Contents

System scenarios

CHAPTER 1

1.1 Scenario 1

1.1.1 Problems faced

Task-based analysis: scenarios

1.1.2 Possible solutions

```
User: <hungry> ‘‘Fred!’’
Fred: <beep!>
U:    ‘‘Where can I eat?’’
F:    ‘‘Pizza Hut 10 mins away’’
U:    ‘‘OK. Show me the way.’’
F:    <shows map and route>
U:    <starts walking>
F:    <updates map as user walks along>

User: <going out on a date> ‘‘Fred!’’
Fred: <beep!>
U:    ‘‘Where is a good French restaurant?’’
F:    ‘‘Provence 20 mins away’’
U:    ‘‘See the reviews’’
F:    <shows reviews by various people>
U:    ‘‘OK. Show me the way.’’
F:    <shows map and route>

User: <starts walking but road is blocked>
      ‘‘Show me another route’’
Fred: <alternative route from current position>
User: <continues walking on new route>
```

1.1.3 Considerations

What happens when the user walks off the path? Should the system catch his attention and tell him so, or should it just silently readjust the routing to the new situation.

What happens when the user cannot follow the path? Maybe a building site has blocked off a road that the system recommended taking. Should the user be able to ask for another route? Maybe just showing the map and leaving the choice up to the user would be simpler.

The GPS unit may not be sensitive enough to tell if the user is following the path exactly anyway.

1.1.4 Solutions chosen and why

We chose to make the routing dialogue as simple as possible. The system simply tracks the user's position and offers a good enough route to the destination from where they are. If the user walks off the recommended route then a new route is calculated from the user's new position.

1.2 Scenario 2

1.2.1 Problems faced

Task-based analysis: scenarios

1.2.2 Possible solutions

Scenario: Showing the way

```
-------------------------------------------------------
User: "Fred!"
Fred: Fred shows up, he looks pleased (wags its tail) "Vuf Vuf"

U: "Find Frb. 5"
F: Runs off the screen and returns a moment later dragging a big map
      onto the screen (The map fills the whole screen).

      The map shows a simple drawing of the area with a blinking dot
      indicating where Alfred is, and a line going from the dot out of
      the map. When Alfred follows the line the dot moves and as he
      approaches the edge of the map it scrolls revealing the Frb. 5
      buildings highlighted.

U: When he gets to Frb. 5 the map disappears and Fred shows up
        begging for a reward. Alfred is pleased with the guide and
        gives Fred a dog biscuit. Fred eats it and looks happy.

-------------------------------------------------------
```

Scenario: Finding Food

```
-------------------------------------------------------
U: "Fred"
F: Fred shows up, he looks pleased (wags its tail) "Vuf Vuf"
U: "Where can I get something to eat
F: Runs out and come in with a small window:
        ------------------------------
       /                              \
      | 1.  Canteen in Fib 5  ca.  10m |
      | 2.  Pizza place       ca.  23m |
      | 3.  Canteen in Novi   ca. 230m |
       \______________________________/

U: Number 1. please.
F: Gets the map

When Alfred had tasted the foot in the canteen he was very disappointed
so he takes the newspaper and slaps Fred.
```

```
F: "piv piv" and runs of the screen with his tail between his legs.
      Next time Fred will think twice about suggesting that canteen
      again.

-------------------------------------------------------
```

Scenario: Playing games

```
-------------------------------------------------------
U: "Fred"
F: Fred comes in to the screen, but Fred is angry (because last
   time he did something Alfred wacked him with the newspaper)
   so sits down with his back to Alfred.
U: "Fred do you want to play ?"
F: Jumps up and turns around and wags its tail "Vuf Vuf"
   A list of games pops up.
U: Chooses TicTacToe.

The Screen fades away and a tictactoe game comes up.
-------------------------------------------------------
```

Scenario: Batteries low

```
-------------------------------------------------------
Fred sleeps almost all the time.
-------------------------------------------------------
```

Analysis

CHAPTER 2

2.1 Catching the user's attention

2.1.1 Problems faced

Need something to take the user's attention

2.1.2 Possible solutions

Critter that goes and finds data must be congratulated and fed?

training = rating = editing

"good dog": given a biscuit
"bad dog": hit on the head with a newspaper

customizable graphics? cat/dog/alligator?

Swap and breed, like Creatures?

Maybe it should walk when you move to show it keeping up with you!

We should be able to play with the creature without asking questions of it.

2.1.3 Considerations

Tamagotchi has a personality - happy or sad. "Users" then compare stats as to how well they have done ("this one's ten days old!") - ref. There is an MIT thesis on the web about different creature interfaces but I can't find it anymore!

2.1.4 Solutions chosen and why

It's a dog!

It doesn't always understand what you say and it has a limited set of commands (not too anthropomorphic).

We should have an animated, cartoon-style interface.

2.2 Affective user interface

Design considerations about the user interface.

2.2.1 Problems faced

Why choose an affective interface? What implications does this have on our project?

2.2.2 Possible solutions

PDAs at the moment are all using a book or calendar style user interface. They pretend to be a diary. This causes problems since the size of the screen is limited and only a certain amount of information can be displayed.

A Tamagotchi has a *very* simple affective interface but it is still effective – users get attached (even if it's only to the extent of comparing stats with other owners).

An affective interface tied to an intelligent agent also maximizes the use of screen space – only important things are displayed.

For first things we need a title that reflects it. Something along the lines of: Affective Interface + Personal Digital Assistant = Case Study: GPS city guide

- As input we have:
  - Time
  - Position
  - Speech
  - Keyboard
  - Database of places and positions. *Note:* The database can be added to! It should also be *typed* so that a restaurant is different from a bus stop.
- As output we have:
  - Speech
  - Graphics
  - Sound

  — all of which should reflect the character and personality of the agent and provide a friendly and adaptive interface
- In between these we need:
  - *Animal behaviour:* "Train" to do new "tricks" by "playing" with it. BUT only if the animal is in a good mood.
  - Basic behaviour:
    - "Fetch!" nearest X
    - "Where is it?"
  - Advanced behaviour:

    - "Stay!" - remember this place
    - "Follow!" - another agent
    - together with "Be followed!" (needs a start and stop)
- *Mood traits:* The agent is sullen when it needs to know more about the user. The user then has to play (or interact in some way) with the agent to make it happy again. See notes below on *disclosure*.
- Maybe there should be male and female versions, like the Love-Tamagotchis, that would play in different ways.

2.2.3 Considerations

We don't need exact equivalents to the dog. As in the "Anti-Mac Interface" (http://www.acm.org/cacm/AUG96/antimac.htm) the user is used to gadgets (especially the mobile phone user) and doesn't need a strong metaphor. However, this is only true of things that are already in the culture...

If the database is typed does this mean we should be able to add types on the fly?

Maybe the database should forget places you say "no" to - a kind of garbage collecting facility!

We should always display the current mood (maybe in the style of a Tamagotchi by showing a number of hearts, 4 being really happy and 1 being really sad).

Young girls make friends by *disclosing* secrets (such as which boy they really fancy). This could be a model for making friends with our agent – you have to disclose your favourite preferences in order for it to like you. For example, your favourite restaurant, the time you like to eat best, or how much money you like to spend.

However, the criteria for the choice *change* with context. For example, you might want a cheap meal when you're feeling poor and a pizza when you're hungry. This could be very difficult to model.

2.2.4 Solutions chosen and why

Working title: "What do you get when you cross a Tamagotchi with a PDA? Case study: A GPS City Guide"

2.3 Initial Analysis

- Input Devices:
  - GPS:

    Input: GPS hardware (and electronic compass). Also takes into account the continuous correction from the base station.

    Output: Current position (x, y, z)
    Current time and date
    Velocity (direction and speed of travel)
    (Orientation – if compass available)
  - Speech Recogniser:

    Input: Spoken commands. Current grammar/keywords

    Output: Command or speech recognised
  - Keypad:

    Input: Signals from phone keypad (or laptop keypad)

    Output: Commands

- Output Devices:
  - Sound
  - Display

  Not much to say about either of these. The sound device might need to output speech.

2.4 Internal State

Mood: stores the current mood of the character and a counter indicating the number of queries the character has answered in that mood.

Database: Each type of place has a separate schema. A new type of place means that a new schema is created. We never want to select across different types of places so it's fine to put them in separate schemas.

e.g.	Eating Place	position, name, opening times, price range
	Offices	position, name, opening times
	Bus Stop	position, name, opening times

– opening times here refer to first and last bus! (position, name) is always the primary key. When a new type is created it can choose from amongst a fixed set of attributes, each with a fixed domain, to add to the basic position and name.

List of attributes (fixed):

Opening times: two times (hours and minutes) When setting them the user can choose from a set of predefined times (e.g. 24hr, 8-16, 10-23) or choose to set the open and shut time individually.

Price range: a keyword from the following list: very cheap, cheap, normal, expensive, very expensive

"History":

This is represented by a decision support system that keeps track of when a particular place was favoured.

For example, this could be done as a table of probabilities:

	MONDAY					TUESDAY..	
	morning	lunch	afternoon	evening	night	morning	nowhen
McDs	10%	30%	0%	0%	0%	0%	20%
Pizza	0%	0%	0%	20%	0%	0%	0%

This would indicate that you nearly always eat pizza sometime during the week, and Monday evening is a particular favourite. McDonalds is not so popular all the time, but it is on Monday lunchtimes.

Nowhen indicates the probability that you don't eat in McDs at all during the week.

2.5 Tag game

Design considerations on tag game function.

2.5.1 Problems faced

Sergio was travelling with three cars following each other. By the time they had crossed the fjord they had all lost each other and had to talk for ages on mobile phones until they found one another again.

2.5.2 Possible solutions

Can the device follow other devices?

Maybe look for another device instead of a fixed place.

2.6 Advanced features

Design considerations on new features in the system.

2.6.1 Problems faced

Do we need advanced features? Anyway, how does the user "play" to disclose things?

2.6.2 Possible solutions

Maybe we should have a "bonding session", together with the "pet psychiatrist"!

The pet will ask questions such as:

"What is your favourite time to eat?" "What is your favourite food?"

If the pet's mood goes up and stays up high enough you get an extra feature when you next go to the pet psychiatrist.

How should the new features be introduced? A quick explanation? By the pet? In text or by voice? What if the user doesn't understand?

2.6.3 Considerations

How should the pet offer help? The known commands are not static since the pet will "learn" (or reveal) new things as it gets to know the user better.

2.6.4 Solutions chosen and why

As another argument for advanced features, look at KPT by MetaCreations (http://www.metacreations.com/). This reveals features only after you have been using (and playing) with the interface for a while. The interface also provides feedback as to how experienced a user it thinks you are. This provides two advantages: firstly that the initial interface is not overwhelming to the user, and secondly that it provides something to show off to your friends!

This developing interface is also extremely appropriate for use in a dialogue system, since these need to deal with users of varying experience of the system - holding the hand of new users whilst allowing old users to steer straight to their desire.

2.7 Commands

2.7.1 Misrecognition and feedback

User is always able to see what the recogniser has understood and is able to say "No, bus stop!"

2.7.2 Cancellation

At any point, the user can cancel the current action by saying, "Cancel". This will always go back to the initial state.

2.7.3 Choice of user input

Most input options can be done by using either speech or by the keypad. If speech is available the character will be shown listening.

The available commands we have chosen are:

2.7.4 BASIC COMMANDS

"Where is the nearest X?" where X is the name of a type (e.g. eating place) The built-in types should be as short as possible and have synonyms.

This will provide a numbered list of the closest places of type X, together with the distances to them and their attributes. This list is sorted according to the preferences in the DSS.

The user can choose to select from amongst these (or reorder them) by saying:

"Show me the Y", where Y is a fixed indicator of an attribute value (e.g. "cheapest" relating to price range).

Whenever the list is displayed, the user can choose to have directions to any item by saying:

"Take me to number Z"

This will display a map with the route from the current location to the chosen target. This map will continuously update until the user has reached the target.

This selection will also automatically adjust the probabilities in the DSS. If a listed place is chosen then its probability at the current time will be increased, and the others in the list will have their probability decreased.

Choosing an item at the top of the list (reinforcing the probabilities) will improve the mood. Likewise, choosing something further down will make the character unhappy.

An unhappy choice should be recorded somehow in the DSS as a source of confusion.

"See the pet psychiatrist" Asks questions to remove possible confusion in the DSS.

Also asks if the user wishes to remove places with a very low total probability, and to remove types of places when there are no places of that type. These two questions are used to tidy up the database.

If the user answers all the questions set by the shrink then the character's mood improves.

If the character is in a good mood then the shrink doesn't ask any questions. If the character has consistently been in a good mood for a specified number of queries then an advanced feature will be made available.

2.7.5 ADVANCED COMMANDS

"Change this place" Changes the attribute values for the current location. The possible values are chosen either by speech or by using the keypad.

If there is no recognised place for the current location then the character shows a list of nearby places and allows the user to choose one to edit. If there are no known places nearby then the character will say so.

"Remember this place" The user is prompted to add a name, a type and attribute values for the current location.

When choosing a type, the user has the option to create a new type. The new type must be given a name that the speech recogniser can recognise as distinct from the names of all the other types. To do this, the speech recogniser has a dictionary of likely things that might need to be located and merely chooses from this list. The user is then asked whether that is what was meant and is given the option to say the same thing more clearly or to choose another name.

Once the name of the new type has been chosen, the user has to select which attributes are relevant and give them values.

Whenever the user is adding a new place, a list of all nearby places that might be the same is shown.

"Find Tag" Tag is the game that is played by children when one person is 'tag' and the others must then run from them. In our implementation things run in reverse: One person decides to be 'tag' and everyone else can then follow him.

This command allows the user to find all the local people who are 'tag'. They are identified by the name that they chose to put into their device when they first turned it on. The user can choose to follow one of them by saying,

"Follow number X!" where the number indicates a position in the list.

This will display a map with the current position, the current position of the 'tag' and a suggested route.

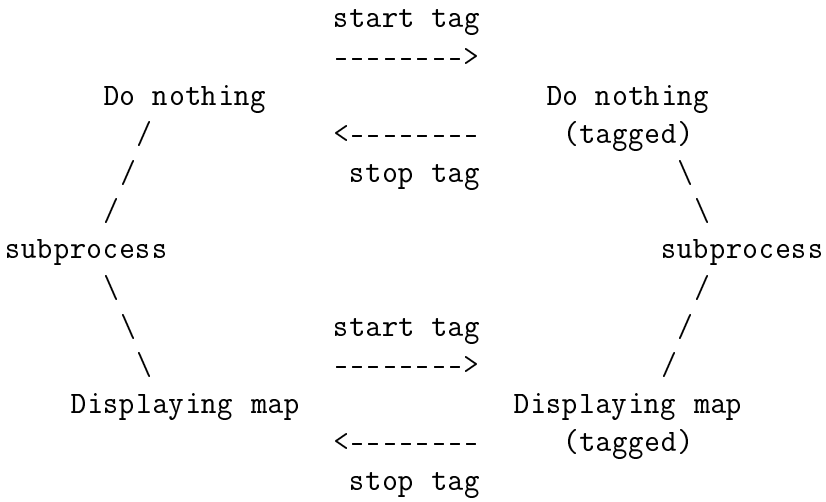
If the 'tag' stops being 'tag' or stops broadcasting for a specified period of time, then the character gives up the search.

"Start Tag" This starts broadcasting the current position for a game of 'tag'. The display will continuously indicate that the user is 'tag' until the user requests that this be stopped by saying,

"Stop tag"

2.8 System States

We have six possible states in our system:



During subprocesses the only top-level command available is "Cancel", else the user is constrained to the current dialogue.

Otherwise all commands are available at all times.

If the system is displaying a map and a top-level command is issued then the system will implicitly cancel the current display before executing the new command.

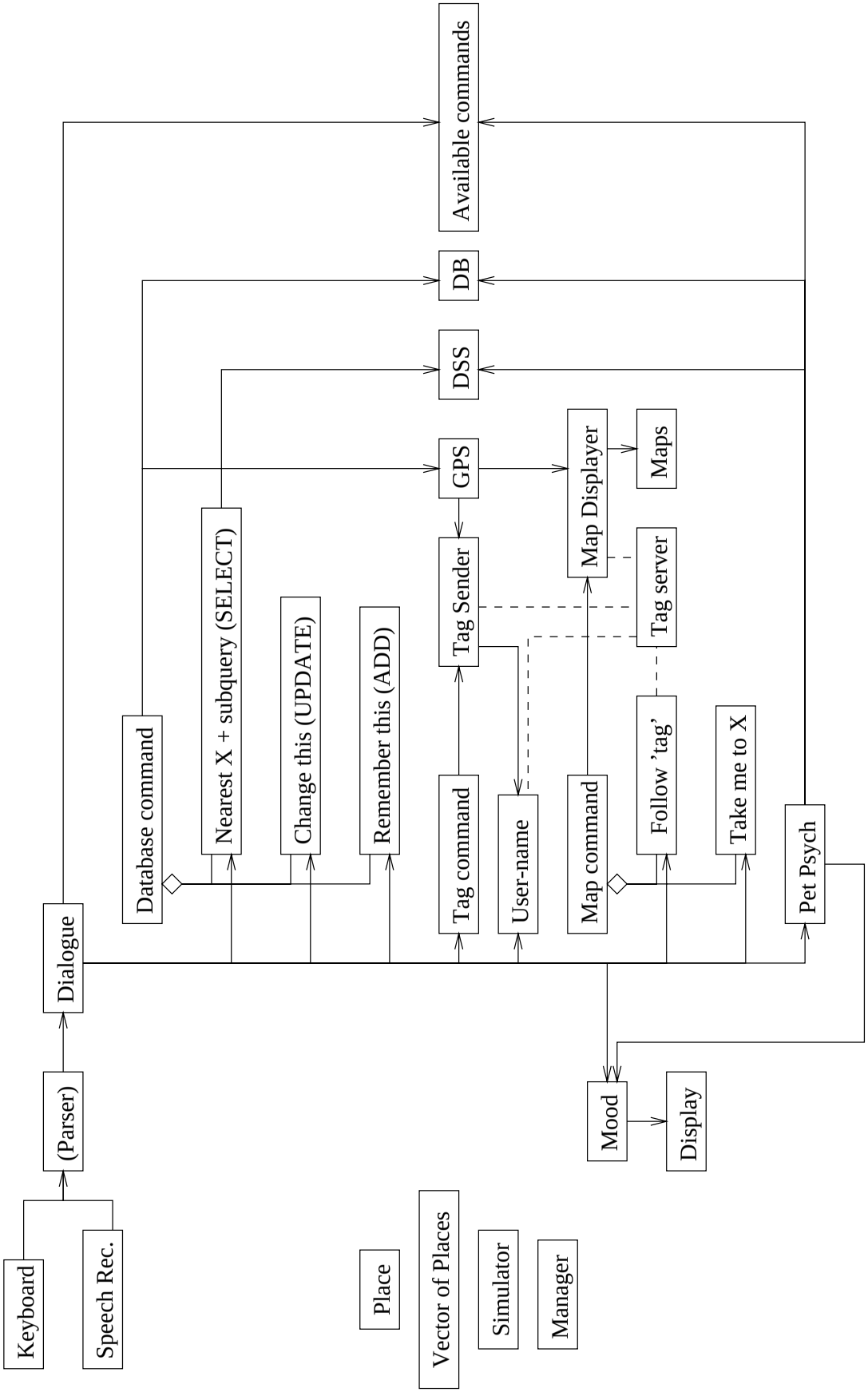


Figure 2.1: Object Orientated Analysis of our system. Arrows indicate 'CONTAINS A' and diamonds indicate 'INHERITS FROM'.

Worknotes

CHAPTER 3

The design was made by specifying interfaces for all the different modules. Each interface was written as a Java interface with Javadoc comments. These interfaces are available on our web site (http://www.kom.auc.dk/~sbc/s9/www/) and on the disk supplied with these worknotes. The design worknotes are grouped according to the different units that they describe.

System issues

CHAPTER 4

4.1 Nearest function

Design specification on the `NearestX` function.

4.1.1 Problems faced

The user should be able to use spatial terms we she communicates with the system like nearest (e.g. Show me the nearest restaurants). This gives some problem in grounding this term, because it can be very individual when a user thinks a restaurant is near (if she has a car or just a bike).

The other problem is a design problem, where should the definition of the spatial term be, in the function that uses them (`NearestX`) or part of the user model?.

4.1.2 Possible solutions

There is different ways of defining the term *near*. The easiest way is to use a constant:

- Near = under 500 meters away.

This is not very useful because we then expect that all users agree that 500 meters is near. Another solution is to make the term *near* into a fuzzy set:

$$Near : p = \left({}^{0m}/_{1}, {}^{100m}/_{0.9}, {}^{200m}/_{0.7}, {}^{500m}/_{0.4}, {}^{1000m}/_{0.3}, {}^{2000m}/_{0.1}\right) \tag{4.1}$$

or even better:

$$Near : \sum_{x>0} x/\frac{1}{0.5 \cdot x^2} \tag{4.2}$$

This is a much better way of defining near, but there is still some problems because we define the fuzzy set function and that might not match all users.

4.1.3 Solutions chosen and why

The DSS already has a definition on how much the user likes to travel, and since this feature is learned by the system, it is a user dependent variable by definition. So the *near* term is implicitly defined in this variable. This gives us the solution that the term near is defined in the DSS so `NearestX` should ask the `DSS` for the nearest places.

To improve a future system the travel definition in the `DSS` might have to refer to specific type, because a user's preferences can change with the type of place she wants to go to (restaurant or bus stop).

4.2 Change this place

Design specification on the `Change this` function.

4.2.1 Problems faced

When the user wants to changes or add a new place (by saying "Change this place" or "Remember this place"), we have to check if there are any places in the database that are close to the current position of the user. If there are any, these should be displayed as a list, so the user can confirm that the place she wants to add doesn't already exist (Remember this place) or that the place she wants to change is actually known by the system.

To find the *near* places we have to define the term near, and then search the database. This raises two problems: one the definition of *near*, and second the design specification of database because to find near places we have to make a `select()` on the whole database, and when we designed the database we said that we never needed to select across tables (different types like restaurant or office) so the `PlaceTable` was designed only to contain one type.

4.2.2 Possible solutions

The near problem is actually a problem of defining the size and shape of buildings, so we know when a user is inside a building but since each building is only saved as one position we need to get all the buildings in some *near* radius of the user and then let the user select the building in mind.

4.2.3 Considerations

To have the shapes of the buildings in the database would demand a lot of time when new places are added and since we want to give the user an option to add new places it has to be fast and easy. Therefore we choose to represent each building by a position and a radius.

4.2.4 Solutions chosen and why

The database problem was solved by adding a `pure` variable to `PlaceTable` class. This variable indicated if a placetable contained different types (e.g. restaurant,office). This way we could make a selection from the whole database and return a `PlaceTable` with near places.

The near problem we solved by adding a AttrLocation class containing a position and a radius. The radius is the size of the building which was set as default to 10 meters, could be changed if the "place" was very small or big. The size of the radius also has to reflect the uncertainty of the user's position.

4.3 Attributes

4.3.1 Problems faced

Should the user be able to add attributes like they can add types?

4.3.2 Possible solutions

Adding attributes on the fly would be slightly awkward as they have to be recognised by the speech recogniser.

Maybe attributes could be added by purchasing an extra pack that would add several pre-defined attributes at once.

For example, adding a shop attributes pack would provide attributes like 'merchandise type' (clothes, shoes, etc).

4.3.3 Considerations

The problem comes in with the database: each type is defined by an individual relation. If we want to add an attribute to a type we have to add a new attribute to the relation. However, this means adding attribute values for each of the items already in the relation, or using NULLs to represent unknown data.

The first of these choices means spending a long time asking the user for more information, and the second means that we are unable to search most of the relation using the new attribute.

We want to have types because we want to be able to know about different things in the environment e.g. restaurant, office, etc.

4.3.4 Solutions chosen and why

We fix the possible attributes (and their domains) and only allow users to add new types.

4.4 Character Animation

Design considerations on the animation of the doggy.

4.4.1 Problems faced

Some users interpreted that when the cartoon was getting "sad" it was their fault and they could get depressed. We got this input after some users saw our suggested animations.

4.4.2 Possible solutions

Instead it could just look confused so that the user would not get unhappy but instead would go to the Pet Psychiatrist and try to stop his doggy from being confused any more.

4.4.3 Considerations

We have at least two different behaviours in our system ("good" and "bad"). For each of these moods we could have different animations.

4.4.4 Solutions chosen and why

The suggested solution sounds reasonable for us and so we have decided to implement it.

DSS

Chapter 5

5.1 Questions from Psychiatrist

Design considerations on how to make the questions for the Pet Psychiatrist.

5.1.1 Problems faced

How should the system manufacture the questions used in the Pet Psychiatrist? We cannot say anything we want because synthetic speech is very difficult to hear down a phone line, especially a mobile phone line.

We also need to make some kind of question framework otherwise it will be very difficult to express the questions to the user.

5.1.2 Possible solutions

Should questions make new connections in the database?

Suggested questions:

- Which one of these (attributes) do you prefer? – the DSS has ratings about attributes so the shrink should ask questions about them.
- Please tell your pet...
- <Showing a line from one extreme to another with numbers in between> Choose the number that best represents your feelings about this parameter.
- <Showing a line from one parameter to another with numbers again> Which parameter do you prefer and by how much?
- *similar to above:* <Show a triangle with different parameters at each vertex and various positions numbered> ... *NOTE* that this does not generalize to further dimensions – the next figure you need is a tetrahedron!

One way of structuring this would be to have a getProblem function in the PetPsych that would return some problem recorded in the history of the DSS (i.e. some point at which the user chose a place that was quite a way off the top of the recommendations).

This problem would be stated in terms of a time and position (to remind the user), the place chosen and the alternatives offered. The dialog system would then ask a standard question using this information, such as:

"A few days ago, you chose to go to McDonalds rather than any of these places <shows a list>. Was that because of (1) the food type, (2) the price range or (3) you wanted to travel more?"

The answer would then be sent back to the PetPsych by providing the place chosen, its alternatives and the attribute supposed to be the reason for the choice. The PetPsych would then update the DSS accordingly.

Note that the user might quite easily say, "None of these was the reason – I just felt like a change" and so the PetPsych must be ready for this.

Another question which should be asked is, "You never select this restaurant when it is offered and appear to really dislike it. Do you want your pet to forget about it?" This would then provide a way of tidying up the database.

An alternative structure would be to have a Question class. All questions would then be multi-guess and would have two main functions, one to convert them to a string (or whatever form is convenient to display/speak them) and another to receive the answer (which would be an index indicating which answer was given). The answer would then be stored in the question object.

The PetPsych would then have a getQuestion function, and to answer there would be a sendAnswer function, which would do the necessary stuff to the DSS. This would involve using RTTI (Run-Time Type Information) to determine what kind of question had been asked.

The Question class could even include an update function so that the Question could perform the update by itself. This would involve the object having been created by the PetPsych with references to the DSS, the DB and the AvailableCommands objects (the objects that need updating).

This avoids large scale use of RTTI (always a good sign).

This solution means we have to find some kind of general description of a question that can be passed to the Dialog to output it to the user.

5.2 Basic Design of DSS

Design considerations on DSS.

5.2.1 Problems faced

How does the DSS work?

5.2.2 Possible solutions

As Steen said - model the user by a set of random variables (RVs).

Each RV reflects the probability that the user likes a particular attribute over time e.g. on Fridays he probably likes fish restaurants.

The RVs are then combined to make some utility function, taking the attributes of a particular place and returning a score.

The DSS object gets a table from the database, together with a time and a position and returns the table sorted according to the utility function. If the user specifies extra stuff (like "cheapest", or "just Italian") then the DSS gets the table, the preference (as the value of an attribute used as a bias), the time and the position and returns another sorted table. It also updates the user model to reflect the fact that this choice was made at this time.

How should the time be modelled? Just four sections per day (morning, lunch, dinner and night), or do we want seasons as well (to cope with the idea that people want to travel less when the weather is bad...)?

Maybe we should use time at current velocity instead of the distance. However, how would we work out the current velocity? If the user had just got off a bus they wouldn't want to jump back on to travel a long way away. Also, the interface has to be consistent – if you stand in the same place you want to see the same restaurants listed, not much further ones if you've just got off a bus.

5.2.3 Considerations

We should take out the time aspect to begin with to simplify the decision-making process.

Restaurants have more than one food type – places serve Mexican and Italian (especially in Denmark!).

5.3 The Probability of an Observation

How do we calculate the probability of the user's choice (used to update the RVs in the DSS).

5.3.1 Problems faced

To update the probabilities of the preferences in the user model, we need to multiply the previous probability of the preferences by the probabability of the new information given the preferences (Bayes rule: $P(prefs|newinfo) = P(prefs) \cdot P(newinfo|prefs)$).

The joint probability of the preferences is easy to calculate (they are independent so we can simple multiply them together), but how do we calculate the probability of the new information given the preferences? And what exactly do we observe each time the user makes a choice?

5.3.2 Possible solutions

Each time the user makes a choice, the system has come up with various suggestions or predictions. When the user chooses one of them we would like the system to take this choice into account and update the preferences in such a way as to increase the score of the chosen choice next time. To do this we need some kind of *error function* that quantifies the amount of error in the system's predictions.

Original one from Steve:

A sort of least-squares error function. We choose some arbitrary point that represents the utility value that the system should have come up with for the place that was chosen. We then go through the preferences, changing the probabilities so that the chosen item achieves that utility value. See figure 5.1.

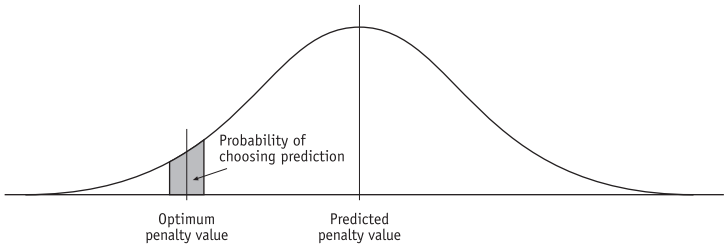


Figure 5.1: Original method for calculating conditional probability

Complicated one from Steen:

Each of the predictions is a utility value somewhere along the utility axis. If we say that each prediction is not particularly accurate then we can put a normal distribution around it and measure the probability of each prediction being in a different place by making various statements of the following kind:

$P(C_2 > C_1 \wedge C_2 < C_3 \wedge \ldots)$

Each of these can be broken down into equations of the following form:

$P(C_2 < C_1) = \int P(C_2 < C \wedge C_1 > C) dC$

All we need to do then is calculate these equations...

Easy one from Steen:

We model the predictions of the system by a normal distribution with its mean at the minimum prediction (the recommended one) and some reasonable standard deviation. The probability of each prediction given the preferences is then considered proportional to the probability density function measured at the prediction value. To calculate the probability, we calculate the sum of the probabilities of each of the predictions and use this as a normalising factor. See figure 5.2.

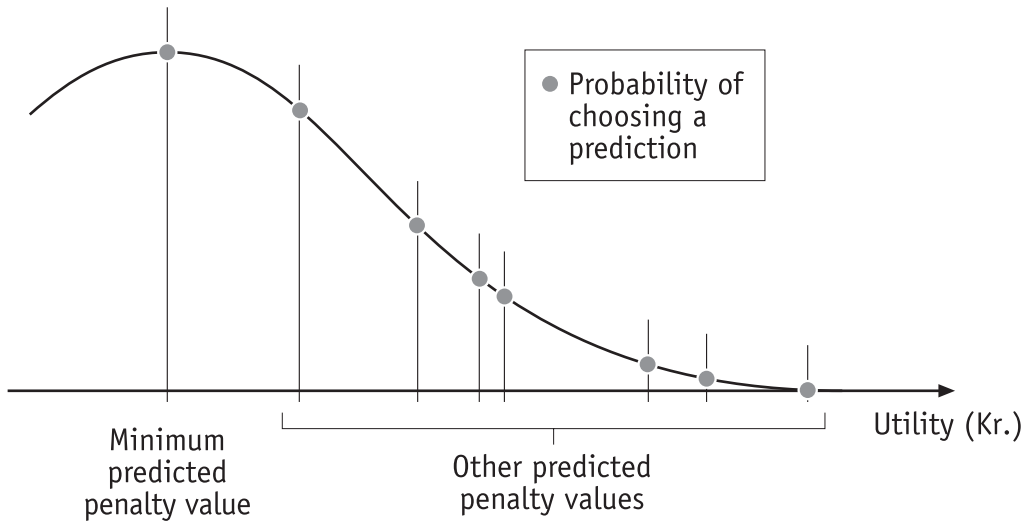


Figure 5.2: More effective method for calculating conditional probability

5.3.3 Considerations

Original one from Steve:

This method has a bug in that it has a stable state in which all chosen places have the optimal utility. Not only is this state stable, but the system would seem to tend towards this state. This is obviously not what we want, since this state will not reflect the user's preferences at all!

Complicated one from Steen:

All very well in theory, but now we have all these nasty integrals to calculate...

Easy one from Steen:

Small problem: brings down anything lower than the min. But then we don't have anything lower than the min...

Larger problem: needs a fair bit of calculation...

5.3.4 Solutions chosen and why

Let's try the easy one from Steen.

5.4 How to Represent a Choice

Design considerations on how to represent the user's choice within the system.

5.4.1 Problems faced

At various points in our interface, the user is presented with a list of places. To the dialog system, this list is represented as two lists of strings: the info list, containing the strings shown on the screen; and the command list, which at this point will contain the numbers of the indices of each of the items in the info list (together with Help and Cancel). To the Java system, however, this list is represented as a PlaceTable: an object that uses a JGL[1] HashSet to implement a Set of Place objects.

If the user chooses one of the places, the dialog system will return an integer referring to one of the items in the info list. How do we then represent this choice in the Java system?

As an example, the Nearest object has a function that provides a PlaceTable of Places that are near the user's current position graded according to their preferences. Another function must take the user's choice, update the preferences and start the MapDisplayer.

5.4.2 Possible solutions

One solution would be to represent the choice as the Place object at the index given by the dialog system. However, this would mean that the Nearest object either has to keep a record of the PlaceTable that is currently in use and the current position of the user or has to be passed these two parameters again when the choice is made.

Another solution is to use a JGL iterator. These are essentially well-behaved pointers. The ForwardIterator can even return the container it refers to. This would seem to indicate that the user's choice can be represented simply by a ForwardIterator over the PlaceTable and the user's current position. However, the ForwardIterators returned by the PlaceTable class iterate over the HashSet it contains and not the PlaceTable itself. Thus we still have to provide the PlaceTable.

5.4.3 Solutions chosen and why

We represent the choice as a Place, the PlaceTable it comes from and the user's current position. This contains all and only the information we need (i.e. we don't need to worry about the state of any object we pass it to).

[1] The Generic Collection Library for Java

5.5 How much information to display

Design considerations on how much information to display to the user.

5.5.1 Problems faced

After an initial user test, a major criticism of the system was that not enough information about each place was being displayed.

What information should the display show for each restaurant?

5.5.2 Possible solutions

The only information we have about each restaurant is its name, its location and its price range. At the moment, we only display the name and the price range.

It would be quite easy to add the distance to each restaurant as well. This would simply involve letting the JavaDialog connect to the GPS unit and have a method to fetch the reference so that the PlaceDialog could fetch the user's current position.

5.5.3 Considerations

One reason for the complaint was that the names of the places were compressed and not very informative (e.g. "FrB" represented the canteen in building 15).

Adding the distance would give an immediate impression as to how far away each restaurant was, without having to display any graphical data.

5.5.4 Solutions chosen and why

We lengthened the names of each of the restaurants ("FrB" became "Building15_Canteen") and added the distance to each of them in the display.

Database

CHAPTER 6

6.1 Representing Attributes in the Database

Design considerations on the attributes in the database.

6.1.1 Problems faced

How should we represent attributes?

6.1.2 Possible solutions

The database is object orientated: there is a class of Places which has several subclasses (Restaurants, BusStops, etc.). Each Place contains a name and a position. Each subclass contains several attributes as well.

Attributes are also objects since their definitions must remain the same across different types of place.

To select from the database, you need to express preferences about the attributes.

Maybe each attribute has a "match" function that operates in a way appropriate to the data it contains.

OpeningHours (represented as a time range) match if the time range overlaps.

Names match if they are equal.

FoodTypes match if there is something left when you AND them together (this allows for a FoodType to be an OR of several different styles, e.g. Italian/Mexican/Seafood).

PriceRanges match if they intersect.

How do we differentiate between similar restaurants? For example, there might be two Italian restaurants very close together with a similar price range, but the user might prefer one due to better service, decor, clientele, etc.

How about putting an extra field into the Places class which is a user-defined preference (i.e. good, average, bad)? This could then be used directly in the utility function and could be set by the user themselves if they chose. This would be another special attribute kind of like Position, in that its use is implicit.

6.1.3 Considerations

What about OpeningHours? It can't just be a simple pair of times since this won't work on weekends. What about festivals? Long Saturdays? What can we choose if we're then going to have to input it all when the user adds a new place?

The user could choose from various default options (e.g. Office Hours, Restaurant Times) and then tweak them.

Søren thought of the Palm's display which shows periods in a week in a graphical form (like a chart).

6.1.4 Solutions chosen and why

We choose to only look at one week's worth of opening hours. Otherwise things get absurdedly complex and something will be missed out anyway.

Each type of place will have its own default opening hours (and other attributes) which the user can change if necessary.

Each Place also has a user-defined preference value that is always average unless it has been specifically changed.

6.2 Attribute Java Interface

Design considerations on the Attribute Java interface.

6.2.1 Problems faced

Every kind of attribute has different ways of being empty. What is meant by being empty may well vary from one attribute to another.

6.2.2 Possible solutions

Make a method setEmpty in the Attribute.java interface that must be made whenever you implement the interface.

6.2.3 Considerations

The aforementioned approach sounds reasonable, as it forces the programmer to consider what it means for an attribute to be empty.

6.2.4 Solutions chosen and why

The aforementioned solution is chosen because of the reasons stated above.

6.3 Database select

Design considerations on select function.

6.3.1 Problems faced

How do we select from the database?

6.3.2 Possible solutions

Once the user has asked to see the nearest restaurants, they can ask to reorder that list according to specific requests. For example, "Show me the Italian ones".

What happens if there aren't any? Do we show an empty list? Or do we just say there aren't any and keep the list as it is?

If we show an empty list, how do we go back one level? At the moment we would have to cancel and then ask for the restaurants again...

And how does this reflect on the DSS? If we're measuring from the top and the top changes, does our measure make any sense?

6.3.3 Solutions chosen and why

When the sub-selection is empty, we just inform the user of this and show the same list. Otherwise we would have to add extra interfacing to jump back one level, or have the user start from the beginning again (not good if they have done several of these selections already).

The DSS measure still makes sense – as long as we measure from the top of the original list and not the reordered one. We can't deal with the fact that one night the user might change their preferences completely (e.g. they're out on a date) – we just get the information that this night they preferred the particular food they selected.

6.4 Choosing attributes

Design considerations on which attributes to put in the database.

6.4.1 Problems faced

Which attributes should we choose and how should they work?

6.4.2 Possible solutions

Each Place has a name and a position built-in. The attributes we have thought of so far are:

TimeRange: Gives the times that the place is available. This would be opening hours for a restaurant and the times of the first and last bus for a bus stop.

FoodType: Specifies the kind of food served at a restaurant, for example Fast Food, Italian, French, Mexican.

PriceRange: The amount that a meal at a restaurant would cost.

Maybe each Place should have a general rating. This rating will only have three levels: good, average and bad. Every place will default to average and will only change if the user specifically requests so (using ChangeThisPlace). This rating would then be used to represent the user's general feelings about a place, feelings that are not covered by any of the other, more factual, attributes. For example, two Italian restaurants might have a similar price range but one might have much better service.

This general rating will add to the utility function directly but will not be updated when a choice is made.

The rating could also be acted upon by the PetPsych, e.g. "Do you still feel that this place is below average when you keep on going there?"

6.4.3 Considerations

The FoodType attribute would have to be able to have several values at a time, since some restaurants (especially in Denmark) serve several different kinds of food.

It might not be a good idea to specify the PriceRange too exactly since different people have different ideas of what expensive is. Instead we could use fuzzy terms such as "very cheap", "cheap", "average", "expensive" and "very expensive".

6.4.4 Solutions chosen and why

We decided to implement PriceRange to begin with, whilst leaving the system open to defining more attributes later.

6.5 Uniqueness of Places in PlaceTable

Design considerations on the PlaceTable structure.

6.5.1 Problems faced

How do we ensure that the PlaceTable structure can only contain unique Places?

6.5.2 Possible solutions

A candidate key for a Place is a combination of its position and its name, i.e. two places can share the same name (e.g. McDonalds) but have different positions and vice versa (e.g. Sergio's office and Adam's office).

JGL[1] supplies a HashSet for storing unique items. The items are stored by putting them into "buckets" according to their hashCode. This allows fast access to the data since we can find the hashCode of an object without having to search through the table at all. However, the default method for computing the hashCode of an Object returns the same code only for the same object in memory. Thus, unless we are searching for a Place using exactly the same object as stored in the HashSet, we will end up searching most of the way through the storage, starting from some random point defined by the (different) hashCode.

A possible solution is to define our own hashCode function, overriding the default one supplied in Object, and based on a combination of the hashCodes of the position value and the name value. We then come across two problems: firstly, we have to make a combination of two hash functions to generate a new hash function; secondly, we face the problem that the resulting hash function collides for two Places, resulting in two different Places having the same identity.

6.5.3 Considerations

Java's inbuilt hashing is not good enough to build hash combination functions that work in the general case. For a start it only uses 32 bits for a hash value, and even worse, each inbuilt class (e.g. Double or String) provides its own individual hash function that isn't necessarily any good. See http://serve.net/buz/hash.adt/java.000.html for more details as to why Java is no good at hashing.

We don't have time to implement a proper table with general hashing and JGL's classes are freely available and do everything else we need.

6.5.4 Solutions chosen and why

It is possible to weakly combine two hashing functions if you know that both the keys that are input into the hashing functions and the hashing functions themselves are independent. Such a combination can be created by XORing the hash values together.

Fortunately, Sun makes the source code for the inbuilt hashing functions available and we can see that the String hashing function is different from the Double hashing function (even though the String hashing function is pretty awful – not only does it consider

[1] The Generic Collection Library for Java

"AAAAAAAAAAAAAAAA" to have the same hash value as "ABACADAEAFAGAHAIA", but the hash values it returns are quite clustered together).

We also know that the input keys (Place name and the three coordinates of the location) are necessarily independent (otherwise we wouldn't need to combine them together at all).

Thus, we make a hash function for a Place that XORs the hash values of each of its keys together and hope for the best!

6.6 Initialization of the database

Design considerations on the database.

6.6.1 Problems faced

It would be extremely convenient to read the database of places from a text file. How do we implement this?

6.6.2 Possible solutions

Each Place should have methods to read and write their descriptions to a file. The question is, should the superclass Place have a method to write its information or should each subclass define it separately?

What should we pass around to read and write from/to? This could be plain strings – in which case each subclass of Place needs to create a StringTokenizer (or a StreamTokenizer if they have a complicated format), tokenize the String and then send the relevant bit off to the Place superclass method to be understood.

We could use the Class.forName() method to create objects of the relevant type and then call a fromString method in the Place to read the rest of the line.

6.6.3 Considerations

Should the general loop over a file of definitions be in the Database or in a separate class?

The Database class does not currently support adding lots of Places at once in the form of a PlaceTable, but the Places could be added one at a time to check that they are unique.

This functionality could be available in the Database class even though it is not made visible in the DatabaseI interface. A Database object could then find the file locally and set itself up.

Each attribute to be read in could also have a fromString method so that attributes could also be automatically created. However, each Place subclass has its attributes fixed so this might not be such a good idea...

e.g. reading in the line "PlaceRestaurant name ((location) radius) officeowner" would be slightly strange.

It would be possible to implement general reading and writing methods in each Attribute and in the Place class so that each subclass of Place would not have to do anything. However, this would mean each Attribute putting in information on which attribute it was. This would make the files unwieldy and the only point of having the text files is so that we can type them in easily. If we want to save the contents of the database we can simply serialize it.

6.6.4 Solutions chosen and why

Use the Class.forName() method.

The fromString method in each subclass of Place must therefore do its best given the String it is presented with. If it fails it can simply return false, since the code that is stepping through the file can provide information on which lines were not valid.

Each Place subclass is responsible for reading and writing its own definition using the toString and fromString methods.

New features

Chapter 7

7.1 DSS in the tagging game

Design considerations on the DSS.

7.1.1 Problems faced

We have found that it could be interesting if the system could also maintain information about which people the user often follows, for example family members and friends.

7.1.2 Possible solutions

This information could be stored in the Decision Support System as some kind of preference: the available tags would then be sorted in a similar manner to the places.

7.1.3 Considerations

In our system the DSS is not going to handle anything related to the Tag application as is not the most important part of our project and could make the DSS very complex.

7.1.4 Solutions chosen and why

This could be a good extension for our system but would add considerable complexity to the DSS since users do not have attributes like places.

7.2 User identification in the tagging game

Design considerations on the tagging game.

7.2.1 Problems faced

If we want to implement a tag application the user must identify himself to the central tag-server.

7.2.2 Possible solutions

The user could use a (spoken) username for the registration. The SIMM-cards in the phones also can serve as a means of identification.

7.2.3 Considerations

Every time the user starts the mobile phone he should have to introduce his username. There could be some problems if two mobiles have the same username (considering that there will be many different user that don't know each other).

7.2.4 Solutions chosen and why

We are going to have a username combined with the phone number. In this way we will have an unique identification of every user.

7.3 Tag Security

Security considerations for the tag aplication.

7.3.1 Problems faced

Once that the Tag utility is implemented we should address the security problems that this option could generate to the final user. One tag could be followed by someone without his authorization.

7.3.2 Possible solutions

In future releases it should be possible for a tagged person to know who is following him and to choose whether or not to allow individual people to follow him.

7.3.3 Considerations

The implementation of the acceptance of a follower would make the implementation of the current system more complex and is not a very necesary feature for the prototype that we aim to build. There should be a server accepting tags and followers, matching them together, reporting lists of tags and followers to users interested...

7.3.4 Solutions chosen and why

We are aware that future releases should implement this security option but due to the complexity of the implementation and time contraints we are not going to do it for the current release.

7.4 Task optimisation

Optimisation of various tasks to be done.

7.4.1 Problems faced

Our system could include a list of different tasks that the user wants to accomplish. For example, first go to have lunch, afterwards to the bank... It could be nice if the system kept track of the list of things to be done and suggested the best order for the the places to be visited.

7.4.2 Solutions chosen and why

This option is not in the scope of our project. It could be implemented in future versions of the system.

GPS

CHAPTER 8

8.1 Introduction to GPS

Early navigation systems

Early navigation systems such as Decca, LORAN C and OMEGA were based on signals from land-based antennas. Achieving complete coverage with this technology was complex in coast-near regions and impossible on the big oceans.

Global Positioning System - GPS

In 1973 the US armed forces started building what is now known as GPS. At first it was intended to be for military use only, but political intervention caused the development of a "dual" system: Only the US armed forces would be allowed to get top precision positioning, and for civil use a random error was introduced (Selective Availability).

24 (21 active and 3 backup) satellites are placed in orbits allowing them to cover the entire planet-surface with their signals. These navigation-satellites uses high frequencies that can penetrate the troposphere and the ionosphere. In that way the signals can be received directly from the satellites. This makes it possible to achieve a remarkable degree of precision – down to centimetres, even.

Each satellite transmits data about transmission-time and -place. The GPS receiver can calculate its distance to the satellite from the data in the GPS signal. With one satellite, this would place you anywhere on the surface of a sphere. Receiving data from two satellites narrows it down to a circle; three satellites to two points (see figure 8.1). Usually one of the two points can be rejected, as it is too far away from Earth, so a fourth satellite is - in principle - not required to get a position fix. However - as the average GPS receivers do not have extremely precise atomic clocks as do the satellites[1] - a fourth satellite is needed to get an exact fix on the position.

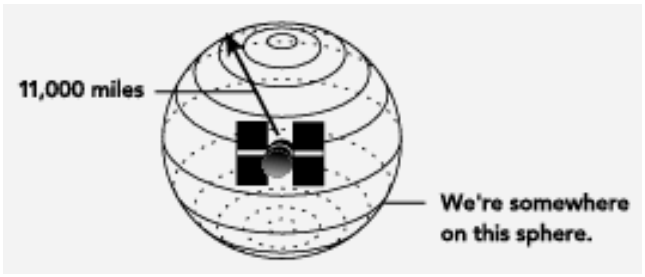
Differential GPS - DGPS

Sometimes there is a need for relative measurements more than absolute "world-coordinates" (longitude and latitude), for example when building roads. For this DGPS has been developed. DGPS involves the cooperation of two receivers, one that is stationary and another that is roving around making position measurements.

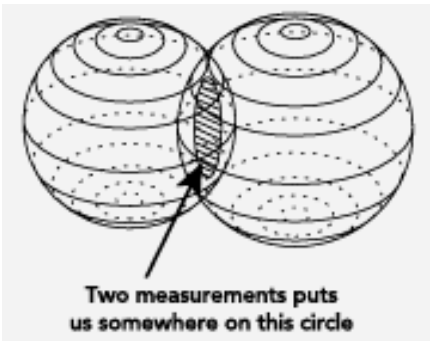
The stationary receiver ties all the satellite measurements into a solid local reference. If two receivers are fairly close to each other, say within a few hundred kilometres, the signals that reach both of them will have travelled through virtually the same slice of atmosphere, and so will have virtually the same errors.

The reference receiver is put on a point that has been very accurately surveyed and kept there. This reference station receives the same GPS signals as the roving receiver but

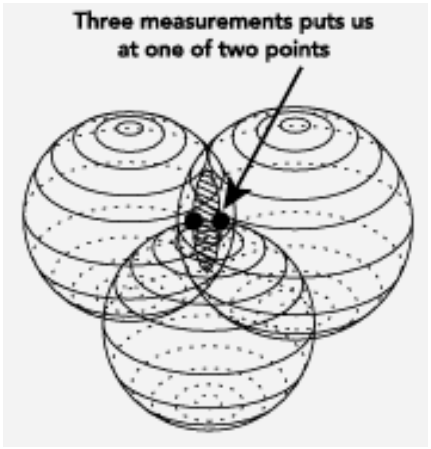
[1] Actually every satellite has *four* atomic clocks - two cesium and two rubidium ...



(a)



(b)



(c)

Figure 8.1: Getting position information using (a) one sattelite, (b) two sattelites and (c) three sattelites.

instead of working like a normal GPS receiver it attacks the equations backwards. Instead of using timing signals to calculate its position, it uses its known position to calculate timing. It figures out what the travel time of the GPS signals should be, and compares it with what they actually are. The difference is an "error correction" factor. The receiver then transmits this error information to the roving receiver so it can use it to correct its measurements. The accuracy then can be in the order of millimetres.

This approach also eliminates the Selective Availability error put in by the American Department of Defence.

The Garmin 25 board is capable of receiving a DGPS signal, but as it is a low-end product it will only achieve an accuracy at its best one to two metres. This however is quite sufficient for our purpose.

8.2 Differential GPS

Design considerations on the GPS module — using Differential GPS.

8.2.1 Problems faced

Due to military demands the GPS signal is distributed with "Selective Availability." It makes it impossible to get the accuracy of the received coordinates better than approximately 100 meters.

8.2.2 Possible solutions

It is necessary to use an additional Differential GPS (DGPS) signal to obtain sufficient accuracy (approximately one meter).

The Garmin board can use a Differential GPS (DGPS) signal to bring the error of the produced coordinates down to one meter. This D-signal can be retrieved from a public accessible mobile-phone number, and then has to be sent to the Garmin board.

8.2.3 Considerations

In order to get sufficient accuracy for our purpose we need to use the DGPS signal.

This in return adds more complexity to the system design: It made it necessary to maintain yet another "open line" to the GPS receiver at the mobile unit through which the DGPS signal continuously was fed.

8.2.4 Solutions chosen and why

It is chosen to make a DGPS server and let the GPS module subscribe to it, in spite of the additional complexity it would introduce: Without it the GPS coordinates are simply *too* inaccurate.[2]

The DGPS server was a stationary server placed at the CPK department. It dials up the DGPS-service which once a second send the current DGPS correction. A DGPS-event is generated, containing the correction. The GPS server feeds this correction directly to the Garmin unit which in return produces corrected GPS coordinates.

[2]The actual DGPS-capable receiver however arrived at a very late time in the project period. The user tests hence were carried out using a simulator.

GPS to Map

CHAPTER 9

9.1 Manipulation of coordinates

Design considerations on the Java `Position` class containing the GPS coordinates.

9.1.1 Problems faced

The Position class contains the GPS coordinates for the specific position (of e.g. places). Should this object representing a set of coordinates also offer methods for converting the GPS coordinates into map coordinates?

9.1.2 Possible solutions

- Let the coordinates object perform the conversion.
- Keep the knowledge of how the maps and the map coordinates are constructed in the map-related objects - that is, perform the conversion in the MapDisplayer object.

9.1.3 Considerations

Although it is "nicer" (in the correct OO-spirit) to keep the manipulations of the coordinates in the coordinate object, it will divide the knowledge of how maps are represented and calculated in two different places.

9.1.4 Solutions chosen and why

The knowledge of maps, their representations etc. will be kept together in the MapDisplayer module, in order to make it easier to maintain and change the implementation of the maps. The MapDisplayer will then be given GPS coordinates and must convert them itself into map coordinates.

9.2 How to scale the map correctly

Design considerations on map scaling.

9.2.1 Problems faced

The map displayer module gets GPS coordinates (latitude and longitude) from the GPS module, but does not know where to place them on the map.

9.2.2 Possible solutions

In order to be able to scale the map correctly, the map displayer module needs to know either one GPS coordinate for a point on a map and a scaling factor - i.e. how far a point must move on the map when it moves for instance, one degree in "reality" - or it must have the GPS coordinates for two points on the map.

9.2.3 Considerations

We have been given the exact latitude and longitude for two points on the campus area by respectively Kai Borre of the GPS department and Karsten Jensen of the land-surveyor's education.

The coordinates of those two points are very accurate!

9.2.4 Solutions chosen and why

We choose to scale the map by using the two coordinates supplied by Kai Borre and Karsten Jensen, as they are very accurate, and we can find the points on the campus map. The map displayer module is then given the two points (by clicking on the map) and the corresponding GPS coordinates are entered.

9.3 Choosing map scaling points

Design considerations on how to scale the maps correctly.

9.3.1 Problems faced

We have two points on the campus area for which we have been given very accurate GPS coordinates (by Kai Borre and Karsten Jensen). One of the points is however too far away from our test-area to be on the part of the map chosen for the test.

9.3.2 Possible solutions

It is possible to use the two provided GPS coordinates and the full campus map to calculate a local scaling factor[1] - i.e. how many meters one degree is north-south and east-west.

Another possibility is to create a new point on the map *inside* our test area, measure the distances to the two known points, and use it all to calculate the GPS coordinates for that new point.

9.3.3 Considerations

Using the printed version of the map (as printed in the folder "Map of Aalborg University") for measurements and calibrations may not be the optimal way in terms of accuracy. But it will be sufficiently accurate for our purposes. (And also the error introduced by this method will be smaller than the error in the GPS signal.)

9.3.4 Solutions chosen and why

To make the calibration of the map as easy as possible we chose to create a new "known" point. Assuming valid GPS coordinates for a point inside the test area simplifies the rest of the map-calibration process.

[1] It will only be a *locally* valid scaling factor, because the "width" of a degree changes as you move along a north-south axis, whereas a degree *in* the north-south direction always is the same number of meters.

9.4 GPS to map

Design specification on how to map GPS positions into map coordinates.

9.4.1 Problems faced

To make the mapping from the GPS positions to a point on the map we need some kind of transition between the two coordinate systems. We need to calculate this transition somehow.

9.4.2 Possible solutions

The mapping to and from GPS position can be done by specifying points on the images with known GPS positions. If two position are known then the scale and the translation between the two coordinate systems can be found. If more points are known then Rotation can also be calculated.

The normal way of translation between two coordinate systems is show in equation 9.1.

$$\begin{bmatrix} x_{gps} & y_{gps} & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} x_{image} & y_{image} & 1 \end{bmatrix} \tag{9.1}$$

If three points are known in each coordinate system the matrix can be found by solving six equations with six unknowns.

$$\begin{aligned} a \cdot x_{gps1} + c \cdot y_{gps1} + t_x &= x_{image1} \\ b \cdot x_{gps1} + d \cdot y_{gps1} + t_y &= y_{image1} \\ a \cdot x_{gps2} + c \cdot y_{gps2} + t_x &= x_{image2} \\ b \cdot x_{gps2} + d \cdot y_{gps2} + t_y &= y_{image2} \\ a \cdot x_{gps3} + c \cdot y_{gps3} + t_x &= x_{image3} \\ b \cdot x_{gps3} + d \cdot y_{gps3} + t_y &= y_{image3} \end{aligned}$$

9.4.3 Solutions chosen and why

We decided first to make an algorithm that only calculates the scale and the translation, this means that equation 9.1 can be simplified to the problem of solving the following equations:

$$\begin{aligned} s_x \cdot x_{gps1} + t_x &= x_{image1} \\ s_y \cdot y_{gps1} + t_y &= y_{image1} \\ s_x \cdot x_{gps2} + t_x &= x_{image2} \\ s_y \cdot y_{gps2} + t_y &= y_{image2} \end{aligned}$$

We then end up with:

$$\begin{bmatrix} x_{gps} & y_{gps} & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{x_{image2}-x_{image1}}{x_{gps2}-x_{gps1}} & 0 & 0 \\ 0 & \frac{y_{image2}-y_{image1}}{y_{gps2}-y_{gps1}} & 0 \\ x_{image1} - \frac{x_{image2}-x_{image1}}{x_{gps2}-x_{gps1}} \cdot x_{gps1} & y_{image1} - \frac{y_{image2}-y_{image1}}{y_{gps2}-y_{gps1}} \cdot y_{gps1} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} x_{image} & y_{image} & 1 \end{bmatrix}$$

This equation is implemented along with an inverse version of it. The inverse transition matrix is used in the `MapEditor` to calculate the GPS position from a pixel coordinate.

Maps

CHAPTER 10

10.1 Map

Design specification on the map class.

10.1.1 Problems faced

Our system needs to show maps of the user's current position, to do so we need access to a database with maps. Unfortunately we are not able to use many of the maps available in these kind of databases because they are usually stored as a bitmap picture with no information on what they are showing (e.g. roads). For the system to be able to find routes on the maps we need more information. Another problem is that the system has a limited amount of storage so keeping the whole world map in the system is not an option. This means that we need a way of representing the maps as pictures of the area together with the road information, making it possible to change the maps on the fly.

This raises a lot of problems:

1. How do we make the connection from a map to the "Real world"?
2. Do we need to rotate and scale the maps?
3. How do we make the transition between maps (smooth/switch)?
4. Is the route finding implemented on each map?
5. How do we find routes across different maps?
6. Do we want to have different kinds of maps (bitmap or vector)?

The problem in mapping coordinates from the real world to the local map is described in chapter 9.

We want to have map objects that are fairly easy to change while the system is running, and without restricting us to one type of map. Therefore the map should have a functions that can do things like:

- Convert a position in the global coordinate system to a local position on the map (pixels).
- Check if a given position can be shown on the map.
- Get the road information on the map.
- Get the map as an image.
- Scale and rotate the map.

10.1.2 Solutions chosen and why

The rotation of the map is only relevant if we know the orientation of the user (with a digital compass), and since we don't have this available right now we are not going to implement any rotation function in the maps. The scale function is also left out of the system because it's not an important part of the system (may be implemented in version 2.0).

When the user is moving from one map to another we have to get the new map and start showing that instead. The transition between maps can be done by smoothly going from one to the other or by a quick switch. The Smooth solution is the best, but the switch is the quickest to implement and is still functional so this is the one that will be implemented.

We don't want to make routes across different maps, because the route is just meant as a suggestion and has to be calculated fast. Making the routes go across several maps can make the route finding quite complex.

10.2 Route finder

Design specification on the route finding algorithm.

10.2.1 Problems faced

For the system to be able to finde routes on a given map, we have to finde a way of representing the roads on the map. We have chosen to represent the roads as an undirected graph where roads are nodes with connections to other nodes (see figure 10.1). We then have to find a path between two nodes in this graph.

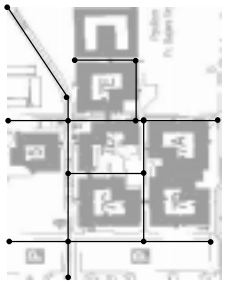


Figure 10.1: Images with the road map (nodes with connections).

10.2.2 Possible solutions

There are many different types of graphs and algorithms, some of them can be found in Algorithms in C++ by Robert Sedgewick.

10.2.3 Considerations

To find routes on a map like this there are a lot of different algorithms. But we need to focus on computation speed of the algorithm which means that all "shortest path" algoithms are out. Since the route is just a suggested one we decided that we don't have to make a suggestion if the route is too long.

10.2.4 Solutions chosen and why

The algorithm that we ended up with is a Direction Guided Depth Search. It is direction guided in the sense that the distance to a node is multiplied by a penalty, and this penalty is varied by the angle between the node line and the direct line to the target node. The penalty is accumulated in each path and the path is discarded if it has a penalty is above a defined value.

The angle ϕ between the "Direct line" and the "Next node" (see figure 10.2) can be written as:

$$\phi = \arccos\left(\frac{a^2 + b^2 - c^2}{2 \cdot a \cdot b}\right)$$

The penalty is then calculated by the formula:

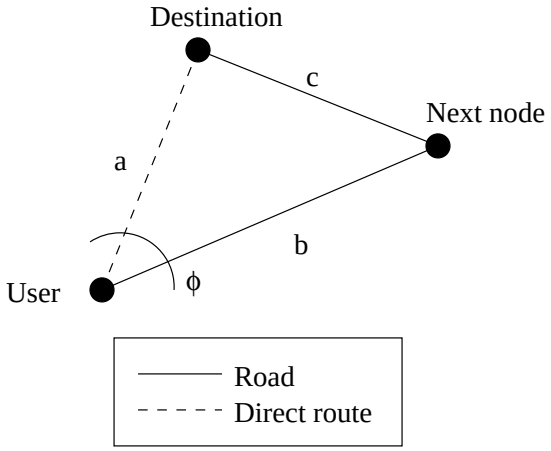


Figure 10.2: A small graph with three nodes. a, b and c represents the distance between the nodes.

$$penalty = penalty + \phi \cdot \frac{2 \cdot b}{\pi}$$

These formulae are implemented in the current system.

10.3 Map Editor

10.3.1 Problems faced

To test the system we needed a map with roads on. Constructing a graph by hand would be trivial work but take some time.

10.3.2 Possible solutions

We decided to make a small map editor. The specification for it was:

Load picture: It had to be able to load a picture (jpg/gif) that could be used as a background map.

Input GPS points: To make the transition between the GPS coordinates and a pixels position we need to specify two pixels' GPS positions. There should be a way of doing this in the editor.

Building the graph: We need to have functions to add and remove nodes and to connect them together to form a road-graph.

Load and save: The map editor has to have a function to save the map, and it could be nice if it could be able to load an already created map.

10.3.3 Considerations

In the `Map Editor` we are able to make *nodes* with no connection to other *nodes* which is not allowed in the system. This is because the routefind algorithm will find the nearest node to the user and try to find a route from this node to the target. If the nearest node doesn't have any connection there will be no route. The same problems can occur if there are groups of nodes that are unconnected to the rest of the graph (see figure 10.3).

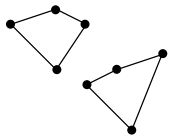


Figure 10.3: This shows a graph with two groups of nodes that are not connected together.

10.3.4 Solutions chosen and why

We ended up with a map editor that implemented all the basic functionalities, but contained no error checking of the graph. Figure 10.4 shows the `EditMap` program displaying the map that was used in the system.

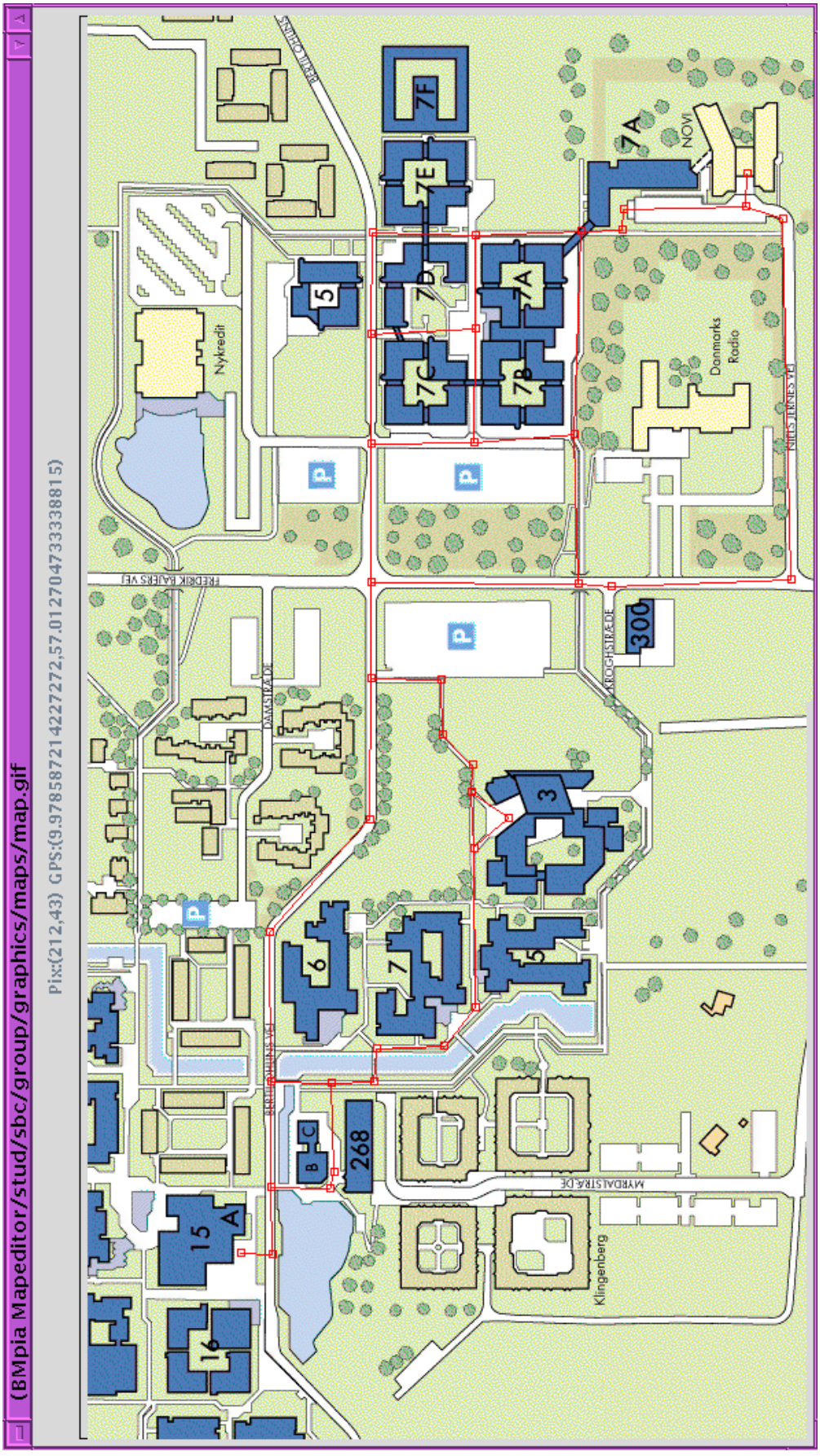


Figure 10.4: The map used in the system with the road graph.

Display

Chapter 11

11.1 Map Displayer

Design considerations on the Java `MapDisplayer` class.

11.1.1 Problems faced

To design a unit capable of displaying the map with a route in two ways: When the user is moving towards a target and when the user is following a tag (i.e. both ends of the route are moving).

11.1.2 Possible solutions

The Map Displayer should register itself as an event-listener with both the GPS module and the Tag Server.

11.1.3 Solutions chosen and why

It seems reasonable to use the scheme outlined above, as it constitutes a clean interface between the different parts of the system.

Hardware

CHAPTER 12

12.1 Linux Setup

In this worksheet we describe some of the problems we encountered when we setup the laptop.

12.1.1 Problems faced

The laptop we got from Bosch was a new laptop with the following specifications:

- Model: DELL latitude CPi D233ST 512KB.
- System chip: Intel Mobile 430TX PCIset.
- PCMCIA: 16 bits.
- Memory: 64MB
- Serial: 16.550 compatible, 16 bytes buffer connector.
- Audio: Crystal 4237B
- Video controller: NeoMagic 2160.
- Video memory:2MB
- Netcard: PCMCIA 3com 3c_589_cs

The laptop ran using Linux Redhat release 5.1 (Manhattan) which comes with kernel version 2.0.34. This kernel supported the netcard but not the sound card.

The first problem we encountered was that no specification was released of the NeoMagic 2160 chipset. This meant that open-source X servers like XFree86-SVGA (version 3.3.2-8) didn't support this chipset.

After surfing around for a while we found another Xserver which supported an unaccelerated version of a slightly older version of the NeoMagic chipset. This implementation is based on the XFree86 3.3.2 link kit. Unfortunately we only got it to run a resolution of 800x600 in 8bpp but it would do.

To get the sound to work we had to recompile the kernel. Unfortunately there were some problems with getting the sound card to work under kernel 2.0.34, so we upgraded the kernel to kernel 2.0.35. Here there were problems with the netcard because of in compatible PCMCIA driver versions. In linux the PCMCIA source is NOT distributed along with the kernels but there is a unique version for some kernels. Finally by reading a lot of HOWTOs and kernel documentation we figured out how to compile and install

new PCMCIA drivers. We ended up with a system running kernel-2.1.129 and PCMCIA-CS-3.0.6.

The network setup caused some days of total frustration, mainly because of the Point-to-Point Protocol (PPP). When we got the pppd to dial up to Aalborg University's modem pool (AUA) and make a ppp connection we encountered new problems. There were no problems in pinging our dynamic IP address and AUA's domain name server i.e. the connection was established. However, it failed to make a telnet to trabant.kom.auc.dk (130.225.51.15). This was because eth0 (Ethernet card 0) was used as the default gateway and that was setup to use static-routes:

```
eth0 net 130.225.49.0 netmask 255.255.255.0 gw 10.225.49.1
eth0 net 130.225.51.0 netmask 255.255.255.0 gw 10.225.51.1
eth0 net 130.225.50.0 netmask 255.255.255.0 gw 10.225.50.1
```

This meant that all unknown IP requests ware forwarded to these gateways, and not to the gateway provided by AUA. The fix made was fast and not so nice. We just made small script that could remove and install all the config files needed to use the kom network. To make a ppp connection we first had to remove the `kom` settings and then start the pppd; when we needed to connect the laptop to the `kom` net we just reinstalled the old config files.

12.2 The serial port

Design specification on how to use the serial port (COM-port) under Linux.

12.2.1 Problems faced

On a PC running Linux the serial port can be opened from Java as a normal file for reading and writing (`/dev/ttySn`, where `n` is 0 for COM1 and 1 for COM2). This approach lacks the ability to manipulate the communication-settings for the port, e.g. the parity, stop-bits, baud-rate etc.

12.2.2 Possible solutions

The port-settings can be done "manually" using the `minicom` program from a prompt. It provides a menu-based interface to all the settings of the serial port.

Another approach is to use a C program and the Java Native Interface (JNI) to alter the port settings.

12.2.3 Considerations

The `minicom` program is a small but nice communication program that gives good overview over the current port settings and even displays the data being received on the port.

Using native C code makes it possible to just start up the entire program without having to worry about the port setting - it is done on the fly.

12.2.4 Solutions chosen and why

The problem of controlling the port settings was solved by using JNI and a C program doing the port-setting. This was chosen because it would allow the system to be started up by a single command. The overview and additional information provided by the `minicom` program is only useful in the initial phases of development.

The GPS unit is set to transmit the current GPS position to the Java GPS module once a second. If the position has moved more than a defined limit, the GPS module generates and broadcasts a `NewPositionEvent` to all the modules that have registered themselves as listeners for this event-type. The GPS module thus acts as a server for all the modules that wants to subscribe to GPS events.

The GPS server itself subscribes to the DGPS module, receiving DGPS-events once a second.

Chapter 13 Test

13.1 Model

The model shown in figure 13.1 is used at Bosch telecom for product development to show the relevant procedures that a product has to go through before it can be put in production. We used this model as a guide line in the development of our system. Not all phases in the model were used (e.g. Preproduction phase).

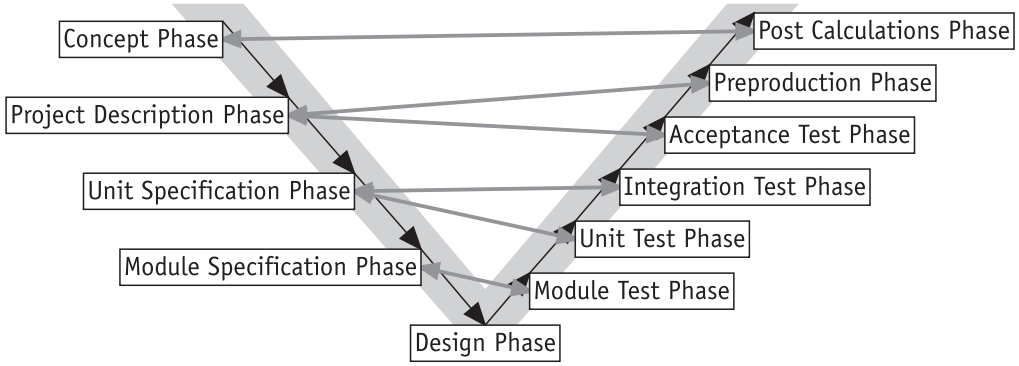


Figure 13.1: Integrated product development.

13.2 Test

The system was tested in four different ways according to the model showed in figure 13.1.

13.2.1 Module and Unit test

The module test was a running evaluation and therefore done while the module was being implemented. A black box test (Unit test) concentrated on testing the module's interface specification, to make sure that each module implemented all the functions in its interface and that it responded in the expected way. This test also made sure that the module was able to run and communicate on different computers and architectures (RMI test). The variables in the DSS had to be configured manually to respond in a proper way.

13.2.2 Integration test

In the integration test we started the system and tried to use the system. We tested that all parts of the system were running and communicating with each other. The first test was made with all modules running on the same computer, and when that was working we moved some of the modules to different computers to test that the system really could run fully distributed.

13.2.3 User test

To evaluate our system we performed a user test of our system. Each user was given a task to achieve using the system. Their actions and those of the system were recorded in a dialogue history while they carried out this task. The users were then interviewed about their views on various aspects of the interaction and asked about their opinions of the system as a whole.

After the test session each user was interviewed. The interview consisted of several open questions designed to give us feedback on the system by ascertain whether they liked the interaction; whether they ever lost track of the dialogue; which display items they found most useful and which not useful; what they thought of the cartoon creature; and whether they considered the system useful.

User profile	
Sex	Age
Female	25
Spoken dialogue exp.	Mobile phone
A lot	yes

System shows	User says	System recognised
Top dialogue	"show me a restaurant nearby"	*misrecognition*
Top (Help)	"show me the nearest places"	nearest
Nearest dialogue	"restaurants"	restaurant
List of near restaurants	"I want restaurant number one"	one
Showing the map.	"Ooh - cool'n'funky!!"	*misrecognition*
Map (Help)	"Cancel"	cancel
Time: 35		
Top dialogue	"Nearest"	nearest
Nearest dialogue	"Restaurants please"	restaurant
List of near restaurants	"five"	five
Showing the map.	"Cancel"	cancel
Time: 15		
Top dialogue	"Nearest"	nearest
Nearest dialogue	"Restaurants"	restaurant
List of near restaurants	"five"	five
Showing the map.	"Cancel"	cancel
Time: 14		
Top dialogue	"Nearest"	nearest
Nearest dialogue	"Restaurants"	restaurant
List of near restaurants	"one"	one
Showing the map	"Cancel"	cancel
Time: 14		

Table 13.1: *System log from user number 1.*

User profile	
Sex	Age
Male	27
Spoken dialogue exp.	Mobile phone
very little	yes

System shows	User says	System recognised
Top dialogue	"I would like to go to a restaurant"	*misrecognition*
Top (Help)	"Show me some restaurants"	*misrecognition*
Top (Help)	"Help"	help
Top (Help)	"Show me the nearest restaurants"	nearest
Nearest dialogue commands	"restaurants"	restaurants
List of near restaurants	"Fib 15"	*misrecognition*
Restaurants (Help)	"number three"	three
Showing the map	"Hmm then what?"	*misrecognition*
Map (Help)	"Help"	help
Map (Help)	"cancel"	cancel
Time: 50		
Top dialogue	"Nearest"	nearest
Nearest dialogue	"show me a restaurants"	restaurant
List of near restaurants	"I like NOVE"	*misrecognition*
Restaurants (Help)	"number two"	two
Showing the map.	"Cancel"	cancel
Time: 23		
Top dialogue	"Show me a nearest restaurant"	nearest
Nearest dialogue	"Yes a restaurant "	restaurant
List of near restaurants	"I would like to go to number one"	two
Showing the map	"What! This is not right!"	*misrecognition*
Map (Help)	"Cancel"	cancel
Top dialogue	"nearest"	nearest
Nearest dialogue	"a restaurant "	restaurant
List of near restaurants	"number one"	one
Showing the map	"Cancel"	cancel
Time: 36		
Top dialogue	"Nearest"	nearest
Nearest dialogue	"Restaurants"	restaurant
List of near restaurants	"two"	two
Showing the map	"Cancel"	cancel
Time: 15		

Table 13.2: *System log from user number 2.*