated speech processing systems are following this shift by including other modalities to retrieve the information [5].

Until recently, the Alparon project at the Delft University of Technology aimed at developing automated speech-processing systems. The focus within the project was on dialog management, developing dialog management strategies that mimic strategies found in human-human dialogs.

Currently, the Alparon dialog manager is being applied in the Personal Intelligent Travel Assistant (PITA). The PITA is a handheld device that provides ubiquitous communication using multiple modalities between travelers and public transport service providers, such as the Dutch railways and bus companies.

Migrating the Alparon dialog manager from a unimodal to a multimodal environment requires several modifications. This paper describes our initial approach to realizing these modifications by applying multimodal fusion as well as the approach currently applied. The latter approach consists of a multimodal dialog act coding scheme as well as an agent-based information retrieval system.

2 Personal Intelligent Travel Assistant

The PITA is a device by which travelers can retrieve information about public transport at any time before or during transport. This communication consists of travel planning and capacity reservation. Additionally, the PITA will guide the traveler by signaling upcoming transfer points and providing information on transfer routes in the nodes. En route, the Pita will also provide the traveler with information concerning delays, calamities, and alternatives routes if applicable. The PITA is a key component in the Seamless Multimodal Mobility research program at the Delft University of Technology.

The PITA provides multiple modalities to facilitate the communication between traveler and information system. These modalities can be described using several characteristics, such as interaction complexity and the environment in which the modality is best applied.

2.1 Automated spoken language

Spoken language interfaces can be used in virtually any environment, although adverse environments, such as driving a car at high speed, cause a significantly lower performance. Over the last decade, speech recognition has improved drastically making it applicable to a wide range of applications, among which are complex information retrieval tasks.

Despite the recent advances, automated spoken language processing still has several drawbacks. In contrast to the suggested unconstrained speech, speech recognizers apply suitably small vocabularies to prevent large numbers of recognition errors. In addition, extensive confirmation scenarios are still needed due to the occurring recognition errors and misinterpretations due to out-of-vocabulary errors.

Spoken language interfaces are especially suited to be used in more complex tasks that cannot be interfaced by using more straightforward text interfaces, because the amount of information would be overwhelming when using a (small) text interface. In the current PITA prototype, automated speech processing is applied to travel planning as it is the most complex task. In addition, travel planning can be performed before taking part in the actual transportation process.

2.2 Short Message Service (SMS)

SMS is an information service that can be found on many platforms for mobile telephony. It can be used to send text messages with a maximum size of 160 characters to other mobile subscribers or information servers. Human-human conversation based on SMS messages usually consists of free-formed text, like "Hi, let's meet at the conference desk at 8 p.m."

Using SMS messages consists of typing complex commands that have to be memorized by the user.

Although sending SMS messages back and forth can constitute a dialog, such dialogs are extremely time consuming. Therefore, SMS is best suited to straightforward initiation-response scenarios including short digressions when unclear or ambiguous information is found. In the PITA domain, SMS can best be applied to providing information on upcoming transfers and delays in transit, especially in crowded environments where social constraints might limit the use of spoken interfaces.

2.3 Wireless Application Protocol (WAP)

WAP combines the best of two rapidly evolving network technologies: mobile data and the Internet. Most of the technology developed for the Internet has been designed for desktop and large computers, medium to high bandwidth and reliable data networks. WAP is a protocol set making Internet-like information services available to handheld devices.

Compared to SMS, information services using WAP can be much more complex while the interface is still straightforward to use for an inexperienced user. However, a WAP dialog is heavily constrained and system driven. The user has to answer the questions asked by the system in order to gain access to the desired information. Diversions of the system scenario are hardly possible. The environmental constraints on WAP are approximately the same as for SMS.

3 Related work

The Alparon dialog manager on which the PITA is based applies plan-based theory where the basic assumption is that the linguistic behavior of agents in information dialogs is goal-directed [3]. To reach a particular state agents use a plan that is often a small variation of a standard scenario. The plan structure resembles the dialog structure. Dialog acts in PITA dialogs are also assumed to be part of a plan and the listener is assumed to respond appropriately to this plan and not only to isolated utterances [8].

Researchers within the Verbmobil project developed a taxonomy of dialog acts [1]. To model the task-oriented dialogs in a large corpus, they assumed that such dialogs could be modeled by means of a limited but open set of dialog acts.

Another approach of modeling dialogs is based on the observation that goal-directed dialogs consist of topical chains that are used to exchange information [2]. Topical chains are sequences of utterances that all communicate information about the same topic. Successive topics are usually related to each other. Topic transition in a dialog is modeled as movements across these topic packets. The relationship between topics is applied in the Alparon dialog manager.

4 The Alparon dialog manager

The dialog manager was originally designed for spoken language interaction. The system applies human-human strategies to structure interaction between user and information system. The focus within the project was on adaptive dialog management taking into account the task complexity and relative user experience with respect to both the task and technology. This problem was tackled by using multiple rule sets to isolate separate dialog phenomena, such as task structure, response generation, and user-interruption handling. These rule sets can be automatically updated to reflect task complexity and user preferences/behavior.

A modular design was chosen to implement a flexible system that is easily extended (see Figure 1). The Alparon dialog manager consists of modules for disambiguation, context updating, dialog updating, response generation, and dialog act generation. Besides these modules, the dialog manager uses a number of blackboards to store the information relevant to the ongoing dialog. Examples of such blackboards are dialog context, dialog history, and control information.

5 Multimodal fusion

Our first approach to extend the system with other modalities was to use a multimodal fusion technique developed for simultaneous multimodal interaction [6]. Multimodal fusion is a variation of the slot-filler method, a well-known method in artificial intelligence. Information necessary for command or language understanding is often encoded in structures called frames. A frame in its most abstract formulation is simply a cluster of facts and objects that describe some typical fact or situation. In Alparon frames contain information about information requests presented by multiple modalities. The principal objects in a frame are assigned names (slots), which can be viewed as functions taking values to instantiate the actual frame. Thus, a particular frame instance represents a structured set of knowledge.

The approach to multimodal fusion was to use a predefined set of frames corresponding to the possible requests in the current application. A slot writer-reader method that installs and retrieves slot values has been implemented as part of a fusion agent module. It uses the keyword approach combined with grammar constraints.

An important component of the fusion agent is the slot buffer. It stores the incoming values for all possible slots defined by the requested vocabulary. This is essential because presented information is often used in the future. Then, the parser fills the slots in the slot buffer that are designated in the utterance using the slot-writer method.

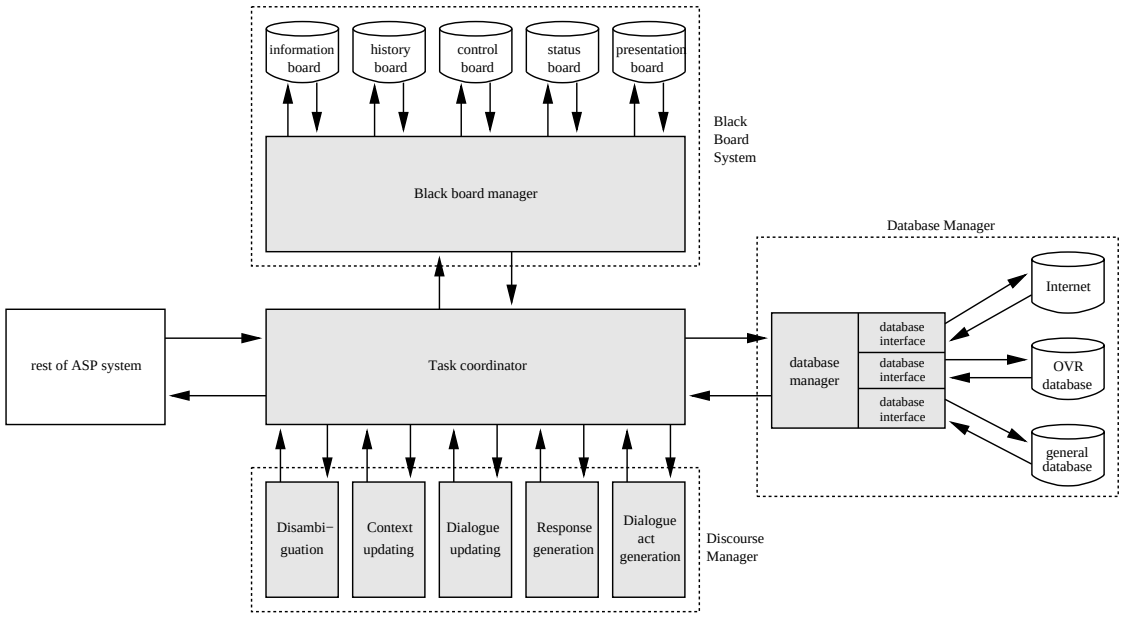

Figure 1 Architecture of the Alparon dialog manager

In this design, the fusion agent interfaces the dialog manager as soon as all the necessary information is available. The information may be provided entirely or partly by any modality.

The instantiation of a particular command frame is done by analyzing the information in the slot buffer. As long as the buffer is not filled, the system will wait for more input, which can be provided using any modality.

To achieve maximum flexibility, the design assures that a particular frame contains the minimum number of slots that are necessary to unambiguously interface the dialog manager.

Although multimodal fusion works well, there are some severe drawbacks. Most importantly, the fusion agent uses knowledge of the dialog state as the slot buffer is based on the expected response of the user. The slot buffer only allows responses fitting this expectation and even waits until a minimum number of slots is filled. This hardly fits the goal of the PITA to develop a flexible and adaptive dialog system as the user cannot divert from the prescribed dialog structure. Another drawback is the use of disambiguation knowledge in the fusion agent; this knowledge was previously applied by the dialog manager only.

Another approach was taken because of these drawbacks and the fact that simultaneous use of multiple modalities is not required in the PITA.

6 The PITA system

As mentioned above, the Alparon dialog manager has been specifically designed for spoken language input. When examining the dialog manager's design in detail, it is, however, apparent that the designers have always taken into account a possible inclusion of other modalities. As the dialog manager uses dialog acts to communicate with other natural language components in the system, most modules are modality independent. Only the disambiguation module (the first module), which takes care of disambiguating user input, and the dialog act generator (the final module), which takes care of translating the system response into suitable dialog acts, are dependent on the modality applied in the information service. Moreover, even their dependence on dialog modality is limited due to the use of dialog acts.

Therefore, our approach to develop the PITA using the flexible Alparon dialog manager consists of two aspects. First, a multimodal coding scheme has been designed to be able to represent different modalities in the dialog manager. Secondly, the rigid database manager in the Alparon system has been replaced by a flexible agent-based interface to access Internet content as well as traditional databases.

6.1 Multimodal coding scheme

The multimodal coding scheme currently applied in the PITA dialog manager is based on the Alparon coding scheme [8]. This coding scheme was mainly focused on analyzing human-human conversation. Consequently, several concepts have been introduced that make the coding scheme better suited for dialog management [9]. Additionally, the coding scheme adheres to the recommendations of the MATE-project, a European research project focusing on the annotation of communicative acts in dialogs [4].

Utterances are coded as comma-separated dialog acts that code the type of contribution made by the utterance, e.g. a request, feedback, etc. Dialog acts are performed in a certain dialog phase and by using one of several available modalities. The latter is important, as some modalities are more error-prone than others are. The informational content of an utterance is coded using a hierarchical tree of information codes.

The representation of the coding scheme has also been changed from a Prolog-like style using comma-separated dialog acts and a deeply nested coding of informational content. This Prolog-like style has been abandoned in favor of Extended Markup Language (XML). This has several advantages. First, XML is becoming a widely applied standard making the dialog manager portable to many environments. Secondly, many dedicated tools are available for dialog analysis, for example the MATE tool bench. Thirdly, standard XML style sheets can be used to translate Internet content and natural language into XML coded utterances and vice versa.

6.2 Agent-based information retrieval

In current information retrieval applications, relevant information parameters are extracted from the user-system interaction. Based on these parameters a query is defined and executed on a database. However, the PITA does not use one fixed information source to retrieve its information: it accesses many different information sources to retrieve the diverse information it distributes, of which many are found only on the Internet.

Using the Internet as information source, PITA has to interface distributed data sources and rapid changing data structures. When a user wants to travel from A to B using his car, train, or even a plane, travel information is needed from different websites. Information updates concerning delays and calamities are provided by yet other sources. These web sources constantly change with respect to content as well as structure. Therefore, intelligent search agents have been developed to find the requested information on the Internet.

Based on the request of the user, the dialog manager activates a search agent. This search agent has access to a specific knowledge base containing specific background knowledge for the current application. In PITA, this information consists of relevant concepts as arrival place, departure place, arrival and departure time, date, flight numbers, etc. A local database contains related instantiations of these concepts, such as city names. The agent does not have any knowledge of the specific structure of the web pages themselves.

Recently, we have developed a prototype that uses such search agents to find relevant travel information on multiple websites. With the relevant background knowledge the agent can start an interaction with the sites of the train companies in Germany, Norway, Italy, Czech Republic, Denmark, and The Netherlands. The agent is able to extract the structure of the sites by interacting with the websites, similar to the KISS method for organizations [7]. Once the structure has been determined, the agent retrieves the requested travel information using relevant input parameters. The dialog manager converts this information into utterances that are suitable for the communication medium used in the interaction.

7 Conclusion

The paper described our approach to extending the Alparon dialog manager with additional modalities. A multimodal coding scheme suitable for dialog management has been designed and implemented in XML. The dialog manager has been modified to be able to handle multimodal dialog acts by updating the respective rule sets for disambiguation and response generation. Additionally, the rigid database manager has been replaced by a flexible agent-based retrieval system to be able to provide the large diversity in information content needed in PITA.

In the near future, we hope to integrate the described prototypes into a fully functional system. This system will be used as a prototype to study user preferences in the PITA domain concerning interaction characteristics as well as preferences concerning domain information, especially at the moment delays and calamities occur in public transport.

8 References

[1] Alexandersson, J and Reithinger, N., Designing the dialog component in a speech translation system, A corpus based approach, In: *Workshop on Corpus-based Approaches to Dialogue Modeling,* Twente, The Netherlands, 35-43, 1995.

[2] Bunt, H.C., Dynamic interpretation and dialogue theory, In: *The structure of multimodal dialog,* John Benjamins Publishing Company, Amsterdam, 1995.

[3] Cohen, P.R., Models of Dialogue, *4th NEC Research Symposium*, NEC & SIAM, 181-204, 1994.

[4] Klein, M., An Overview of the State of the Art of Coding Schemes for Dialogue Act Annotation, In: *Text, Speech and Dialogue,* Lecture Notes in AI (1692), Springer Verlag, Berlin, 1999.

[5] Marsic, I., Medl, A., and Flanagan, F., Natural Communication with Information Systems, Proc. of the IEEE, 88(8), 1354-1366, 2000.

[6] Rothkrantz, L.J.M., and André, M., Redundancy and Ambiguity in Multimodal Human Computer Interaction, Euromedia'99, Munich, 1999.

[7] Ton, L. and Rothkrantz, L.J.M., Determining User Interface Semantics using Communicating Agents, TSD 2001, submitted.

[8] Vark, R.J. van, Vreught, J.P.M. de, and Rothkrantz, L.J.M., Classification of Public Transport Information Dialogues using an information Based Coding Scheme. In: *Dialogue Processing in Spoken Language Systems,* Lecture Notes in AI (1236), Springer Verlag, Berlin, 1997.

[9] Vark, R.J. van, Designing a Multimodal Coding Scheme for PITA, TRAIL 5th Annual Congress, 1-18, 1999.

Knowledge Based Speech Interfacing in the SWAMP Project

C.K.Yang L.J.M.Rothkrantz

Delft University of Technology, Zuidplantsoen 4, 2628 BZ Delft, the Netherlands

Abstract

Speech technology is rapidly developing and has improved a lot over the last few years. Nevertheless, speech-enabled applications have not yet become mainstream software. Furthermore, there is a lack of proven design methods and methodologies specifically concerning speech applications. So far the application of speech technology has only been a limited success. This Paper describes a project done at CMG Trade Transport & Industry BV. It is called SWAMP and is an example of the application of speech technology in human-computer interaction. The reasoning model behind the speech interface is based on the Belief Desire Intention (BDI) model for rational agents. Other important tools that were used to build the speech user interface are the Microsoft Speech API 5 and CLIPS.

1. Introduction

Speech is the most common mode of communication between people. Although speech communication is not a perfect process, we are able to understand each other with a very high success rate. Research has shown that the use of speech enhances the quality of communication between humans, as reflected in shorter problem solving times and general user satisfaction [1]. Furthermore, speaking to humans subjectively seems to be a relatively effortless task [2]. The benefits mentioned above are some reasons that have moved researchers to study speech interaction systems between humans and computers.

In September 1999 CMG Trade, Transport & Industry BV started the Wireless Automotive Messaging (WAM) project. Its purpose was to develop new wireless services in the field of traffic and transport. The WAM application is based on the Client-Server model. The server is stationary while the client travels with the user in his car. Because the clients are mobile, communication is based on wireless techniques.

This paper discusses the SWAMP[1] project started in October of the following year. The purpose of the SWAMP project was to analyse if a speech interface is better suited for the WAM pilot. Therefore the WAM client is extended with a speech interface: the SWAMP client. This offers a way for the driver to interact with the system while his hands and eyes remain free, ideal for car driving situations.

[1] SWAMP is an acronym for Speech Interfacing in the Wireless Automotive Messaging Pilot

2. The SWAMP client

Speech interaction between the user and the SWAMP application is based on dialogues. Generally, the user starts a speech interaction by indicating (via speech) what his desires are. The system then leads the user through a dialogue in which it tries to retrieve information regarding these desires. If eventually all the necessary information is collected, the application takes the appropriate actions to realise the user's desires.

The general assumption behind the speech interface is that the user wants to accomplish something with his utterances, i.e. he has a certain goal in mind. The set of all services the SWAMP application has to offer is just a subset of all the goals the user can possibly have. Goals that don't correspond to a service, however are beyond the domain of the speech interface and are ignored.

The speech interface is divided into 3 components.

1 The speech recognition or ASR component:
  Its function is to recognise the user's utterance and transform it into a format that can be processed.
2 The dialogue management component:
  Its function is to process the input from the speech recognition component to figure out what the user wanted to accomplish and take the appropriate actions to realise the user's wishes. This component is the main focus in this paper.
3 The speech synthesis or TTS component:
  Its function is to generate speech output to the user.

Figure 2-1 gives a graphical overview of how the speech interface is implemented. The main application is the original WAM client modified in such a way that it can communicate with the dialogue manager.

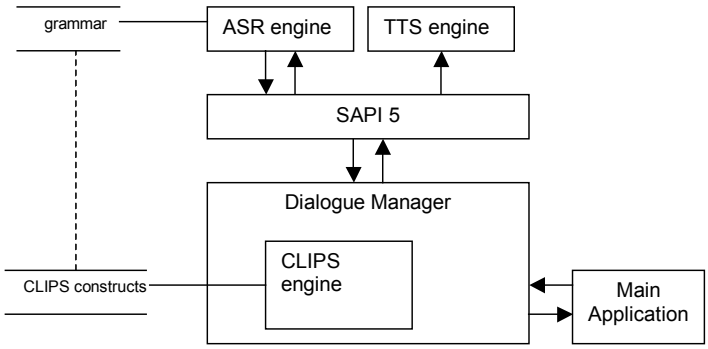

Figure 2-1 Overview of SWAMP implementation

The SWAMP client is implemented in C++ (Microsoft Visual C++ 6.0 enterprise edition). Initially it was the intention to build the speech interface to run on Windows CE but due to limitations in software and hardware of handheld computers, Windows NT was ultimately chosen.

The Microsoft Speech Application Programming Interface 5.0 (SAPI5) is used as middle-ware between the engines and the SWAMP application. SAPI5 acts as a communication layer between the dialogue manager and the speech resources (ASR and TTS engine). It takes care of hardware specific issues such as audio device management and removes implementation details such as multi-threading. This reduces the amount of code overhead required for an application to use speech recognition and synthesis. Another advantage of using middle-ware is that the choice of the final ASR and TTS engine can be postponed till a later stadium (e.g. until there is more budget for better engines).

The CLIPS expert system tool is designed to facilitate the development of software to model human knowledge or expertise. CLIPS is embedded in the SWAMP client. It can be viewed as the knowledge processing and management unit of the dialogue manager.

3. Dialogue design

With each initial utterance from the user, the speech interface tries to find the corresponding service involved. Once the goals of the user are clear it tries to accomplish the service by checking whether all the information needed is available. If this is not the case, the speech interface must initiate a dialogue to retrieve the required information from the user until the task can be performed. All possible dialogues that the speech interface can be involved in must be designed beforehand. This includes speech prompts for each situation, and all possible user responses on those prompts. Furthermore design involves the definition of a grammar that captures the syntax of whole conversations into a few simple grammar rules.

3.1 Design approach

The goal of the speech interface is to give a user access to the SWAMP services by means of simple speech interaction. To achieve this, one can choose between two different approaches: 1) demand a longer learning time for the speech interface and require the user to adapt his speaking style or 2) make it easy for the user by allowing an extensive grammar and modelling more and more complex dialogues so that the user can speak to the system as with another human.

Speech User Interface (SUI) designers have learned that humans are extraordinarily flexible in their speech and readily adapt to the speaking style of their conversational partners. This is not a new finding: think about how easily we adjust our speech depending on whether we are speaking to children or other adults. This flexibility has useful implications for designing the speech interface: after extensive use of the speech interface (as the user gets acquainted with the grammar and has more experience) some dialogues become less and less common. Since the user will adapt his style of interacting and refrain to only those dialogues that were successful in the past. Because of this finding and the choice of our typical user (“he is familiar with current computer technology”) the first approach was chosen: only model the most common utterances and let the user adapt to it.

3.2 Dialogue representation

Without a proper representation technique, the dialogues can quickly become very complex and unmanageable. In this project dialogues are represented by flow diagrams containing nodes representing start/begin points of a dialogue, boxes representing actions (e.g. an utterance from a user or an action from the system), diamonds representing decisions point and arcs to connect the nodes, boxes and diamonds. A dialogue always begins with a start node and ends with an end node. Within these nodes, the dialogue travels from box to box along the arcs and branching at the decision diamonds. A successful dialogue corresponds to a path in the flow diagram from the start node to the end node.

Speech dialogues are context sensitive. In our representation, the context is defined by the positions within the dialogue flow. Each box represents a certain state or context. The arcs branching from a box indicate the options available within that context and the branches leading to a box define how that context can be achieved.

The power of above dialogue representation technique lies in the fact that dialogues are represented in a generic way. E.g. the (user action) boxes define what the user can say at that moment in the dialogue, but not how it must be said (this is defined in the grammar). In this way, a single path in the dialogue flow diagram can represent whole categories of similar dialogues.

A well-modelled dialogue flow diagram is one where each possible dialogue flow can fit in. This implicates that common communication errors, such as misunderstandings, should be modelled as well as mechanisms for correcting and preventing these errors, such as requests for confirmation and roll back. Table 1 shows an example dialogue for the kilometre registration (KM registration) service. The flow of this dialogue fits into the flow diagram in Figure 3-1 (accentuated).

In practice the dialogues can become so complex and the dialogue flow diagrams so large that it is best to split them up into one main dialogue and several smaller sub dialogues. For each sub dialogue a separate dialogue flow diagram is designed and referred to in the main dialogue flow diagram (by means of sub dialogue nodes). Another use for the dialogue flow diagrams occurs during the testing phase. Since each path from the start node to the end node corresponds to a successful dialogue. The correctness of the implementation of the dialogues can easily be verified if all the paths in the dialogue flow diagrams can be traversed.

Table 1: Example dialogue

U: Change trip type S: Is it a business or a private trip? U: It's a business trip? S: OK, what's the project ID for this business trip? U: Project ID is SWAMP S: Do you want to set the project ID to SWAMP? U: Yes S: OK, trip type is set.

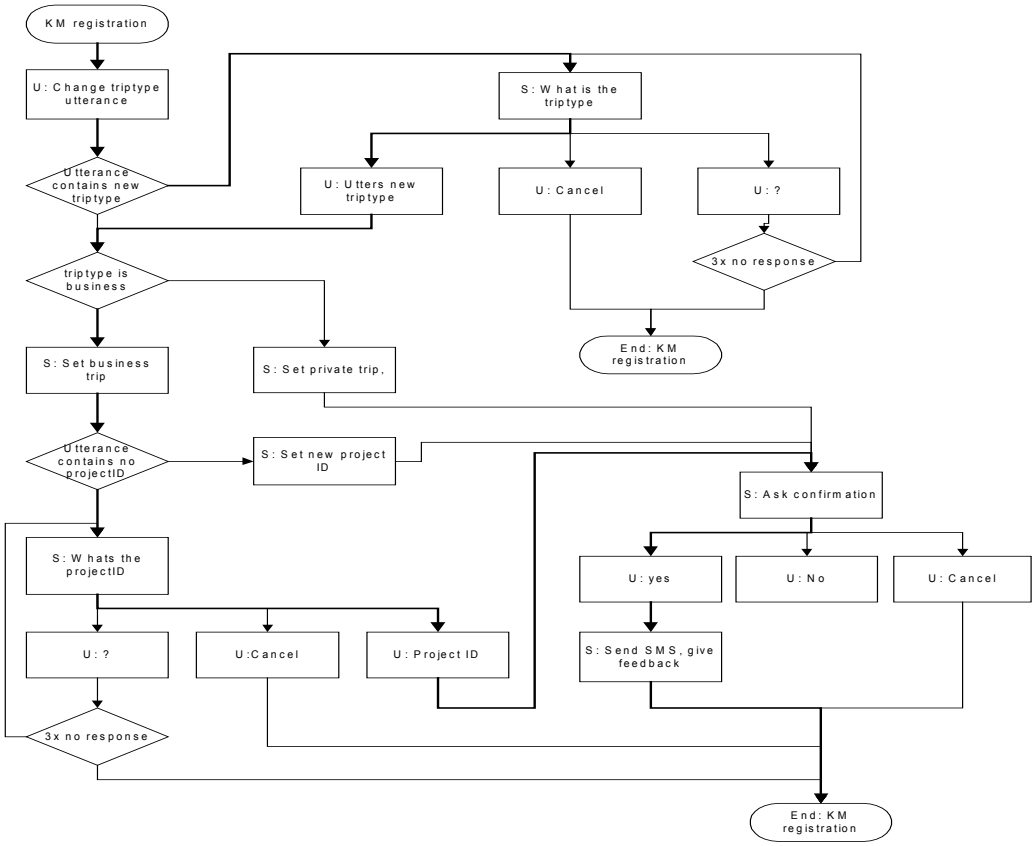

Figure 3-1: Dialogue flow diagram for the KM registration service

3.3 Grammar

The SAPI5 design specification requires the grammar of an application must be a context-free grammar (CFG) written in a format specified in the SAPI5 grammar schema. This schema describes the SAPI 5.0 speech recognition grammar format and is based on the XML framework. The ASR engine uses the CFG to constrain the words contained in the user's utterance that it will recognise.

Basically the grammar file consists of a set of grammar rules in the grammar schema syntax. The complete specification of the schema can be found in the SAPI5 online help. Grammar rules can have an activation state, which can be set to active of inactive. SAPI5 recognises active rules and conversely does not recognise deactivated ones. The application may change the state of the rules during execution. So if a rule is no longer needed, it may be deactivated.

In order to indicate the functional parts of a sentence i.e. the parts that actually contain relevant information, the CFG can be extended with semantic information declared inside the grammar. This enables the ASR engine to associate certain recognised word strings with name/value-meaning representations. The dialogue

manager then applies these meaning representation associations to understand and control the dialogue with the user.

The grammar rules are derived from a corpus of utterances by hand. Crucial in this process is the determination where the relevant information is located within an utterance. Once this is accomplished, the derivation process is straightforward.

4. The reasoning model

In the search for a suitable reasoning model for the dialogue manager: one that is capable of adequately describing the reasoning behaviour of the dialogue manager, the Belief-Desire-Intention (BDI) model [3] was chosen. In an implementation of a dialogue manager according to this model, the dialogue manager continuously executes a cycle of observing the world and updating its beliefs, deciding what intention to achieve next, determining a plan of some kind to achieve this intention, and then executing the plan.

There exist a correspondence between concrete CLIPS data structures and the attitudes in the BDI model. Beliefs in the BDI model are implemented as facts and rules in CLIPS. Facts are used to construct the dialogue manager's internal representation of the world. Facts can be seen as propositions and thus can only consist of a set of literals without disjunction or implication. Therefore special rules (belief rules) are used to complete the representation of beliefs. Belief rules represent the general relationship between facts (e.g. IF utterance=help THEN AlertLevel=high).

One way of modelling the behaviour of BDI reasoning [4] is with a branching tree structure, where each branch in the tree represents an alternative execution path. Each node in the structure represents a certain state of the world, and each transition a primitive action made by the system, a primitive event occurring in the environment or both. In this formal model, one can identify the desires of the system with particular paths through the tree structure. The above description of the branching tree structure is logically similar to the structure of the dialogue flow diagrams described in section 3.2. In fact, both structures represent exactly the same: a path through the dialogue flow diagram is a successful dialogue, which is also a desire and therefore a path through the branching tree of the BDI reasoning model. As a result, the dialogue flows diagrams can be treated as the structures that describe the behaviour of the dialogue manager. They are directly implemented in CLIPS rules, each rule corresponds to a branch in the dialogue flow. Rules are both the means for achieving certain desires and the options available for the dialogue manager. Each rule has a body describing the primitive sub goals that have to be achieved for rule execution to be successful. The conditions under which a rule can be chosen as an option are specified by an invocation condition. The set of rules that make up a path through the dialogue flow, correspond to a desire.

The set of rules with satisfied invocation conditions at a time T (the set of instantiated rules) correspond to the intentions of the dialogue manager at time T. Obviously the intentions of the system are time dependent. The dialogue manager adopts a single-minded commitment strategy, which allows continuous changes to beliefs and drops its intentions accordingly. In other words the intentions of the system can be affected by the utterances of the user in contrast to blind commitment in which a intention is always executed no matter changes in beliefs.

5. An example

In the previous section it was shown that the desires of the dialogue manager can be represented by dialogue flow diagrams. The flow diagrams are systematically translated into an executable system formulated in CLIPS rules. This section discusses the implementation of the desires. In particular the heuristics used for the translation from dialogue flow diagrams to CLIPS rules.

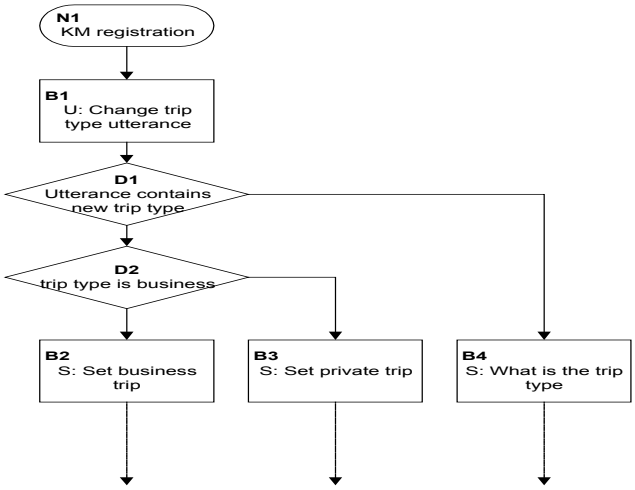

Figure 5-1: Part of the dialogue flow diagram for KM registration service

Suppose we must transform a dialogue flow diagram as in Figure 5-1. This dialogue is initialised when the user utters a phrase that matches the grammar for a change trip type utterance (box **B1**). Notice that box **B1** has 3 branches (to the boxes **B2**, **B3** and **B4**), furthermore we see that the action in **B1** is a speech action from the user. From this we conclude that the dialogue flow should be implemented using 3 speech rules. The invocation conditions for each rule are the evaluated values of the expressions in the decision diamonds **D1** and **D2**. The body of each rule contains the actions specified in the corresponding destination boxes. Furthermore, the body of the rules also contain actions to anticipate what follows after the action e.g. after box **B4** the user must supply the new trip type so the grammar rules for trip type utterances should be activated.

The CLIPS rule in Figure 5-2 corresponds to the branch from box **B1** to **B2** (Figure 5-1). The keyword RECOGNISED in line 2 indicates that a user's utterance is recognised. The grammar rule that matched the utterance is VID_KMREG_TRIPTYPE. Furthermore, the property name TripType with value business satisfies condition **D1** and **D2**. The actions taken satisfy **B2** (between the <SAY> tags in line 5) and anticipate future utterances of the user by activating the VID_YES_NO grammar rule and de-activating all other main grammar rules. Thereby limiting the user's input to only boolean values. The other actions in the rule body are used to update the internal representation of the world.

```
1    (defrule KM_Registration_Business
2       ?in<-(RECOGNISED 161 VID_KMREG_TRIPTYPE 50 TripType business)
3       ?pos<-(POSITION MAIN RUNNING)
4       =>
5       (printout t "<SAY>Do you want set the triptype to business?</SAY>
6                   <ACT>VID_YESNO</ACT>
7                   <DEACT>"?*Mainrules*"</DEACT>
8                   <REACT></REACT>" crlf)
9       (retract ?in)
10      (retract ?pos)
11      (assert (POSITION MAIN KMREG))
12      (assert (WANT CONFIRM))
13      (assert (QUESTION KMREG business))
     )
```

Figure 5-2: CLIPS rule - part of the KM registration service

6. Conclusions

The model presented here allows for man-machine speech interaction. Indeed the speech interface of the SWAMP application implemented according to this model is capable of handling simple dialogues with the user. The dialogues are described and visualised as generic flow diagrams resembling branching tree structures [4]. The chosen representation technique has also contributed greatly to the containment of the complexity in the dialogues models. Furthermore it allows an easy translation to executable CLIPS rules.

References

[1] A. Chapanis, "*Interactive Human Communication: Some lessons learned from laboratory experiments"*, In: Shackel, B. (eds). "*Man-Computer Interaction: Human Factor Aspects of Computers and people*", Rockville, MD: Sijthoff and Noordhoff, page. 65, 1981.

[2] H. Nusbaum, et al., "*Using Speech recognition systems: Issues in cognitive Engineering*", In: Syrdal A. et al. (eds), "*Applied Speech Technology*", Boca Raton, CRC press, page. 127, 1995.

[3] M. Wooldridge, "*Reasoning about Rational Agents*", The MIT Press, Cambridge, Massachusetts, 2000.

[4] A. Rao and M. Georgeff, "BDI Agents: From Theory to Practice", *in Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95),* 1995.

Intelligent dynamic route planning

G. Eggenkamp L.J.M. Rothkrantz

Delft University of Technology, Zuidplantsoen 4, 2628BZ Delft, the Netherlands

Abstract

In this paper the possibilities of artificial intelligence and especially of expert systems in the field of route planning using dynamic traffic data are explored. An expert system that has been built to perform dynamic routing and a dynamic route planner using a (traditional) shortest path algorithm are introduced. Using both implementations a comparison is made between the expert system approach and the shortest path approach. It is concluded that the expert system shows great potential. It outperforms the shortest path algorithm in computation time and the routes the expert system finds are indeed the shortest routes.

1. Introduction

Currently no dynamic route planners are available. Although the highway network in the Netherlands is flooded with cars and is subject to heavy congestion in both rush hours no planner is available that finds the shortest route in this (congested) network. The only option some route planners and car navigation systems offer is to 'block' roads that are congested and to find the best alternative route without using this road. Consequently a route is advised that might well take longer then the route along the congested road, since this road is not even considered anymore. Since dynamic data are available from the MONICA monitoring system (the detection loops under the highways) a study has been carried out to develop such a dynamic route planner.

When research was carried out to the performance of shortest path algorithms, like Dijkstra's algorithm, to find the shortest route while using dynamic data, it showed that the computation time degrades significantly when dynamic data are incorporated. Consequently, other possibilities were investigated and since humans are quite well capable of finding alternative routes in the case of congestion it was decided to study the feasibility of an artificial intelligence approach. In this paper the feasibility of an expert system in a dynamic route planner is discussed and a comparison is made with a 'regular' shortest path algorithm.

This paper will start with a problem definition (section 2) and an introduction of a shortest path algorithm that can be used to find the shortest path in a dynamic network (section 3). In section 4 an introduction of the expert system that was constructed is given, while in section 5 the results of both approaches are presented. In section 6 some conclusions are given.

2. Problem definition

The highway network can be represented by a graph, as with static routing problems. The highway network that was considered in the research is the network that is being monitored by the MONICA system, since only dynamic data are available of these roads. In Figure 1 this network is shown. Somehow the dynamic aspect of the data has to be taken into account. The travel times between different edges (cities, junctions) change in time, and these changes have to be taken into account and incorporated in the graph. When, for example, travelling from Amsterdam to Delft in the morning rush hour, a departure of only 5 minutes later, can affect the travel time by more then 20 minutes, since major congestion may have occurred along the route during these 5 minutes. For example an accident might have happened or a sudden peak in cars that want to access the highway may have occurred.

A space time extended network (STEN) explicitly represents time by having a complete layer of all nodes of the physical network per time period. The first occurrence of STEN in literature can be found in [4]. Other applications of space-time expanded networks can be found in [1,3,6]. In all these publications time expanded networks were used to solve traffic assignment models, which are dynamic flow problems. In dynamic route planning only the shortest path has to be found, no dynamic flow problem has to be solved. Consequently, the same approach can be used, only with a flow of 1 for all links.

Concordant to these publication the space-time expanded network can be constructed as follows:

- for each period p create a complete layer of all nodes of the physical network,
- for each node in period p (all nodes with the same t), create links to the nodes it is connected with in the physical network in the corresponding period 'layer' p + d, with d the travel time when starting at period p,
- for each node in period p create a link to the same node in period p + 1 (it is also possible to stop in a node).

An example of a network constructed this way is shown in Figure 2. The original graph consisting of nodes A to G is repeated for each time interval. The edges between the nodes A to G (the lowest graph at t = 10:01) represent which nodes are connected to each other. For clarity these edges have been kept in the different layers of the graph to show these connections. The thicker links that intersect the different layers are the actual road connections. Their length (and thus the layer to which they go) represents the travel time when starting at the time of the layer in which they start. For each layer for all nodes all outgoing links are constructed and labelled according to the travel time at that moment. It should be noticed that in Figure 2 not all the links are shown, since that would have resulted in a cluttered figure.

3. The extended Dijkstra algorithm

The most secure way to find an optimal route from an origin to a destination at a specific time of the day would be to find the optimal route in the graph that was constructed in the previous section. In [4] a proof is given that a dynamic routing problem that is expanded in the way described can be solved using static shortest path algorithms.

In practice the graph that is proposed in section 2 better can not be constructed, since this would require a lot of computation time. It would be far more efficient if travel times only were estimated if they are really needed. Consequently an algorithm was constructed that finds the shortest path in this 3-dimensional graph and which only estimates travel times if necessary. When the algorithm was constructed and was reviewed thoroughly, it was discovered it differs with Dijkstra's algorithm only slightly. Consequently, it was called the 'extended Dijkstra algorithm'. Since the Dijkstra algorithm is widely known, no explanation is given here. It can be found in [3].

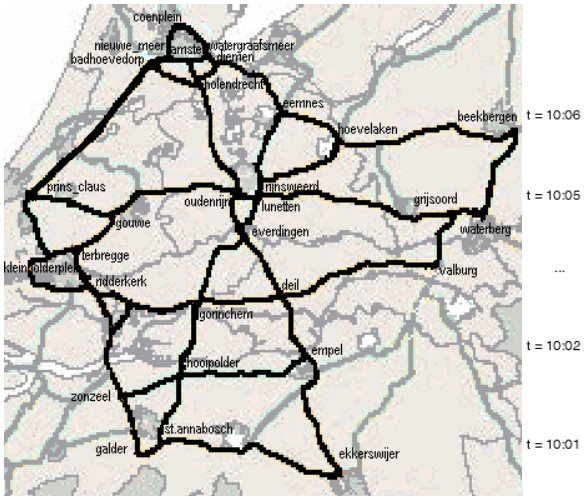

Figure 1. The freeway network that is monitored by the MONICA system.

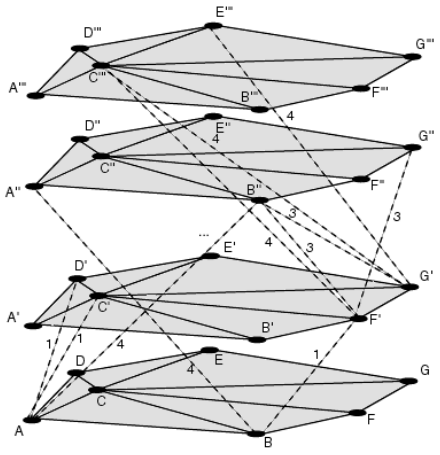

Figure 2. A space-time expanded network.

It should be noticed that no research was done into the estimation of travel times, which is a very complex process. During this project we focused on one main aspect: route planning. Consequently, it was assumed travel time estimates were available and a set of historical data was used as 'dummy' data.

4. Expert system

In this section the expert system that has been constructed is introduced. As was stated in section 2 the problem domain of the expert system is given by the road network that is given in Figure 1. The expert system should find the fastest route in this network.

4.1 Knowledge elicitation

The knowledge the expert system should possess consists of alternative routes in the case of congestion along a part of a road. This knowledge can be made explicit in two ways. Firstly, experts can be interviewed. These experts should be experienced 'traffic jam travelers' that often have tried alternative routes in the case of congestion along a part of the freeway they normally use. Secondly, the map of the Netherlands combined with historical traffic data can be investigated, to see which alternatives are reasonable in the

case of a traffic jam. In this project, the second approach was chosen. The reason to do this was as follows. Travelers are not able to monitor the routes they did not choose. When they have chosen an alternative, afterwards they do not know if it was faster then the original route or other alternatives (unless they know another traveler, who tried the alternative at the same time). Consequently, the perception the traveler has of the quality of alternatives he tried can be wrong, since it may also be influenced by other incentives then the shortest travel time.

4.2 Level of detail

For the level of detail in which congestion in the road network is monitored route sections between junctions where one can change freeways were chosen. Since routes are only optimised in the freeway network (and not considering secondary roads), only at junctions the route can be changed. Since secondary roads are not taken into account, it is not interesting to construct rules on the basis of congestion between two ramps: for all ramps that are between two junctions, the same rules would be constructed, since only at the first junction after the ramp it is possible to change the route. In Figure 1 the different road parts between junctions can be found (the junctions are identified by their names).

4.3 Route representation

The expert system is provided with a number of route parts (trajectories), that each have a predicate, which can be 'route', 'file', 'entrance' or 'exit'. These route parts are delimited by two junctions. An example of such a route can be found in Figure 4. The route on the map was generated by a static route planner and was translated to a route for the expert system. Right of the figure the translation of the route can be found.

(entrance coenplein nieuwe_meer)
(route nieuwe_meer badhoevedorp)
(route badhoevedorp burgerveen)
(route burgerveen prins_claus)
(route prins_claus ypenburg)
(exit ypenburg kleinpolderplein)

Figure 4. The route from Amsterdam to Delft and the format that is given to the expert system.

It was chosen to keep the choice whether there is congestion along a trajectory, outside the domain of the expert system. A separate module was constructed in which it is decided if the delay is significant enough to consider the trajectory congested. This module provides the expert system the route with its appropriate predicates.

4.4 Construction of the rule base

For each trajectory along the road network rules were made stating which alternative to take if the trajectory was congested. Since there are 92 edges in the network that is

monitored by the MONICA system (Figure 2) the construction of these rules was a time consuming task. The alternative route that can be taken when a route part is congested depends on the direction one is coming from and the direction one is going to: the rules for alternative roads depend on the previous and following route parts. The different steps, which are needed to construct the different rules for each trajectory are given in the following action list.

1. Determine the possible directions where one can be coming from
2. Determine the possible directions where one can be going to
3. Determine the different alternative routes for each possible route, by investigating the map and historical data.
4. Calculate the 'detour time' of each alternative: the time needed to make the detour in the best case (no congestion along the alternative route).
5. Order the different routes according to this 'detour-time'.
6. When the 'detour-time' is too large, do not use the alternative.
7. Check with reports of car drivers if no routes are missing

The first two steps are very straightforward, the possible directions can be found by having a look at the map. The third step requires by far the most time: in this step the different alternative routes have to be chosen. When these routes are known, their travel times can be computed. The fifth step is important to find the best route as quickly as possible. This property can be very useful if the dynamic route planner is used in a real-time environment and there is a time constraint. When alternatives become available very fast, while searching is continued for better alternatives, the best route found so far can be used if the time constraint has to be met. Of course, it would be optimal if the first route found also is the best route and this is examined using this parameter. Consequently, the order in which the alternatives are searched should depend on the travel times and the chance of congestion along these alternatives, when there is congestion along the trajectory for which alternatives are searched.

5. Results

To be able to compare both methods the following four parameters were chosen: 1) the number of travel time estimates, 2) overall computation time, 3) shortest route found and 4) order of found routes.

The number of travel time estimates parameter was chosen since it gives an indication of the performance of the algorithm. Since the travel time estimation is the process that needs the most computational time the number of travel time estimates will strongly indicate the total computational time needed.

The overall computation time parameter is included to be able to judge the performance of the expert system. It could be possible, the expert system approach needs only few travel time estimates, but is very slow itself, since the rule base is very large.

The third variable on which the methods will be compared is the shortest route that is found. Since it is mathematically proven that the extended Dijkstra algorithm will return the shortest route this parameter is only applicable to the expert system.

The last variable which will be carefully examined is the order in which alternative routes are given. This aspect is also only applicable to the expert system, since Dijkstra's

algorithm does not return any other route then the best one. It should be examined how many alternatives have to be computed to find the shortest one. As was stated in section 4.4 this can be important in a real-time environment.

5.1 Testing protocol

To test both approaches several departure and destination addresses were chosen between which the shortest route had to be found. Firstly the routes were searched on a free-flow network, without congestion. Congestion was created along one of the trajectories in the shortest route that was found and both algorithms were applied again. Again congestion was created along one of the trajectories of the newly found route and both algorithm were applied. This process was repeated until all trajectories were delayed. In Figures 5 and 6 the first two iterations of this process are illustrated. In Figure 5 the free flow route that was found is shown. Congestion was created between the Prins Claus and Badhoevedorp junctions and both algorithm were applied. The shortest route found now is shown in Figure 6. Now congestion was created between the Holendrecht and Diemen junctions and the same process was repeated. In Table 2 an overview of the results of the first three steps of this testing procedure for the route between Zoetermeer and Muiden is given.

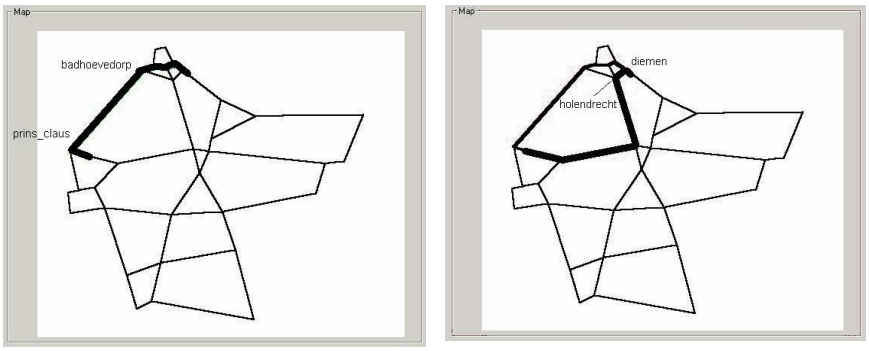

Figure 5 and 6. Free flow route and route if there is congestion between the Prins Claus and Badhoevedorp junctions.

Table 2. Test results of route between Zoetermeer and Muiden.

	exp. syst. trav. est.	exp. syst. comp.time	graph alg. trav. est.	graph alg. comp.time	order found	routes are the same
Original route, A12-A4-A10-A1 (48')	0	9500	722	17470	0/0	Yes
Delay between prins claus and badhoevedorp (A4), +20',A12-A2-A9-A1 (53')	93	10000	1286	34270	1/3	Yes
Delay between holendrecht and diemen (A9), +15', A12-A2-A10-A1 (54')	94	10490	1285	34880	3/3	Yes

5.2 Results

In Table 3 the results of this testing procedure for different routes are shown. In the most left column the routes are denoted together with the number of iterations with congested

trajectories that was carried out. Columns 2 to 5 show the average number of travel time estimations and the average computation time of both methods. In column 6 some kind of indication is given which alternative of the expert system was the best route. In Table 2 this was indicated as x/6 for each route, which means that 6 alternatives were generated and the first one of these was the fastest one. The indication that is given in Table 3 is simply the sum of the total routes found (right number) and the sum of the numbers that indicate when the route was found (left number). A value of 4/13 indicates that overall measured the fourth alternative was the right one, given thirteen routes. The last column indicates the number of correct routes out of the number of total routes found.

Table 3. Average values of testing procedures.

	expert syst. travtime est.	expert syst. comp. time	graph alg. travtime est.	graph alg. comp.time	order found	same routes
Muiden-Amerongen (6)	96	9403	1268	33958	4/13	5/6*
Amerongen-Delft (7)	142	10511	1570	46656	8/15	7/7
Amsterdam-Apeldoorn (7)	203	8457	1436	36744	13/42	7/7
Deventer-Gouda (8)	130	10011	1015	32034	12/29	7/8*
Weesp-Moordrecht (10)	466	14664	1302	33238	21/65	9/10*
Total average	**229**	**10899**	**1310**	**36216**		

* The routes were different, although the travel time was the same.

Table 3 shows the expert system requires significantly less computation time then the shortest path algorithm. The overall computation time is a factor 3.5 less, while the difference of the number of travel time estimates is almost a factor 6. Since it is expected the estimation of the travel time will take (by far) the most computation time in a real-time estimation it can be expected the expert system will perform even better when used together with MONICA data: the data the detection loops generate have to be combined with historical data using some kind of prediction algorithm, which will take a lot more computation time then currently was needed, since dummy data were used.
In Table 3 it can be seen that the few times the routes were different (three times out of approx. 60 routes), the travel times were the same. Since different algorithms were used to find the shortest route both approaches returned a different one, although the other alternatives were also returned. As a result it can be stated that the quality of the routes found by the expert system is very good. On the other hand it should also be remarked that the testing procedure influenced the results a little bit. During the testing procedure no scenario's were tried to frustrate the expert system. It would be possible to create such congestion along all reasonable alternatives that a very strange alternative would become the best one. When for example travelling from Amsterdam to Utrecht one could create severe congestion along all 'normal' alternative roads such that one would have to travel via Apeldoorn and Arnhem, back to Utrecht to have the fastest route. Of course such situations are very rare in reality.
With respect to the order in which the expert system generates alternative routes it can be remarked the results are quite well. Most of the time one of the first routes that is generated actually is the fastest route. On the other hand it can be noticed that sometimes the best route is one of the last routes found. This is a consequence of the unpredictable behaviour of congestion. As was stated in section 4.4 it was tried to rank the different alternative routes in such a way that the alternative with the highest chance of being the

best one was tried first. Since a chance guarantees nothing sometimes other routes are better. Especially when more then one trajectory is congested along a route the best alternative can be one of the last ones tried. Two or more trajectories are congested, so two or more file predicates will instantiate different rules. The order in which these rules fire cannot be regulated in a way the rule with the 'best' alternative fires first, since more then one trajectory is congested and it can not be stated beforehand which alternative will be most promising in that case. Consequently the alternative that are fired by one rule might all be tried first after which the second rule fires which contains the best alternative.
The last remark that can be made is concerned with the implementation. In section 4 it was stated the travel times of each alternative should be computed to prevent the travel times of alternatives being computed that will not make a chance since their detour time is larger then the delay due to the congestion. Since the construction of the rule base took much more time then expected this implementation was not made. Subsequently sometimes alternatives were tried that should not be tried at all.

6. Conclusion

In this paper the possibilities of an expert system in the field of dynamic route planning were discussed and a comparison was made between a shortest path algorithm and the expert system. The expert system showed great potential. Not only performs the expert system much better with respect to computation time, the routes the expert system returns are as good as the routes the conventional shortest path algorithm computes and the expert system shows great possibilities when real time constraints are placed. The expert first generates all possible solutions and then computes their travel time one by one. As soon as the travel time of a solution has been computed the solution becomes available.

The most important drawback of the expert system approach is the construction of the rules. This is a very intensive process and requires a lot of time.

References

[1] H.K. Chen. *Dynamic travel choice models: a variational inequality approach*. Springer, Heidelberg, 1999.

[2] G. Eggenkamp. *KRIS: Knowledge based routing information system*. Graduation thesis, TU Delft, 2001.

[3] J.R. Evans and E. Minieka. *Optimization algorithms for networks and graphs.* Marcel Dekker Inc., New York, 2nd edition, 1991.

[4] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, New Yersey, 1962.

[5] B. Ran and D.E. Boyce. *Modeling dynamic transportation networks: an intelligent transportation system oriented approach.* Springer-Verlag, Berlin, 2nd edition, 1996.

Dynamic vehicle routing using an ABC-algorithm

R. Kroon, L.J.M. Rothkrantz *

Abstract

In the past years the application of agent algorithms based on the natural behaviour of ants have shown to be successful in routing data through communication networks. Using the trail-laying abilities of ants the mobile agents are able to create well performing routing tables. In this paper an Ant Based Control algorithm is applied to the routing of road traffic trough a city. The algorithm is tested in a simulation environment that makes it possible to show the effect in different cities and circumstances. The agents do not move through a real city, but use a model of a city map. This model is supplemented with actual data from the traffic in the city. This enables the agents to divert traffic from congested routes, which improves travelling-times.

Key words: *Routing algorithms, mobile agents, ant-based algorithms, distributed routing*

1. Introduction

Road traffic is getting busier and busier each year. Everyone is familiar with traffic congestion on highways and in the city. And everyone will admit that it is a problem that affects us both economically as well as mentally. Furthermore finding your way in an unknown city can be very difficult even with a map. Navigation systems like CARiN can help in such cases. These systems display the route to be followed when the user has entered his destination. The latest versions are also able to use congestion information to avoid trouble spots. But such information is only available for highways and not in a city.

This paper addresses the dynamic routing of traffic in a city. We want to set up a routing system for motor vehicles that guides them through the city using the shortest way in time, taking into account the load on the roads. Furthermore we want the routing system to be distributed, for more robustness and load distribution.

The routing system uses a routing algorithm based on earlier versions of Ant Based Control-algorithms. Exact routing algorithms like Dijkstra's algorithm only apply to central routing. And ant-based algorithms have proven to be superior to other distributed routing algorithms in [1,2]. In [2] an ant-based algorithm was used for routing and load balancing in a telephony network. In [3] the algorithm is applied to packet switched networks with basic ideas taken from [1]. And now we will apply a variant of the algorithm to a traffic network in a city.

2. Theory

This section presents a short introduction to an important aspect of the behaviour of ants and the basic ideas of ant based control.

2.1 Emergent behaviour of ants

Insects like ants perceive only a very local piece of the world they live in. But it is no coincidence that they do find their way back to a food source or their nest. When a group of such animals interact they can exhibit a higher-level behaviour. This behaviour is most often called emergent behaviour. Instead of a central controlling authority ants interact with their environment to achieve common goals. The resulting behaviour of an ant colony can be very complex.

The emergent behaviour is enabled by stigmergy. Stigmergy is a way for entities to communicate indirectly with each other through the environment. An ant colony uses this for finding the shortest route from their nest to a food source and back. Ants only react to local stimuli from their environment, but they can change some of those local stimuli. Such a modification will influence future actions of other ants at that location.

The ants lay pheromone, a kind of hormone, as a mutual signalling system. When looking for food, the ants follow the pheromone trails with a probability proportional to the strength of the trail. They do not necessary

* R. Kroon, L.J.M Rothkrantz
Delft University of Technology, Knowledge Based Systems, Mekelweg 4, 2628 CD Delft,
The Netherlands, E-mail: L.J.M.Rothkrantz@cs.tudelft.nl

follow the track with strongest pheromone trail. There will often be a certain amount of error (or noise). The strength of the trail sensed by an ant depends on the original strength and the time elapsed since the pheromone was laid. This is because the pheromone diffuses in time. Several ants can travel the same route, resulting in a pheromone trail laid by different ants at different times. The pheromone trail sensed by ants is therefore a composite one. The probability that an ant chooses a particular route depends on the concentration of the pheromones. A stronger pheromone trail increases the chance an ant chooses the route belonging to that pheromone trail. This mechanism makes the ants bias towards the shortest paths. It works for the following three reasons:

- Shorter routes will be completed earlier than longer routes and thus attract other ants *earlier*.
- The ant density will be bigger at shorter routes when the ants choose the alternatives equally likely, and more ants produce *more* pheromone.
- Ants travelling shorter routes will arrive earlier. This causes the pheromone to be *stronger*, because less of the pheromone trail has diffused.

This strengthening process may continue until there is no more pheromone on the longer paths.

2.2 Ant-based control for network management

We can use the idea of emergent behaviour of natural ants to build routing tables in any network. We will apply it in a traffic network in a city, i.e. the composition of the roads and their intersections. This network is represented by a directed graph. Each node in the graph corresponds to an intersection. The links between them are the roads. Mobile agents, whose behaviour is modelled on the trail-laying abilities of natural ants, replace the ants. The agents move across the network between randomly chosen pairs of nodes. As they move, pheromone is deposited as a function of the time of their journey. That time is influenced by the congestion encountered on their journey. They select their path at each intermediate node according to the distribution of the simulated pheromone at each node. Each node in the network has a probability table for every possible final destination. The tables have entries for each neighbouring node that can be reached via one connecting link. The probabilities influence the agent's selection of the next node in their journey to the destination node. The probability of the agents choosing a certain next node is the same as the probability in the table.

The probability tables only contain local information and no global information on the best routes. Each time an agent visits a node the next step in the route is determined. This process is repeated until the agent reaches its destination. Thus, the entire route from a source node to a destination node is not determined beforehand.

Agents are launched at each node with regular time intervals with a random destination node. They travel around the network using the probabilities in the probability tables. The probabilities per destination are all filled with equal values for all nodes before the process begins.

3. Design

This section explains the design of the routing system.

3.1 Dynamic data

We want to route the traffic dynamically through a city. Therefore we need dynamic data about the state of the traffic in the city. This can be gathered from sensors in the road-surface. Such sensors can count vehicles and measure the speed of the vehicles. That information can be used to compute the time it takes to cover a part of the road. Another source can be the traffic information services. They can inform the system about congestion, diversions of the road, roadblocks and perhaps open bridges. And finally the vehicles themselves can provide the system with information about the path they followed and the time it took them to cover it. The current technology enables us to fix the position of a vehicle with an accuracy of a few meters. That position can be communicated to the system along with the covered route.

For our routing system we will at first only use the latter type of information as dynamic data. But of course the model is open for additional types of dynamic data. The information from the vehicles is handled by a separate part of the routing system, called the timetable updating system. This subsystem takes care that the information is processed for use by the ant-based algorithm. This way one vehicle drives a certain route and sends its performance to the routing system. Another vehicle is able to use that information to choose the shortest route.

3.2 Architecture

We will now explain the structure of the system from the viewpoint of the vehicle and its driver. A vehicle is driving through a city and it wants to know the way. The driver enters the address where he wants to go and expects a routing system to tell him where to go. Besides the destination the routing system needs to know the location where the vehicle is at the moment. Therefore the vehicle sends a request to a satellite of the GPS (Global Positioning System). This is shown by arrow A in figure 1. GPS is a system that can determine a position of the sender with an accuracy of a few meters. So the GPS-satellite answers the vehicle with its current position (arrow B). This position is measured in latitude/longitude co-ordinates. In the vehicle these co-ordinates are translated in a position on a certain road with the aid of a digital map of the city. Now the vehicle has enough information to request the routing system what route to follow. The vehicle sends its position and its desired destination along with the request for the route to the routing system (arrow D). Arrow E is the answer from the routing system that contains the route that the vehicle should follow. These steps are pretty obvious, but we have skipped arrow C. This arrow indicates that the vehicle provides the routing system with information about the route it has followed since the previous time. The information consists of (1) the location and time at the moment of the previous update, (2) the location and time at this moment and (3) the route that the vehicle has followed in between these times and locations. Table 1 shows a detailed enumeration of the information that is send along the indicated arrows.

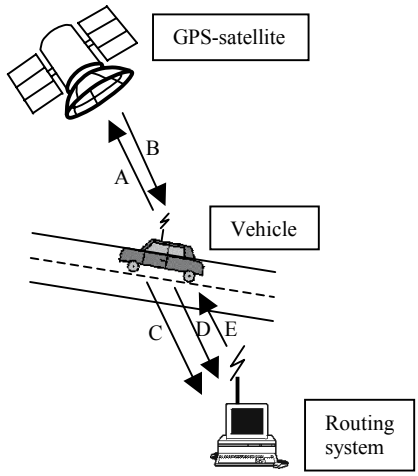

Fig. 1 *Communication of the vehicle*

Table 1 Communicated data between the different objects

Arrow	From	To	Data
A	Vehicle	GPS-satellite	REQUEST_POSITION
B	GPS-satellite	Vehicle	ANSWER_POSITION, latitude/longitude co-ordinates
C	Vehicle	Routing system	UPDATE, previous time/position, covered road A, covered road B, covered road C, …, current time/position
D	Vehicle	Routing system	REQUEST_ROUTE, current position, destination
E	Routing system	Vehicle	ANSWER_ROUTE, road A, road B, road C, …

3.2.1 Communication in time

Now we will give an impression of how the communication works in time. Firstly, a REQUEST_POSITION (arrow A) will be send with some regular time interval, for example every minute. Directly after that an ANSWER_POSITION (arrow B) will be sent back to the vehicle. The information that is sent with an UPDATE (arrow C) is most valuable as soon as a new position is known. The fact is that the UPDATE must be sent together with a current position, and the longer it is delayed the older the data is. Therefore an UPDATE will always be sent directly after an ANSWER_POSITION. This does not alter the fact that an UPDATE does not have to be sent after every ANSWER_POSITION. This frequency can for example be lower to reduce communication. Finally a REQUEST_ROUTE (arrow D) and an ANSWER_ROUTE (arrow E) will also always succeed an ANSWER_POSITION. When a new route is requested and acquired, the old route is overwritten. Clearly a REQUEST_ROUTE needs a current position. An old value for the position could cause a vehicle to be travelling on a road that is no longer in the list of its new route. The interval by which a REQUEST_ROUTE is sent can differ from the former messages. After the first route is acquired it will mostly be valid for the rest of the travelling-time. So the frequency can be lower than the frequency of the other messages. Figure 2 clarifies again which messages succeed each other.

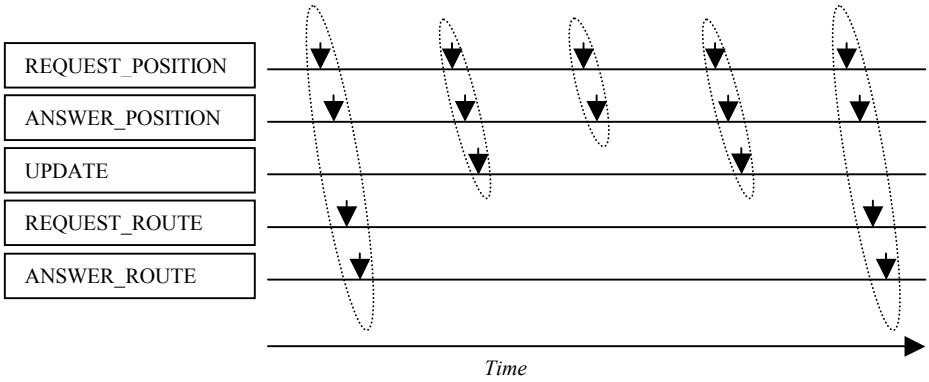

Fig. 2 *Succeeding messages*

3.2.2 Distributed routing system

The use of the ABC-algorithm allows the routing system to be distributed. This means that the computation is done on several computer systems that are mutually connected via a network. Distribution of computational power gives some advantages above a central routing system. Firstly what normally has to be done by one computer system is now done by several computer systems, which increases the speed and the memory space. Secondly, when properly implemented the failure of one of the systems does not have to imply a total break down of the routing system. This does however involve some extra communication necessary for information that is not available on the concerning computer system. It will eventually depend on the amount of information that needs to be communicated between the computer systems whether the actual speed will be higher than with a central routing system. Another possible disadvantage is that the traditional routing algorithms cannot be used. Those routing algorithms allow for perfect routing, i.e. giving the best routes possible. The ABC-algorithm only approaches the best routes. But the results of this research will have to show that the ABC-algorithm is sufficiently accurate to route the traffic.

3.3 Routing problem

The most important problem of this research is solved by the *timetable updating system* and the *route finding system*. These two subsystems together form the *routing system*. The relations are shown in figure 3. The function of the *route finding system* will be clear: we are building a system to route vehicles. The reason why we need the *timetable updating system* is the following. The *route finding system* needs information about the state of the network. A static route finding system could use a fixed set of data, but we will use a dynamic route finding system that needs dynamic data. Those data are provided by the *timetable updating system*. That information can be for example the load of the parts of the network but a more direct and therefore more practical type of information is the time it takes to cover a road. Vehicles send information about their covered route to the *timetable updating system*. From that information this system computes the travelling-times for all roads and stores it in the timetable in the *memory*. Besides the timetable also a history of measurements is stored in the memory. The reason to keep a history will become clear later in section 3.3.1. The *route finding system* uses the information in the timetable to compute the shortest routes for the vehicles. When a vehicle requests route information, the *route finding system* sends this information back to the vehicle.

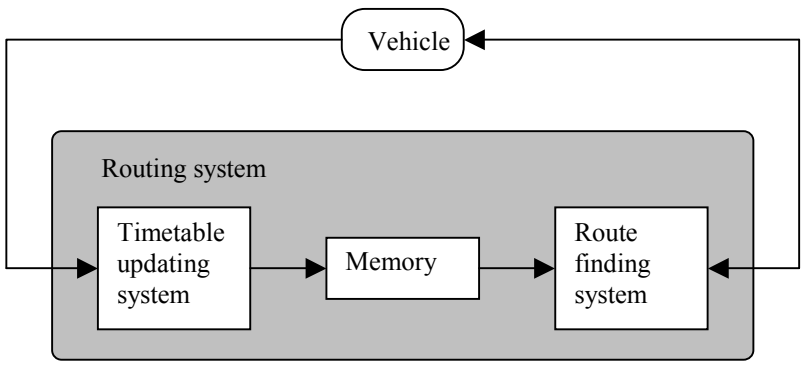

Fig. 3 *Design of the routing system*

3.3.1 Timetable updating system

This subsystem could receive its information about the traffic network in the city from different sources. This could for example be directly from sensors in the road-surface. But the main sources are the vehicles themselves. They provide the system with information about the path they followed and the time it took them to cover it. With this information the timetable updating system computes the travelling-times for every part of the road. The travelling-times are placed in the timetable. This timetable can be seen as a two-dimensional matrix with all intersections of the traffic network along both axes. When one can go from one intersection directly to another, there will be an entry in the table that represents an estimate for the time to cover that road. The intersections that cannot reach eachother unless via another intersection will have no entry in the table. Figure 4 is an example of a traffic network and table 2 shows its timetable.

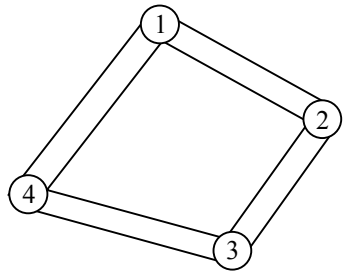

Fig. 4 *A simple traffic network*

Table 2 *The timetable matching figure 4*

From: \ To:	Intersection 1	Intersection 2	Intersection 3	Intersection 4
Intersection 1		25 sec		33 sec
Intersection 2	24 sec		18 sec	
Intersection 3		18 sec		20 sec
Intersection 4	31 sec		23 sec	

We will now explain how we represent a traffic network in our model. Figure 5 shows a simplified part of a city map. Figure 6 shows the internal representation of that map. As one can see there is a forward and a backward link for every part of the road. This represents that the traffic can move in both directions. Furthermore one can notice that at every point where a driver must choose between more than one road, the road is divided into separate links.

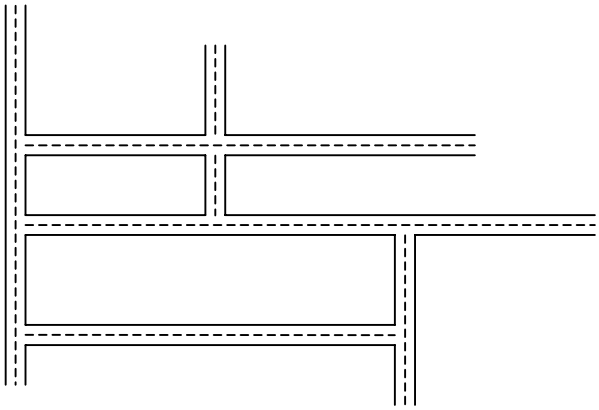

Fig. 5 *A part of a city network*

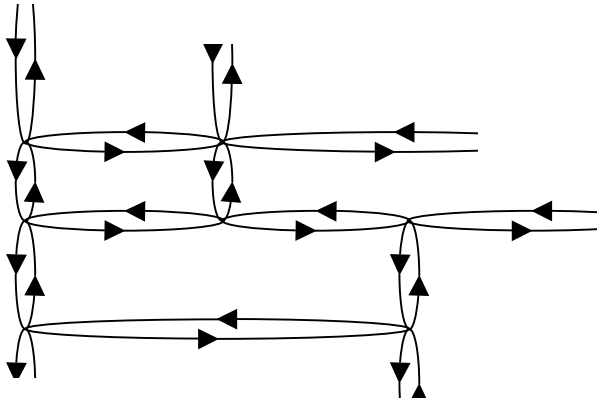

Fig. 6 *Internal representation*

The timetable updating system computes the time for every part of the covered road by using dynamic information from the vehicles and the static information about the network. Therefore we need the total of the covered road of a vehicle since the last update of that vehicle to compute an estimate for the time to cover the separate links:

$$D = \sum_l d_l$$

$$M_l = \frac{d_l}{D}(t_2 - t_1) \qquad (1)$$

d_l is the covered distance on link l. D is the covered road for the update of this vehicle.
t_1 and t_2 are the times of the updates. M_l is the measurement of the time for link l.

On quiet roads it is very well conceivable that there will be no vehicles that send route information for a long time. When there are no updates for a certain part of a road we still want to know an estimate for the time it takes to travel that way. For that purpose we can use the length of the road, the maximum allowed speed and some correction factor. The length and speed yield a time estimate for the road. That time can be adjusted a little with

the correction factor to represent that the average speed will be a bit higher (or lower) than the maximum allowed speed. This yields a default value for the travel time of the road. When we look at a road with a length of 200 meter where the average speed is 50 km/h (= 14 m/s), the default value will be 14 seconds. When at some time a vehicle covers that road in 20 seconds we want the entry in the timetable to be adjusted so that it represents a value of about 20 seconds. This way the routing algorithm will less likely use the road for other vehicles. But half a day later it is useless to know that there has ever been a car that was delayed on that road. The situation has changed many times since then. In fact information older than an hour is usually obsolete. So what we really want is that new information gives an impulse to adjust the time, but the effect should gradually diminish. And after for example an hour the effect of the information should be faded out completely. If closely after the first a second car provides the system with new information, then we should compute a weighed average. The information of the first car counts less than the information of the second, because it is older. And when the information of both vehicles gets a little older the default value should become more important. There are several ways to accomplish this, but we will use the following function because it is intuitively useful and easy to adjust.

$$\left.\begin{aligned} T(t) &= \frac{D + \sum_k W(t - S_k) \cdot M_k}{1 + \sum_k W(t - S_k)} \\ &\textit{with} \\ W(t) &= w \cdot f^{t^2} \quad \forall\ 0 \le t < h \\ W(t) &= 0 \qquad \textit{else} \end{aligned}\right\} \quad (2)$$

Here $T(t)$ is the value in the timetable at time t, D is de default value in the timetable, M_k is the result of the k^{th} measurement acquired from the information of a vehicle, S_k is the start time at which the information is added. $W(t)$ is the weight of a measurement at time t, w is the weight of the measurement at the start time, f is a factor that diminishes the weight in time, h is the time that a measurement has any effect on the outcome.

To explain this function: as long as there are no results of a measurement, only the default value D influences the value in the timetable. When, at time S_1, the first measurement is received, that value M_1 gets a weight of w (for example 25). Then the value for the timetable is a weighed average:

$$\frac{1 \cdot D + 25 \cdot M_1}{26}$$

The weight of M_1 fades in time when f is positive and less than one, for example 0.99. With the used function $W(t)$ the result of a measurement on the value of the timetable will be shaped like a half bell. This is shown in figure 7.

After h time a measurement does not have any effect anymore. So all measurements must be kept in memory for h time. The function takes care to average measurements when there are more than one available for a certain road. When a new measurement becomes available the values should be recomputed. But the values also have to be recomputed at regular intervals because of the fading effect.

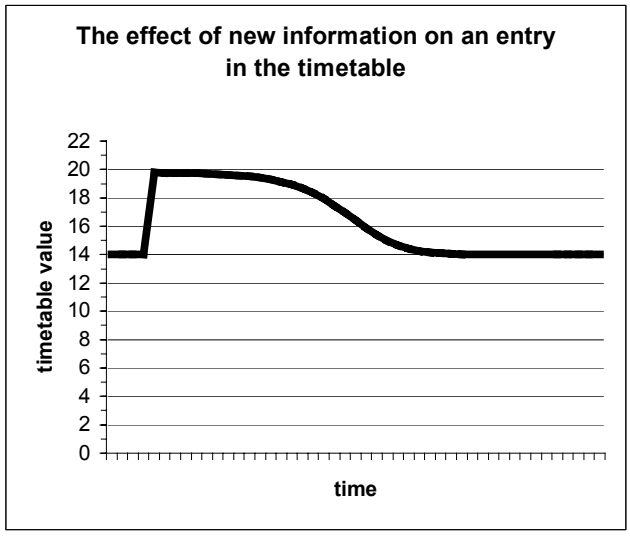

Fig. 7 *An update gives an impulse to a value in the timetable*

3.3.2 Route finding system

This system uses the earlier mentioned Ant-Based Control algorithm (ABC-algorithm). This algorithm makes use of forward and backward agents. The forward agents collect the data and the backward agents update the corresponding probability tables in the associated direction. The algorithm consists of the following steps:

- At regular time intervals from every network node **s**, a forward agent is launched with a random destination **d**: F_{sd}. This agent has a memory that is updated with new information at every node **k** that it visits. The identifier **k** of the visited node and the time it took the agent to get from the previous node to this node (according to the timetable) is added to the memory. This results in a list of ($\mathbf{k}$, $\mathbf{t}_k$)-pairs in the memory of the agent. Note that the agent can move faster than the time in the timetable.
- Each travelling agent selects the link to the next node using the probabilities in the probability table. The probabilities for the nodes that have already been visited by this agent are filtered out for this agent. Then a copy of the remaining probabilities is made for this agent and these probabilities are normalized to 1. Only this agent uses this temporary probability distribution to choose a next node, so the probability table is not updated yet.
- If an agent has no other option than going back to a previously visited node, the arising cycle is deleted from the memory of the agent.
- When the destination node **d** is reached, the agent F_{sd} generates a backward B_{ds}. The forward agent transfers all its memory to the backward agent and then destroys itself.
- The backward agent travels from destination node **d** to the source node **s** along the same path as the forward agent, but in the opposite direction. It uses its memory instead of the probability tables to find its way.
- The backward agent with previous node **f** updates the probability table in the current node **k**. The probability p_{df} associated with node **f** and destination node **d** is incremented. The other probabilities, associated with the same destination node **d** but another neighbouring node are decremented. The used formulas are given below.

The probability of the entry corresponding to the node **f** from which the backward agent has just arrived is increased using the following formula:

$$P_{new,f} = \frac{P_{old,f} + \Delta P}{1 + \Delta P} \tag{3}$$

Here, $P_{new,i}$ is the new probability, $P_{old,i}$ the old probability and ΔP the probability increase. ΔP should be inversely proportional to the age of the forward agent. The formula we use is:

$$\Delta P = \frac{a}{t} + b \tag{4}$$

Where a and b are constants and t is the trip-time of the forward agent from this node to the destination node. This trip-time is the sum of the trip-times from this node to the destination node of the forward agent. We do not take into account that the conditions of the traffic network can change from the moment that the node is visited by the forward agent and the updating of the backward agent.

The other entries in the probability table with the same destination but other neighbouring nodes are decreased using the formula:

$$P_{new,i} = \frac{P_{old,i}}{1 + \Delta P}, \quad \forall\, i \neq f \tag{5}$$

These formulas ensure that the sum of the probabilities per destination remains 1. Probabilities can only decrease if another probability increases. Probabilities can approach zero if other probabilities are increased much more often. This is not very desirable, because in time it may appear that the choice associated with that probability is the best at that time, but the agents will not detect it because they hardly ever take that route. This problem can be solved after analogy with the natural ants; they do not always use the pheromone trail as their guide, but sometimes just explore new routes. Therefore we introduce an exploration probability as a minimum value for each probability. An example could be 0.05 divided by the number of next nodes. After setting this minimum, the probabilities per destination are normalized to one again. This ensures that none of the entries in the probability table will approach zero.

For a given value of ΔP, the absolute and relative increase of P_{new} is much larger for small values of P_{old} than for large values of P_{old}. This results in a weighted change of probabilities. The formulas were taken from[3]. The probability tables are initialised with equal values in such a way that all the probabilities for one destination sum up to 1. The first agents do not have any information about routes, let alone the quality of the routes. The performance of the routing system will therefore be very bad and cannot be evaluated properly.

The quality of the routes found by the agents improves with time. At first the agents will find many cycles, but the number of cycles decreases as the probability tables are filled with information that is more accurate. Appearing congestion causes further adjustments. Finally the vehicles will be routed according to the highest probabilities in the tables. They do not have to explore other routes. They just want the best route.

4. Implementation

To test the proposed routing system a simulation environment is being developed. The development environment used is Borland Delphi 5. As well as the routing system also a simulation of traffic in a city is build. The traffic simulation should provide the dynamic information about the state of the roads in the city. And it also will use the routing system to route a certain fraction of the vehicles through the city. This simulation environment is still unfinished. Figure 8 and 9 give an impression of how a map of a city is used as a model in the simulation environment.

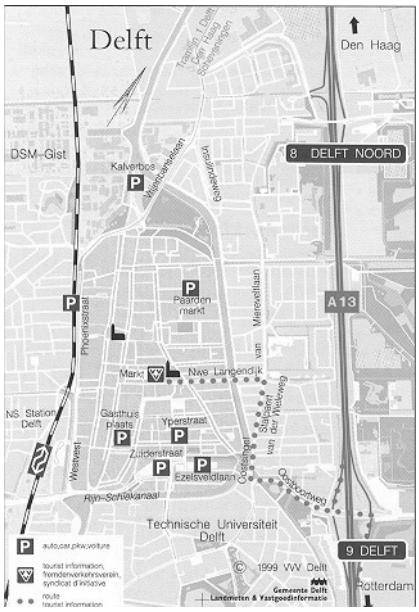

Fig. 8 *Map of centre of Delft*

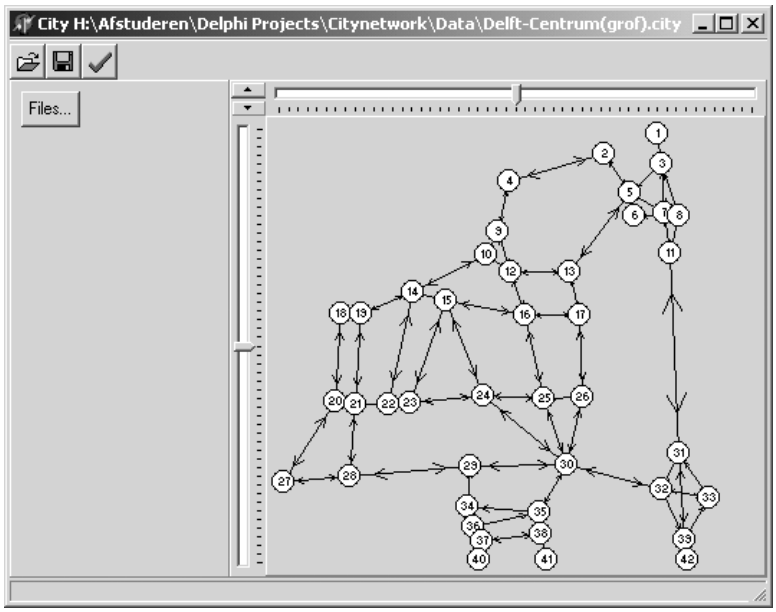

Fig. 9 *Visualisation of the map by the simulation environment*

References

[1] G. Di Caro and M. Dorigo. *AntNet: distributed stigmergetic control for communication networks*. Journal of Artificial Intelligence Research (JAIR), Volume 9, pages 317-365.

[2] R. Schoonderwoerd, O. Holland, J. Bruten, L.J.M. Rothkrantz. *Load balancing in telecommunication networks*, Adaptive Behaviour, 5, 2 , 1997.

[3] L.J.M. Rothkrantz, J.C. Wojdel, A. Wojdel, H. Knibbe. *Ant based routing algorithms*, Neural Network World, Volume 10, 2000, pages 455-462.

A REACTIVE DRIVING AGENT FOR MICROSCOPIC TRAFFIC SIMULATION

Patrick A.M. Ehlert *and* ***Leon J.M. Rothkrantz***
Knowledge Based Systems Group
Department of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: P.A.M.Ehlert@its.tudelft.nl, L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

reactive agents, microscopic traffic simulation, multi-agent systems, driving behaviour

ABSTRACT

Computer traffic simulation is important for making new traffic-control strategies. Microscopic traffic simulators can model traffic flow in a realistic manner and are ideal for agent-based vehicle control. In this paper we describe a model of a reactive agent that is used to control a simulated vehicle. The agent is capable of tactical-level driving and has different driving styles. To ensure fast reaction times, the agent's driving task is divided in several competing and reactive behaviour rules. The agent is implemented and tested in a prototype traffic simulator. The simulator consists of an urban environment with multi-lane roads, intersections, traffic lights, and vehicles. Every vehicle is controlled by a separate driving agent and all agents have individual behaviour settings. Preliminary experiments have shown that the agents exhibit human-like behaviour ranging from slow and careful to fast and aggressive driving behaviour.

1 INTRODUCTION

In the last two decades, traffic congestion has been a problem in many countries. To reduce congestion, most governments have invested in improving their infrastructure and are exploring new traffic-control strategies. A problem is that infrastructure improvements are very costly and each modification must be carefully evaluated for its impact on the traffic flow. Computer traffic simulations form a cost-effective method for making those evaluations. In addition, traffic simulations can evaluate the improvements not only under normal circumstances, but also in hypothetical situations that would be difficult to create in the real world. Obviously, the used simulation model needs to be accurate in modelling the circumstances and in predicting the results.

Intelligent agents can be used to simulate the driving behaviour of individual drivers. The adaptability and flexibility of intelligent agents allows them to control various types of vehicles with different driving styles. Each agent is equipped with its own behaviour settings to simulate personalised driving behaviour. This way, the simulated vehicles will behave realistically and the interaction between multiple drivers can be studied.

This paper describes a model of a reactive agent that can perform tactical-level driving. Tactical-level driving consists of all driving manoeuvres that are selected to achieve short-term objectives. Based on the current situation and certain pre-determined goals, the agent continuously makes control decisions in order to keep its vehicle on the road and reach its desired destination safely.

2 MICROSCOPIC TRAFFIC SIMULATORS

Many traffic simulators that are used today are macroscopic simulators. Macroscopic simulators use mathematical models that describe the flow of all vehicles. These models are often derived from fluid dynamics and treat every vehicle the same. Only the more advanced models can differentiate between vehicle types (e.g. cars, trucks, and busses) and even then all vehicles are treated equally within one vehicle type.

In real life many different types of vehicles are driven by different kind of people, each with their own driving style, thus making traffic flow rather unpredictable. In microscopic simulations, also called micro-simulations, each element is modelled separately, allowing it to interact locally with other elements. For example, every simulated vehicle can be seen as an individual with the resulting traffic flow being the emergent behaviour of the simulation. Microscopic traffic simulators are able to model the traffic flow more realistically than macroscopic simulators.

A Multi-Agent System (MAS) [Ferber 1999] can be used to form the basis of a microscopic traffic simulator. The main components (agents) of a multi-agent traffic simulator will be the vehicles. Every vehicle is controlled by an individual agent. Other important elements of the simulated environment can also be modelled as agents, for example a traffic-light agent that controls a group of traffic lights. In 1992 Frank Bomarius published a report on such a MAS [Bomarius 1992]. His idea was simply to model all the used objects as agents that could communicate the relevant data. Four years later two MSc students at the University of Edinburgh implemented this idea for their final MSc project [Chan 1996], [Chong 1996]. Their nameless text-based simulator uses Shoham's AGENT-0 architecture [Shoham 1993] to create multiple agents that function as vehicles or traffic lights, but also as roads and intersections. As the emphasis of their project was on creating a MAS-simulation and not necessarily creating realistic driving behaviour, all their vehicle agents use very simple rules based on gap acceptance and speed. More advanced behaviours like overtaking cannot be modelled due to

the simplicity of both their agent and simulation environment.

A more advanced simulation environment is the SHIVA simulator, which stands for Simulated Highways for Intelligent Vehicle Algorithms [Sukthankar et al 1996]. The SHIVA simulator was especially designed to test tactical-level driving algorithms and allows fast creation of different test scenarios. In his PhD thesis Rahul Sukthankar describes a reasoning system for tactical-level driving called POLYSAPIENT that was tested with SHIVA [Sukthankar 1997]. A drawback of the SHIVA simulator is that it needs a special SGI machine to run and is not publicly available.

At first glance, the approach we used with our driving agent resembles the POLYSAPIENT reasoning system used by Sukthankar, but its implementation is quite different. First of all, our simulator implements an urban environment. SHIVA and most other traffic simulators model highway or freeway traffic. Second, with our agent multiple behaviour parameters can be set to produce the desired driving behaviour. Most other simulators only use one or two driving-behaviour parameters (usually gap acceptance and preferred speed or an aggression factor) or none at all. Third, by using relatively independent behaviour rules our agent's functionality can be expanded or altered easily and the agent can be used in completely different environments.

3 TRADITIONAL VS. REACTIVE AGENTS

Traditional intelligent agent architectures applied in artificial intelligence use sensor information to create a world model [Wittig 1992], [Rao and Georgeff 1995]. The world model is processed by common search-based techniques, and a plan is constructed for the agent to achieve its goal. The plan is then executed as a series of actions. This traditional approach has several drawbacks. Sensor constraints and uncertainties cause the world model to be incomplete or possibly even incorrect, and most traditional planning methods cannot function under noisy and uncertain conditions. Furthermore, in complex domains like tactical driving it is infeasible to plan a complete path from the initial state to the goal state, due to the large amount of searchable states and the inability to perfectly predict the outcome of all possible actions. The amount of possible states 'explodes' if realistic manoeuvres such as aborted lane changes and emergency braking are taken into account. As a result a real-time response cannot be guaranteed, making the traditional planning methods unsuitable for tactical driving.

Reactive agents, also called reflex or behaviour-based agents, are inspired by the research done in robotic control. Their primary inspiration sources are Rodney Brooks' subsumption architecture [Brooks 1986] and behaviour-based robotics [Arkin 1998]. Reactive agents use stimulus-response rules to react to the current state of the environment that is perceived through their sensors. Pure reactive agents have no representation or symbolic model of their environment and are incapable of foreseeing what is going to happen. The main advantage of reactive agents is that they are robust and have a fast response time, but the fact that pure reactive agents do not have any memory is a severe limitation. This is the reason that most reactive agents use non-reactive enhancements.

4 DRIVING AGENT MODEL

We have designed a model of a reactive driving agent that can control a simulated vehicle. The agent is designed to perform tactical-level driving and needs to decide in real-time what manoeuvres to perform in every situation. These decisions are based on the received input from the agent's sensors. After the agent reaches a decision, the instructions are translated into control operations that are sent to the vehicle.

The driving agent is modular in design. Every part can be adapted, replaced or otherwise improved without directly affecting other modules. The agent consists of: several sensors to perceive the environment, a communication module, a memory for storing data and controller for regulating access to the memory, a short-term planner, multiple behaviour rules and behaviour parameters, and an arbiter for selecting the best action proposed by the behaviour rules. A picture of the agent's layout is shown in Figure 1.

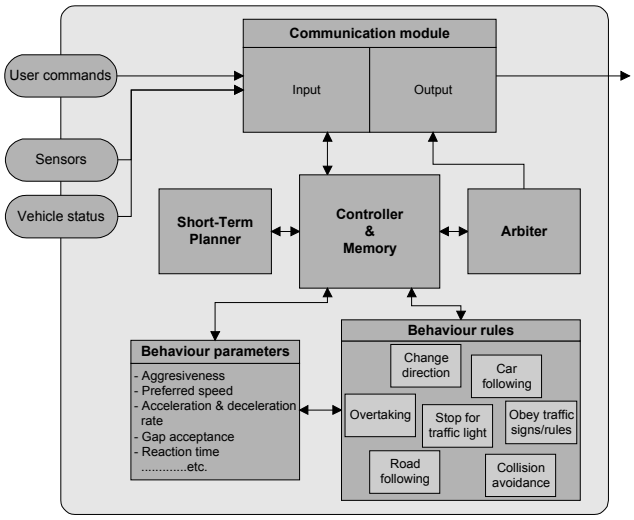

Figure 1: Driving agent layout

Our agent uses both traditional and reactive methods to perform its task, but the emphasis is on the latter since fast response times are important. Sensor information is stored in the memory and forms a temporary world model. Reactive procedures called behaviour rules or behaviours use the available information in the memory to quickly generate multiple proposals to perform a particular action. Planning in the traditional sense is not applied. The short-term planner only uses simple linear extrapolation to calculate the expected positions of moving objects and the arbiter determines the best available action based on the priority ratings of the action proposals included by the behaviour rules. We will discuss the agent's reasoning process, behaviour rules and behaviour parameters in more detail in the next subsections.

4.1 Reasoning

The complete loop from receiving sensor messages to sending an output message to the vehicle can be seen as one reasoning cycle. The timing of a reasoning cycle and the activation of the agent's parts are done by the controller that also regulates the access to the memory. Since we want the driving agent to react in at least real-time, the agent is able to complete several reasoning cycles per second. The activation of the agent's parts is shown in Figure 2.

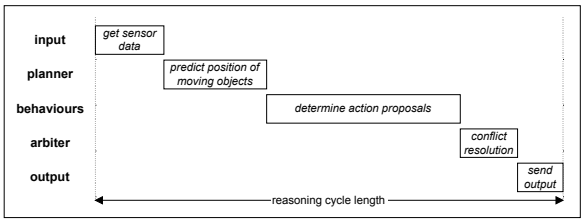

Figure 2: The reasoning cycle regulated by the agent's controller

The agent uses two types of sensor information. The first type gives information about the agent's environment, for example the distance and angle to objects, or the width of the agent's current lane. The second sensor type returns information about the agent's vehicle. This includes speed, acceleration, heading, wheel angle, and fuel level. In addition, the agent can receive orders from the user.

All information that is sent to the agent is received by the agent's communication module that contains knowledge of the used protocols. When a message is received, the communication module tries to recognise the message format, the sender and its content. When the message is ok, the input section of the communication module temporarily stores it until all received messages can be written to the agent's memory. Temporary storage is necessary since one does not want data in the memory to be read and written at the same time. Outgoing messages can be sent immediately since no conflicts can arise there. Next, all incoming messages are transferred to the agent's memory and the short-term planner makes a fast prediction of the position of all moving objects in the environment. Then the actual reasoning of the agent is performed by the behaviour rules, also called behaviours. They specify what to do in different situations. Based on the available data in the agent's memory, every behaviour can propose an action. All action proposals have a tally or priority rating. The arbiter selects the best proposal based on the priority ratings and sends it to the communication module. Finally, the communication module translates the proposal to control instructions that can be understood by the vehicle.

4.2 Behaviour rules

The agent's driving task is divided into several subtasks that are automated by independent behaviour rules. This way the agent's functionality can be expanded easily without any modifications to the existing behaviours. The used behaviour rules are very much dependent of the agent's environment. Instead of a highway environment often used in traffic simulations, we have chosen to let the agent drive in an urban environment. The reason for this is that an urban environment is one of the most difficult and complex traffic scenarios. In a city a lot of unexpected events can happen and the agent has to deal with many different situations. This way we can show the potential of our driving agent concept. Note that the design of our agent does allow driving in other environments. Only the agent's behaviour rules might need to be adapted or expanded. For our city environment we designed the following behaviours:

Road following

The road-following behaviour is responsible for keeping the agent driving on the road. Besides controlling the lateral position of the agent's vehicle, based on the distance to the road and lane edges, the road-following behaviour also influences the agent's speed. It makes sure that it slows down for curves and on straight roads it will accelerate until the desired speed set in the agent's behaviour parameters is reached.

Intersection / changing directions

If the agent approaches an intersection, its speed is reduced, precedence rules are applied, and the agent will choose one of the sideroads. Usually, this direction is chosen randomly, but it can also be set by the user. The changing-directions behaviour can be split up into several sub-behaviours, one for each type of intersection (e.g. intersections with or without traffic lights, or a roundabout). This is consistent with the fact that humans use different strategies to handle different types of intersections.

Traffic lights

The traffic-lights behaviour makes sure that the agent stops for red or yellow traffic lights if possible. The behaviour checks if the sensed traffic light regulates the agent's current lane and slows down the vehicle. The agent's start-point for braking depends on its preferred braking pressure (deceleration rate) and is stored in the behaviour parameters.

Car following

The car-following behaviour ensures that the agent does not bump into any other vehicle. If another car is driving in front of the agent, speed is reduced to match that car's speed. The precise braking pressure depends on the speed difference between the agent's vehicle and the other vehicle, the distance between them, and the set gap acceptance of the agent.

Overtaking and switching lanes

Related to the car-following behaviour is the overtaking-and-switching-lanes behaviour. If a slower vehicle is in front of the agent, it may decide to overtake this vehicle. This decision depends on the velocity difference between the two vehicles and the available space to overtake the vehicle, both in front and to the left of the other vehicle.

Applying other traffic rules

Besides traffic lights and precedence rules at junctions, other traffic rules need to be followed. Examples are, not driving at speeds above the local maximum, driving on the right side of the road as

much as possible (in the Netherlands), and no turning in one-way streets.

For the traffic-rules behaviour it is necessary to keep track of the traffic signs and restrictions encountered by the agent. Because the memory of the agent will clear data on a regular basis to save space, the traffic-rules behaviour needs to keep track of these signs itself, in its own private memory space. This memory space is embedded within the behaviour. Note that the behaviour also needs to keep track when the signs and rules apply. Usually, turning onto a new road will reset most of the current restrictions.

Collision detection and emergency braking

The collision-detection and emergency-braking behaviour is a special kind of safety measure that is activated when the agent is on a collision course with an object. It can be seen as the human reflex to brake if something pops up unexpectedly in front of the vehicle. The behaviour tries to ensure that the vehicle can be halted at all times before it hits an object. Actions from the emergency-braking behaviour have the highest priority and always overrule all other behaviours.

4.3 Behaviour parameters

In order to create different driving styles all behaviour rules are influenced by behaviour parameters. One of the most important (visible) parameter is the driver's choice of speed. This choice has a large effect on the different driving subtasks. Drivers that prefer high speeds are more likely to overtake other vehicles than slower drivers and usually brake harder. Another implemented factor is the distance the driver keeps to other cars, also called gap acceptance. Aggressive drivers keep smaller gaps than less aggressive drivers. A third parameter is the driver's preferred rate of acceleration or deceleration. Again, aggressive drivers tend to accelerate faster than less aggressive drivers.

Besides the above-mentioned behaviour factors, other aspects can influence an agent's driving behaviour, for example the reaction time of an agent and the range of its sensors. An agent's reaction time can be altered by changing the length of its reasoning cycle. The sensor range determines the visibility of the agent and can be used to simulate fog or bad weather conditions.

5 IMPLEMENTATION

We have constructed a prototype traffic simulator program to test our driving agent design. The programming language we used to build the simulator is Borland Delphi 5 Professional for NT. We have chosen this language in part since we were already familiar with it, but mainly because Delphi is an easy language, very suitable for quick prototyping.

Our simulator uses a kinematic motion model that deals with all aspects of motion apart from considerations of mass and force. The model implements smooth motion of vehicles, even during lane changes. Furthermore, the vehicles can move along realistic trajectories, but since forces are not modelled, the vehicles will perform manoeuvres without slipping.

5.1 The prototype simulator

The simulator program roughly consists of four elements: a user interface to provide visual feedback, a simulation controller, an environment containing simulated objects, and the driving agent model. The task of the simulation controller is to start, pause or stop a simulation run, and keep track of the elapsed time. The controller also initialises, starts and stops the used driving agents. During simulation, the controller regularly sends an 'update' order to the environment. The environment then calculates the new values for all its objects and sends relevant visual feedback to the screen. This 'simulation update' loop is shown in the left part of Figure 3. By default the update frequency is about 20 times per second, but this rate can be adjusted so that the program can be run on slower computers.

The environment is formed by all the simulated objects together. Different environments can be loaded via Map Data Files. These files contain a description of a road network and traffic control systems. Loading a Map Data File initialises the environment and data about the simulated objects described in the file is stored in the environment. Our current simulator implementation contains multi-lane roads, intersections, traffic lights, traffic light controllers and vehicles.

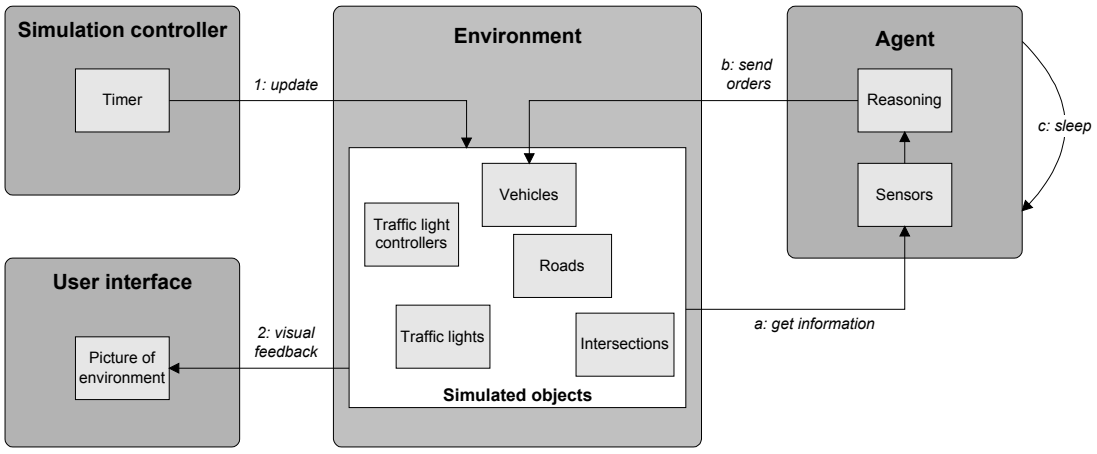

Figure 3: Simulation and agent update loops

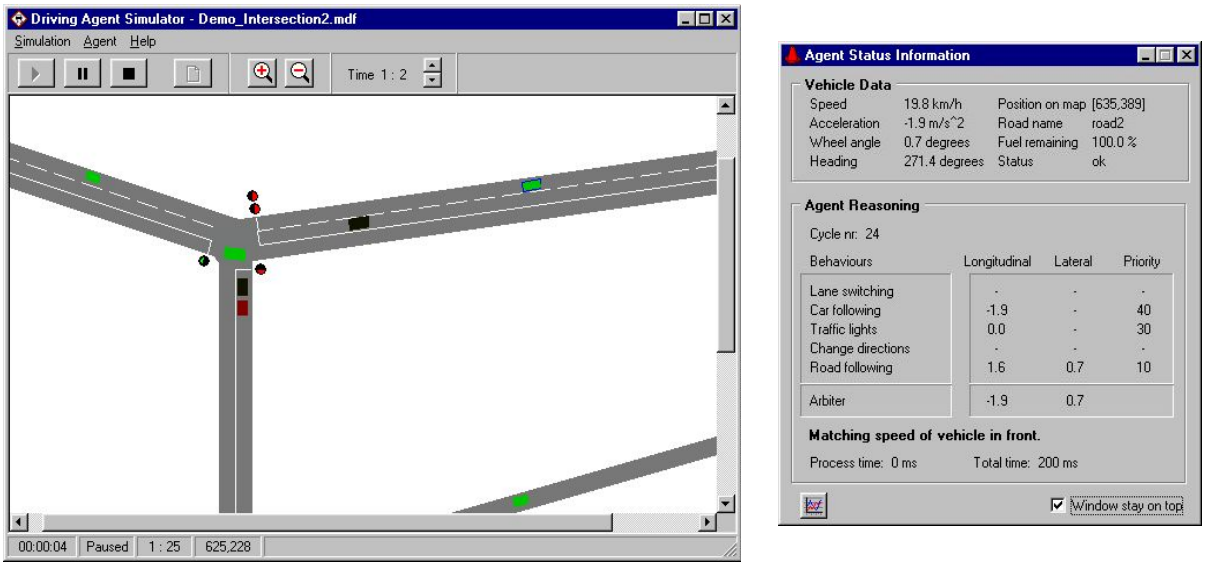

Figure 4: Screen shot of the prototype simulator used to test the driving agent

5.2 The driving agent

Every vehicle in the environment has its own driving agent, but there is one agent that has the focus of attention and can be 'controlled' by the user. This means that the user can change the settings of this agent's behaviour parameters and can follow its reasoning process in the Agent Status Information window shown in Figure 4.

All agents are implemented as threads, initialised by the simulation program. The advantage of using threads is that the simulation can be faster, running threads in parallel (if the operating system allows it), and that the agents can run independent of the simulation program. The disadvantage is that there is a limit to the number of threads one can use, because the overhead in managing multiple threads can impact the program's performance. The execution loop of an agent is shown in the right part of Figure 3. After the agent finishes a reasoning cycle its thread is put asleep for the rest of its cycle time, which is set by the simulation controller. This is done to prevent agents from using all available CPU time. By default an agent's cycle time is 200 ms, so the agents will perform 5 reasoning cycles per second.

The implementation of the behaviour rules is done using if-then rules. All behaviours are divided into several tasks. Tasks are executed in a serial manner, the least important task first and the most important task last. This way the important tasks 'override' the action proposals of less important tasks. The execution of the behaviour rules is also done consecutively, but in this case the execution order does not matter since the arbiter will wait until all behaviours are finished determining their action proposal.

6 RESULTS AND DISCUSSION

We have presented a model of a reactive driving agent that can be used to control vehicles in a microscopic traffic simulator. A prototype simulation program was constructed to test our agent design. Although we have not validated the used parameters yet, preliminary experiments have shown that the implemented agent exhibits human-like driving behaviour ranging from slow and careful to fast and aggressive driving behaviour. Here we present the results of one of our experiments, done using the first five behaviour rules discussed earlier in the "behaviour rules" section. The experiment consists of two different drivers approaching an intersection and stopping in front of a red traffic light. Both drivers perform this task without any other traffic present. The first driver is a careful driver with a low preferred speed, reasonably large gap acceptance and a low preferred rate of deceleration. We call this driver the 'grandpa' driver. The second driver is a young and aggressive driver, with a high preferred speed, small gap acceptance and a high preferred rate of deceleration. The drivers start at the same distance from the intersection. The speed of both vehicles during the experiment is shown in Figure 5.

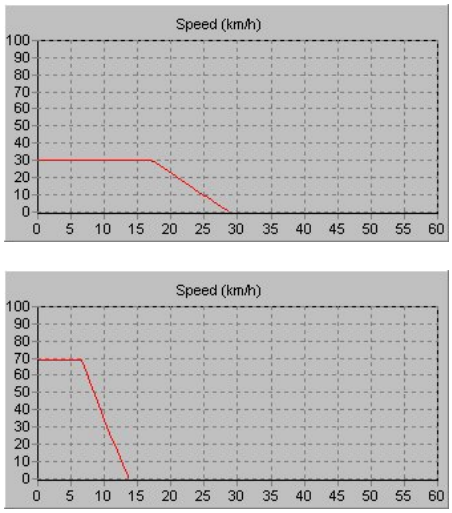

Figure 5: Speed of the grandpa driver (top) and young aggressive driver (bottom) during the experiment

Since the grandpa driver is driving at a lower speed, it takes a while before he starts braking, but his braking start-point (50m) is closer to the intersection than that of the young aggressive driver (65m), due to his lower speed. The difference between the used braking pressures is clearly visible. Both drivers brake with a relatively stable deceleration (approximately 0.7 m/s^2 and 2.7 m/s^2), which is consistent with human braking behaviour.

The experiment was done several times, but in almost all cases the shown graphs were roughly the same. Also, the precise stopping positions of both vehicles were approximately the same in all experiments. The young aggressive driver had a tendency to brake relatively late and often came to a stop just in front or on the stopping line. The grandpa driver on the other hand always came to a full stop well ahead of the stopping line. The stopping positions of both vehicles during one of the experiments is compared in Figure 6.

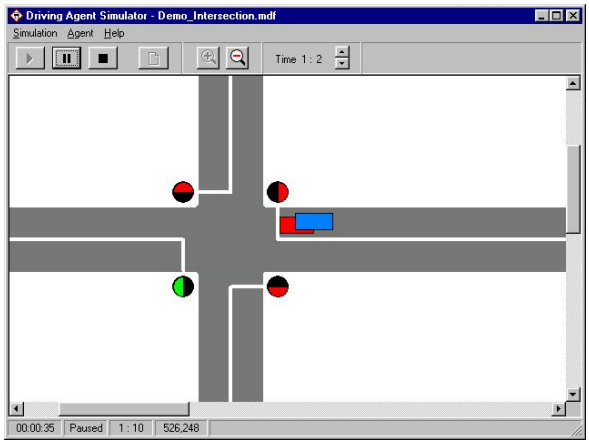

Figure 6: Compared stopping positions of the young aggressive driver (red vehicle) and grandpa driver (blue vehicle)

The aim of our simulation program was to test the design and functionality of our driving agent, but as a result its current implementation is rather inefficient since we did not optimise it for speed. Our main focus was on the correctness of the agent's driving behaviour and reasoning process. The computer used to implement and test the program is an Intel Pentium III, 450 MHz with 64 MB of RAM, running the Microsoft NT 4.00 operating system. On this computer we were able to run experiments with up to 30 vehicles. Experiments with more vehicles are possible, but result in a slow-running simulation. For this we are working on improving the simulator's memory management and processing speed.

A drawback of our simulator is that some unrealistic assumptions were made. Agent perception is perfect. All agents have a field of view of 360 degrees and objects are not obscured or covered by other objects. Further, vehicle actions are atomic. For example, braking is applied instantly after the action is sent to the vehicle. In real life this would occur more gradually. Also, pedestrians, cyclists and crosswalks are not yet modelled so the agent's ability to react to unexpected events was not yet accurately tested.

7 CONCLUSIONS AND FUTURE WORK

The main advantage of agent-based microscopic traffic simulation over the more traditional macroscopic simulation is that it is more realistic. Instead of using general traffic-flow models, traffic becomes an emergent property of the interaction between agents. Another advantage is that agent-based simulation is more flexible. Changes to traffic scenarios can be made quickly by altering the position of individual vehicles and changing agent parameters. A disadvantage is the increase of computational resources and the higher number of parameters that need to be set and validated.

Preliminary experiments have shown that our driving agent exhibits human-like driving behaviour and is capable of modelling different driving styles, ranging from slow and careful to fast and aggressive driving behaviour.

At the moment we are experimenting with different types of agents in several scenarios. The goal is to study the current possibilities of our traffic simulator and agent in order to improve them further. The simulation environment should be made more realistic by adding new objects, such as busses, trucks, emergency vehicles, pedestrian crossings, cyclists, traffic signs, trees and buildings. Once the simulator is improved with the new objects the agent's functionality must be extended to deal with these objects. In addition, the simulation environment needs to be validated. Although we have tried to use realistic values for vehicle acceleration, turn radius, road size etc., the used settings might prove to be inaccurate. We also need to study human driving behaviour more extensively in order to validate our driving style models.

The drawback of the proposed improvements will be that both the simulation environment and the agent will need more computation time and will run more slowly. Therefore, we are considering using a distributed approach in the future so that the driving agents can run on different computers. The simulation controller and environment can act as a server and the agents can be the clients communicating to the server.

REFERENCES

Arkin, R. C. (1998) *Behavior-based robotics.* The MIT Press, Cambridge, Massachusetts.

Bomarius, F. (1992) "A Multi-Agent Approach towards Modelling Urban Traffic Scenarios." Research Report RR-92-47, Deutches Forschungszentrum für Künstliche Intelligenz, September 1992.

Brooks, R.A. (1986) "A robust layered control system for a mobile robot." MIT AI Lab Memo 864, September 1985. Also in *IEEE Journal of Robotics and Automation* Vol. 2, No. 1, March 1986, pages 14-23.

Chan, S. (1996) *Multi-agent Traffic Simulation – Vehicle.* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh.

Chong, K.W. (1996) *Multi-Agent Traffic Simulation - Street, Junction and Traffic light.* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh.

Ferber, J. (1999) *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison Wesley Longman Inc., New York.

Rao, A. S., and Georgeff, M. P. (1995) "BDI Agents: From Theory to Practice." In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, June 1995, pages 312-319.

Shoham, Y. (1993) "Agent-oriented progamming." In *Artificial Intelligence 60*, pages 51-92.

Sukthankar, R., Hancock, J., Pomerleau, D. Thorpe, C. (1996) "A Simulation and Design System for Tactical Driving Algorithms". In *Proceedings of artificial intelligence, simulation and planning in high autonomy systems*.

Sukthankar, R. (1997) "Situation awareness for tactical driving." Phd Thesis, Technical report CMU-RI-TR-97-08, Robotics institute, Carnegie Mellon University, January 1997.

Wittig, T. (1992) *ARCHON: an architecture for multi-agent systems.* Ellis Horwood Limited, England.