CoINS: Context sensitive Indoor Navigation System

Fernando Lyardet* Jan Grimmer * and Max Mühlhäuser
Darmstadt University of Technology
CS Department - Telecooperation
Hochschulstrasse 10, 64289 Darmstadt, Germany
fernando@tk.informatik.tu-darmstadt.de
Also at SAP Research, CEC Darmstadt

Abstract

The Context sensitive Indoor Navigation System (CoINS) implements an architecture to develop context-aware indoor user guidance services and applications. This paper presents a detailed discussion on algorithms and architectural issues in building an indoor guidance system. We first start with the World Model and required mapping to 2D for the process of path calculation and simplification. We also compare several algorithm optimizations applied in this particular context. The system provides the infrastructure to support different techniques of presenting the path and supporting user orientation to reach a certain destination in indoor premises.

1 Introduction

Context aware systems seek the integration of sensed and derived data in order to situate user activities and provide a more meaningful interaction with information systems. The use of contextual information opened a new range of applications for supporting the user in working scenarios such as offices and manufacturing, or in hazardous environments such as construction. An important topic in context aware systems is the location of the user and how to take advantage of it. However, both sensing location and its application have proven to be challenging. In recent years, a number of technologies have been developed in order to solve the sensing problem, since GPS is not suitable for indoor use. The currently available technologies for sensing location include infrared [1], Radio Frequency [19], ultrasound [17], and even magnetic fields [12]. These different technologies are rather complementary than interchangeable since their own properties make each of them suitable for different scenarios. The user location information has been exploited for augmenting reality with information and provides user guidance in exhibition settings, and to detect whether a user is at a particular room or in front of a particular object [16]. Other applications of indoor location information have been envisioned, although up to now, no general solution approach was available to provide such services in multiple scenarios. This is the case of using indoor location information for assisting users navigating within indoor scenarios in either private or public areas. A few examples of such scenarios are large corporate campuses where employees walk between different offices as a part of their work, and airports where passengers must arrive on time at their departure gates. Other examples are construction, mining workers and firemen in restricted and dangerous settings, or even museums where visitors would like to walk through its collection following different criteria such as period, technique or style. We present in this paper CoINS, a Context aware Indoor Navigation System, which implements a set of services for supporting user guidance in indoor premises. In the following sections we will present the different technologies involved, a detailed architecture of the system, as well as a detailed discussion of the navigation resolution algorithm. Finally we also outline our ongoing work in this area.

2 CoINS Overview

The Context Indoor Navigation infrastructure has been developed in order to provide user support in working environments. The architecture should be flexible enough to different guidance alternatives according to the user experience, situation and profile that would better take advantage of the user's cognition resources. Furthermore, clear extension mechanisms should be in place in order to accommodate ever increasing contextual sources of information and user adaptation functionality. Therefore the CoINS infrastructure has been developed under a service-oriented architecture that enables the deployment of such services either at the user's device or anywhere in the network. This approach, together with the ever increasing number of computers available, allows a growing instrumentation of both living and working environments. Such expanded computational capabilities can now be used to balance the trade-off between real world operation complexity and user simplicity in favor to the users. The CoINS architecture was developed with this ubiquitous computing notion [19] in mind.

2.1 A General Approach for Indoor Guidance

Providing indoor guidance requires a proper representation of the environment or *"world model"*. Guiding a user through a building not only requires information about which rooms exist and different ways to identify them, but the application also has to know about the positions of these rooms relative to each other to calculate a path from the user's current location to his or her destination. Additionally, when descriptions like *"the cup on your left"* or the *"XX machine behind you"* shall be generated, the system needs information about the position of different objects. Many scenarios do not require such level of definition. Furthermore, in many cases, such information is not available anyway. Therefore, the system further decouples guiding functionality into 2D path generation and user orientation. This further expands information sources to other industry standards such as DXF [8].

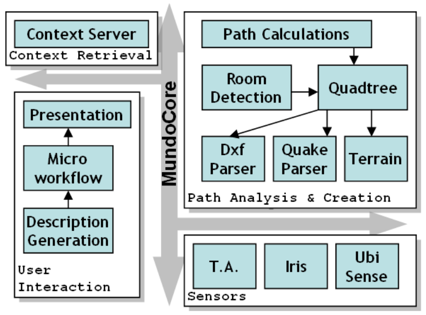

Figure 1. Overview of System Components

The description of a path between two locations should be *"abstract"* and *"minimal"*. By *"abstract"* we mean without any binding to a particular modality of conveying information to the user. *"Minimal"* is a desired property, where the description contains no superfluous or repetitive descriptions in order to enable afterwards a more accurate generation of directions in any modality. Once the path is calculated, the system must finally interact with the user. However, in order to support future modalities, CoINS integrates a separate workflow description of the whole activity of user guidance. CoINS allows a novel degree of configuration of how the overall user guidance support must operate, by decoupling the overall process of guidance from the path calculation, modality and user description subsystems.

3 World Model Description

The CoINS System has been designed to be embeddable, extensible, uses standard interfaces. To provide guidance between two locations, only 2D representations are required, although the software is able to convert 3D representations into 2D as well.

3.1 Describing the World as Quake III maps

CoINS makes use of the Quake III format [10, 18] for the world descriptions, since it is widely known and open source tools are available to create world descriptions. Quake III maps basically consist of two kinds of components: "brushes" and "entities". Brushes are arbitrary convex shapes. Especially walls, floors, and ceilings are modeled that way. Textures can be put on brushes to describe the looks of the object in the game. For instance, textures exist to make a brush look like a stone wall or a metal ceiling. However, these are only relevant for use in the game and are not needed for CoINS. The World thus consists of "brushes" (convex objects) and each side of a brush is indicated by three points. These points can be arbitrary points on the infinite extension of the surface area generated by the intersection of these planes. The geometry is calculated then in three steps:

1. The infinite plane described by each triangle is calculated and represented in its coordinate form which consists of a vector $\vec{n}$ which is the normal of the plane (that is, "standing" at right angle on it) and a constant d such that $\vec{n} * \vec{p} + d = 0$ for each point p on the plane.

2. The intersection of each pair of planes is calculated. If such an intersection exists, it is described by a straight line as a pair $(\vec{o}, \vec{d})$ of vectors such that $\vec{o} + r * \vec{d}$ is a point on the line for each $r \in \mathbf{R}$.

3. The intersections between these lines and the planes are calculated, resulting in eight points which are the corners of the brush.

It would have been possible to use the compiled BSP files instead. Alas, restoring the original geometry from this information would be computationally expensive and would not offer any additional features. We have used the GtkRadiant modeling kit [11] to create world models for CoINS. However, the maps created are not automatically interchangeable between the game and CoINS: entities are not solid in Quake by default, and measuring units are not the same. One unit equals one cm in CoINS but one inch in Quake. This means that the map should be scaled by a factor of $\frac{1}{2.54}$.

For CoINS, an additional entity *wm_door* was added. As the name implies, it is used to identify doors or, more generally, openings in walls through which the user can go from one room to another. This is needed for identifying separate rooms and building the search graph for path calculation. The process is explained in the following section.

3.2 Mapping into 2D and Room Detection

When guiding the user through an area, frequent path re-calculations are needed. This process would be very time-consuming if the calculation was based on the three-dimensional model, which usually cannot be afforded in an embedded environment. Since Java3D is not fully supported on J2ME environments, these calculations have been re-implemented. A 2D representation has been used to perform the calculations, since these are much easier and faster in 2D space. In particular, an Octtree instead of a Quadtree would have been needed if a 3D space representation were used, resulting in an increased computational cost. In most cases, users will mostly move in two dimensions. Movements in the height dimension usually only occur when changing floors, which could be modeled by using separate maps for each floor.

3.3 Mapping a 3D model into 2D

The current approach implemented in CoINS is as follows:

1. A plane is created which is parallel to the ground. The z-value (that is, the height of the plane) is currently at 160 cm, which is approximately the height of the eyes of an average person.

2. The intersections between this plane and every "element" of the world (walls, doors, objects) is calculated, resulting in a set of polygons each of which is described by a list of points.

3. These polygons still have three dimensions. However, since the plane was created at constant height, all polygons, or rather all points describing the polygons, have the same height value. By ignoring this constant z-value, these three-dimensional polygons are transformed into 2D-space.

4. The points of each 2D polygon are sorted clockwise by calculating their convex hull which is required to do any further calculations and draw the polygons in the GUI.

3.4 Data Structures for Room Detection and Path Creation

3.4.1. The Convex Hull. A convex hull [6] for a set of coordinates or points is the minimal polygon containing all these points.

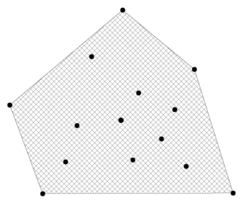

Figure 2. A convex hull

It is calculated with the Jarvis March algorithm [7]:

1. The algorithm starts by selecting a point p_0 which is guaranteed to be on the hull. CoINS does this by selecting the point with the minimal x coordinate. If several points with that x exist, the one with minimal y is selected from these.

2. Then, it creates straight lines between the selected point p_0 and each other point in the set. The point p_1 with minimal angle from the y-axis to the connecting line $p_0 \rightarrow p_1$ is the next point on the hull.

3. Analog to step 2, the point p_{i+1} is selected as the point with the minimal angle between the line $p_{i-1} \rightarrow p_i$ and $p_i \rightarrow p_{i+1}$.

4. Step 3 is repeated until p_0 would become selected again as a new point on the hull.

3.4.2. Quadtree. The two-dimensional data generated as described in the previous section consists of an unordered set of polygons. Areas that are within the bounds of the map but not covered by a polygon are implicitly considered empty. However, for further calculations this representation is insufficient. To calculate paths and determine which coordinates belong *"together"*, that is, are in the same room, it is necessary to create some kind of raster data. A possible solution could be using a two-dimensional array of data representing the respective *"terrain type"*. Each cell of the grid would be represented as one element of the array. However, the question for an optimal resolution arises. Representing each single pixel (which is an area of 1x1 centimeter in our case) is clearly overstated and would result in enormous calculation time when calculating areas and paths. On the other hand, a grid too large is too imprecise and undesirable as well. Usually, in large open spaces a larger grid should be used than in smaller corridors. Especially when the system needs to calculate if the user has left the correct path, it is desirable to determine the distance of the user from the path in terms of *"grid areas"* instead of centimeters. A grid too small would lead to false positives (that is, the user is only a few centimeters away from the path but the system would interpret it as too far away) while a grid too large would lead to false negatives (for example, the user might already be in another room but still in the same area of the grid). Therefore, a grid that adapts to the detail level of the terrain is desirable. A common data structure that offers these features is the **quadtree** [13]. Quadtrees are a concept from computer graphics where they are used for efficient storage of image data [20], which is a similar problem to our terrain representation. Furthermore, calculations like efficient neighbor finding (described later) can be done on quadtrees with relatively little effort. Therefore, the unordered set of polygons which resulted from the transformation process in the previous section are now converted into a quadtree structure.

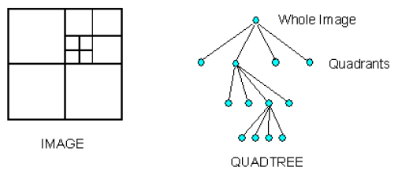

Figure 3. Quadtree example

Quadtrees are trees in which each node can have up to four children, and they are built in the following way:

1. The image is separated into four quadrants (called *NW*, *NE*, *SW*, and *SE* here).
2. If a quadrant is not "homogeneous", that is, all pixels have the same color, it is split further.
3. If an area is split, the sub-areas are added as children to the node representing the larger area.
4. The splitting process stops when all subregions are homogeneous or a specified resolution or tree height has been reached.

Through the process described, quadtrees offer a variable resolution down to pixel resolution which can be changed at runtime by expanding further nodes.

3.4.3. Neighbor Finding. For path calculations and room detection, it has to be known which areas are adjacent to a given area, that is, which are their respective neighbors in space. Unfortunately, adjacency in space does not relate directly to a simple relation of the node positions within the quadtree. It is possible however, to find the respective neighbors by means of tree search. One commonly used algorithm was proposed by Samet [13].

3.5 Room Detection

For context sensitive navigation, it may sometimes be necessary to identify which areas (that is, grid cells) belong to one room. For instance, a description like *"Go to the kitchen"* might be useful. In this case, new activity descriptions may be triggered not by passing certain waypoints among the path but by entering a cell that belongs to the mentioned area, the kitchen in this case. Determining the boundaries of an area seems simple at a first glance, but it is not. Especially if doors were simply modeled as notches in the walls, as the Context Server originally did, it is not always unambiguous to which room an area belongs to. To be more precise, it is not easy to determine if two directly adjacent cells also belong to the same room. Figure 3 shows the situation.

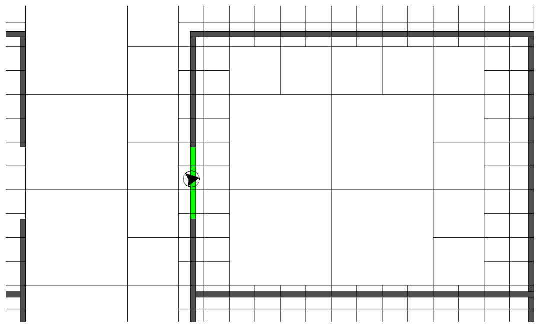

Figure 4. Ambiguity Problem and Solution Approach

Two questions arise here: first, it is not simple to determine which room the user is in. Second, even if the cells that are ambiguous for a human are ignored and only the cells that could easily be assigned to a room by a human eye, this is not as easy for a computer. An additional separator is needed to determine the exact boundaries of a room.

4 Path Calculation

All data structures described are required to support the path calculation process. A number of general algorithms have been published that solve this problem. The probably most widely known one is the one of E. Dijkstra [3]. The basic algorithm has a complexity of O(n^2). Several optimizations have been introduced since then, and we have measured several of them in order to determine the most suitable for our purpose. The optimizations include early termination, Bidirectional Dijkstra, and Goal Directed Search.

4.1 Comparison of Path Calculation Approaches

We realized a comparison to determine the optimal search algorithms for CoINS, selecting four Dijkstra variants. These algorithms have been tested and compared using a sample structure of 3328 nodes and 1000 arcs.

Algorithm	Average time (ms.)
Early Termination (E.T.)	968.146
Bidirectional	1 110.290
Goal-Oriented with E.T.	747.240
Bidirectional, Goal-Oriented	804.118

Table 1. Comparison of Optimizations.

We have used the Building of the CS Dept. of Darmstadt as a demo scenario with a grid size of 80 x 80 cm, 1000 random pairs of nodes (source, destination) were created (only node pairs where the destination could be reached from the source were allowed). Afterwards, the path from source to destination was calculated using Dijkstra with Early Termination, Bidirectional Dijkstra, Goal-Oriented Dijkstra with Early Termination, and Bidirectional Goal-Oriented Dijkstra algorithms on a notebook PC (Intel Celeron 1300 MHz). The average calculation times are displayed in Table 1. The plain Dijkstra algorithm without Early Termination and the Goal-oriented Dijkstra without Early Termination were not tried since they are provably never faster than the Early Termination variant. As can be seen in Table 1, the bidirectional variant of the algorithm does not pose an optimization against their single-directional counterparts. Even if the complexity might be lower in theory, the overhead for double management of visited nodes, predecessor tables and additional calculation of the abort condition (one node has been finally marked by both algorithms) after each step seems to outweigh the advantage of lower complexity.

4.2. Path Simplification

Figure 6 shows the result of a path calculation from a starting point (the point next to the arrow icon representing the user) and an end point. Each blue circle represents a way point among the path where the user would receive instructions.

It becomes obvious that the representation shown here is suboptimal for three reasons.

- Even when following a straight line, the number of waypoints at which the user receives instructions is way too high. This would result in a number of unnecessary "walk forward" instructions irritating the user. However, this problem could easily be solved by only giving instructions when the direction the user should change direction.

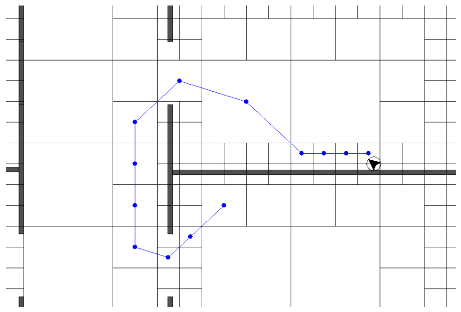

Figure 5. Path before simplification

- Since the algorithm uses the centers of the grid cells as waypoints and only allows connections between adjacent cells, the path is not totally optimal in terms of length as it would be if each pixel of the map were a waypoint candidate.

- For the same reason, there are a number of unnecessary waypoints where the user is given the instruction to turn. In the example above, users would expect to be guided directly to the door in a straight line instead of having to perform several turns.

For the reasons stated above, a simplification of the path became necessary. The first idea was just to remove the waypoints where no change of direction is necessary. This would have been a simple process solving problem number one, however the problems of optimal paths and unnecessary turns would remain. Therefore, another method was chosen. Figure 6 shows the result of the simplification process. The number of relevant waypoints has been reduced, resulting in straight connections even between cells that are not directly adjacent. Turns only become necessary where needed. The algorithm uses the quadtree structure to remove unnecessary steps and thus optimize the path as is described in the following. It is obvious that if a direct "line of sight" between two points, even not belonging to directly adjacent nodes, can be established, the user can walk from one point to the other without the need of intermediate waypoints. Thus, the superfluous waypoint can be removed from the path (the algorithm for calculating if a line of sight is not interrupted by any cell is explained below). Iteratively checking the next node if a line of sight can be established is of unnecessary complexity, especially if a large number of small cells are in a straight line.

Therefore, the next "relevant" waypoint is determined by using nested intervals. Two values `upper bound` and `lower bound` are used.

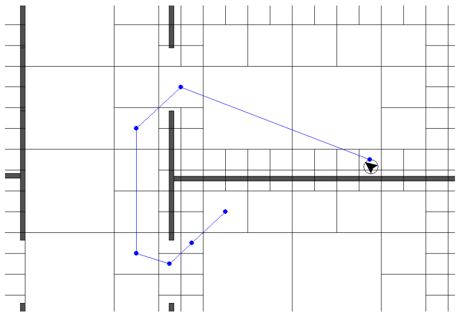

Figure 6. Path after simplification

1. The `lower bound` is set to the node directly following the start node in the path (which is always directly reachable).
2. The `upper bound` is set to the last node of the path, the destination node.
3. If a direct line can be created between the two points (that is, all `Terrain` objects of the intersected nodes return `true` for `isPassable()`), the calculation is finished.
4. If a direct line can not be created, a `candidate` in the middle of the path at $\frac{path.length}{2}$ is tried.
5. If a direct connection to the `candidate` can be established, the `lower bound` is set to the index of the `candidate`, otherwise the `upper bound` is set to the index of the `candidate`.
6. If `upper bound` and `lower bound` are not equal, the algorithms continues at step 5.
7. When `upper bound` and `lower bound` are equal, the algorithm is finished. All nodes between the last relevant waypoint and the `candidate` are removed and the algorithm continues with the old `candidate` as new starting point.

4.2.1. Line of sight calculation. Line of sight calculation, that is, determination if any obstacle is on the direct connection between two points, is not totally trivial. Of course, it would be possible to check if the line connecting the points intersects with any of the polygons from the `World2D`. However, since this had to be done many times (in the worst case, the number of iterations would equal the number of nodes on the path minus one multiplied by the number of polygons), this would be far too complex. Therefore, line of sight determination is done by using the structure of the quadtree.

1. The node representing the smallest area containing the starting point and the end point of the line is found. The algorithm recursively determines which child of a node contains both points. If the node itself but none of its children contains both points, it returns itself as result. Otherwise, the child node containing both points becomes responsible for finding the right node.
2. When the correct node has been found, the nodes that have to be checked for intersection can be restrained to its children.
3. The intermediate nodes, that is, the grid cells intersected by the line, are determined in a similar way. The node determined above checks which of its child nodes are intersected by the line.
4. If an intersected node is a leaf, it is added to the result collection, otherwise its children are checked and so on. That way, all cells intersected by the line are collected without having to check all cells in the map.

5 User Guidance

The CoINS system decouples the whole process of guiding a user by including a small, embeddable workflow engine called Micro workflow [9]. In this way, both the supporting services and the process of indoor user guidance can be further modified. Another advantage is that this approach also enforced further separations of responsibilities in order to allow a possible change of the workflow engine if required.

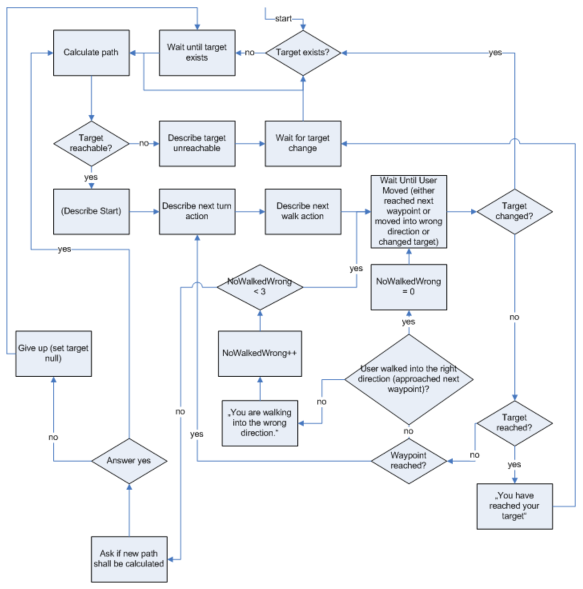
start
Calculate path
Wait until target exists
no
Target exists?
yes
Target reachable?
no
Describe target unreachable
Wait for target change
yes
(Describe Start)
Describe next turn action
Describe next walk action
Wait Until User Moved (either reached next waypoint or moved into wrong direction or changed target)
Target changed?
yes
NoWalkedWrong < 3
yes
NoWalkedWrong = 0
yes
no
Give up (set target null)
NoWalkedWrong++
User walked into the right direction (approached next waypoint)?
no
„You are walking into the wrong direction."
no
yes
Target reached?
no
no
Answer yes
no
Waypoint reached?
no
yes
„You have reached your target"
Ask if new path shall be calculated

Figure 7. Workflow Implementation Overview

One possible way of guiding the user is showing the path on a two-dimensional map. Most people will already be familiar with such a representation. This provides the user with a bird's eye view of a floor description showing the current location point followed by a trail of segments to be followed. The approach is rather simple to implement (we have also done it for testing purposes in CoINS), other authors have proposed improvements for the mobile setting[2, 4], and is much better than text to convey spatial directions [15].

However, not everybody is quick in interpreting floor plans, and a further cognitive load is put on the user to interpret the next step to take. Also, the approach is rather limited for further user adaptations. User adaptation should be done through different modalities as well as the directions conveyed to the user. As an example, consider the guidance needs for both local and visitor users. Local users know the environment and overall building structure. They also know other colleagues and where they are.

References to places, people, objects, and spatial landmarks [14] together with their spatial relationships can be helpful to build a mental trip on the guided user. Descriptions could also be decorated with information on whether the user has been already there and for which occasion. Descriptions also must vary following the progress of the user, increasing on demand and as the user gets closer to the intended locations and if further away from familiar places. On the other hand, visitors may need a more detailed description since references to people and areas may be meaningless. Furthermore, landmarks are restricted to what is visible for the user at the current location. A difficult aspect to study is how much description should be given to the user, to enable accomplishing the route and at the same time, avoid confusion and over-explanation, which in turn prevents the user from understanding the current situation.

6 Mobile Infrastructure

6.1 MundoCore Middleware and Software Architecture

MundoCore is a middleware that implements a Peer to Peer (P2P), publish-subscribe based communication infrastructure. Its purpose is to separate communication code from the application logic. It provides automatic node discovery for location transparency, that is, the programmer of an application does not need to know about the address of its communication partners (a server, for instance). It also supports an object serialization mechanism similar to Java Serialization. A Publish/Subscribe based communication paradigm is available as well as Remote Method Calls (RMC) based on it.

For the implementation of CoINS, we adopted a service oriented architecture because it allows deploying application components according to different scenarios and improve their reuse by other applications. The specific advantages of adopting a SOA architecture for CoINS can be summarized in 3:

1. Mobile environments differ in their computing capabilities. The implementation we have done using a PDA is perhaps the less restrictive in this kind of scenarios. However, since all components are implemented as stand-alone services we can easily reduce the available functionality at the mobile terminal to fit a more restrictive environment like a cell phone.

2. The SOA flexibility also reflects our view on how instrumented or ”smart” environments should grow: by deploying ad-hoc devices and services that can be combined and progressively improve the available functionality, as opposed to monolithic architectures and fixed, plan scenarios envisioned in a drawing board.

3. Besides its support to SOA applications, MundoCore provides a network communications overlay. This overlay allows a user to transparently continue accessing a service across the different WLANs that are usually available in large facilities.

6.2 The Talking Assistant

The Talking Assistant [1] is an audio-based device intended as a general purpose personal device for ubiquitous computing environments. It provides a well-defined minimal functionality which permits very small form factors, allowing the user to carry the device always with her. Audio devices do not need large displays to offer useful interaction possibilities, and through advanced manufacturing techniques, could be reduced to ear-plug size in a few years.

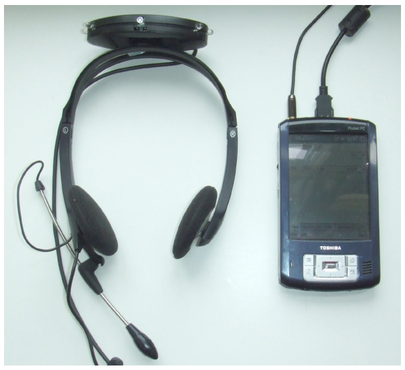

Figure 8. The Talking Assistant

The Talking Assistant is a Pocket PC based system that offers two basic functionalities: Firstly, it integrates a compass to determine the current head orientation of the user.

This information is shared with other devices, like the Context Server, via the MundoCore framework. Secondly, it provides a speech recognition and synthesis engine, so users can interact with the system via voice.

6.3 The Context Server

Applications requiring sensor data usually have to collect and process information on their own, which is an unnecessary effort, since handling has to be implemented in each new application. We have used a Context Server developed in our group, that takes on this work of sensor data collection and aggregation. Furthermore, Applications can request information from the Context Server via MundoCore Remote Method Calls or get notified via Publish/Subscribe. The Context Server offers a means to determine which objects are visible from the current location and head orientation of the user. For modeling the position of items in the three dimensional space, it uses the compiled form of the Quake III map format.

7 Conclusions and Future Work

We have developed CoINS, a service oriented, context aware indoor navigation system. During this work, we have implemented the algorithms and services required and we have presented a detailed discussion of its implementation. Further development is currently focused on developing a more general framework to address the problem of user guidance as in [2].

Finally, in section 8 we described part of our on-going work on a framework for multimodal user guidance. The CoINS system addressed in its architecture and implementation several challenges in providing this functionality both in a mobile and infrastructure settings. From the architectural point of view, it also introduces a novel degree of customization by externalizing the user guidance process into a workflow description, a degree of flexibility that makes CoINS more adaptable for a good number of scenarios.

We also want to continue improving the path calculation performance. In this area, a further optimization not discussed in this paper was attempted using the with JGraphT's Fibonacci Heap [5] implementation. This algorithm has a theoretical O($nlogn + m$), although the preliminary result showed an execution time of 691.327 ms, possibly due to some required optimizations missing.

8 Acknowledgements

The authors would like to express their gratitud to Dr. Gustavo Patow from the Grup de Grafics de Girona and Dr. Erwin Aitenbichler from the Telecooperation Group at TU-Darmstadt for their valuable comments.

References

[1] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser. Talking Assistant: A Smart Digital Identity for Ubiquitous Computing. In *Advances in Pervasive Computing*, pages 279–284. Austrian Computer Society (OCG), Austrian Computer Society (OCG), 2004.

[2] A. Butz, J. Baus, A. Krüger, and M. Lohse. A hybrid indoor navigation system. In *IUI2001: International Conference on Intelligent User Interfaces*, New York, 2001. ACM.

[3] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, (51):161–166, 1950.

[4] Fraunhofer. Messe Navigator. `<http://www.iis.fraunhofer.de/ec/navigation/indoor/projekte/messe/index_d.html>`, accessed at April 2006.

[5] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[6] M. Husty. Darstellende geometrie – technische mathematik. 2005.

[7] R. A. Jarvis. On the identification of the Convex Hull of a Finite Set of Points in the Plane. *Information Processing Letters*, (2):18–22, 1973.

[8] N. Johnson. *The coming age of calm technolgy*. McGraw-Hill, 1991.

[9] D. Manolescu. Workflow Enactment with Continuation and Future Objects. OOPSLA'02, ACM, November 2002.

[10] K. Proudfoot. Unofficial Quake 3 Map Specs. `<http://graphics.stanford.edu/~kekoa/q3/>`, March 2000.

[11] Radiant. `<http://www.qeradiant.com>`, March 2006.

[12] F. Raab, E. Blood, O. Steiner, and H. Jones. Magnetic position and orientation tracking systems. *IEEE Transactions on Aerospace and Electronics Systems*, 15(5):709–717, January 1979.

[13] H. Samet. The quadtree and related hierarchical structures. *ACM Computing Surveys (CSUR)*, June 1984.

[14] M. Sorrows and S. Hirtle. The Nature of Landmarks for Real and Electronic Spaces. In C. Freksa and D. M. Mark, editors, *COSIT '99*, volume 1661, pages 37–50. Springer, 1999.

[15] S. G. Towns, C. B. Callaway, and J. C. Lester. Generating coordinated natural language and 3d animations for complex spatial explanations. In *AAAI '98*, pages 112–119, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[16] R. Want, A. Hopper, V. Falco, and J. Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

[17] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personnel Communications*, 4(5):42–47, October 1997.

[18] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *Personal Communications, IEEE*, 4:42–47, 1997.

[19] M. Weiser. The computer for the twenty-first century. *Scientific American*, (7):94–100, September 1991.

[20] G. Z. Yang and D. F. Gillies. *Computer Vision*, chapter 4.