J Intell Manuf (2010) 21:133–144
DOI 10.1007/s10845-008-0165-6

A holonic architecture for the global road transportation system

Frédérique Versteegh · Miguel A. Salido · Adriana Giret

Received: 9 August 2008 / Accepted: 14 August 2008 / Published online: 12 November 2008

Abstract In recent years the development of automated traffic systems has been gaining increasing attention, and a substantial amount of effort has been invested in trying to find a solution to problems associated to road transport. Among these problems are road accidents caused by human-related factors, such as tiredness, loss of control, a slow reaction time, limited field of view, etc. Another transport-related problem is that of loss of time which may be caused by a slow driving speed due to weather conditions, road conditions, visibility, traffic congestion, etc. In this paper, we present a distributed architecture and the underline distributed algorithm for solving the global road transportation system, which is being developed by several European Universities.

Keywords Distributed models · Holonic systems · Route planning

Introduction

In recent years, many automated traffic systems have been developed to find a solution to problems such as those outlined above. The Global Automated Transport System (GATS) (Zelinkovsky 2008) is a driver-less, integrated transport system. It has the ability to simultaneously coordinate the macro and micro needs of road transport networks. Millions of vehicles can be optimally, simultaneously and automatically "driven" over a virtually unlimited geographic region, including whole continents, while at the same time having the requirements of each individual vehicle and its passengers attended to. It is an innovative concept, based on simple, recognized principles and proven technologies.

The application of this system will revolutionize road travel by dramatically increasing safety, reducing congestion, and eliminating driving-associated stress and fatigue. The consequence will be an overall improvement in the quality of everyday life.

Due to its decentralized, modular architecture it can be implemented with the same ease and simplicity in small contained areas, such as airports and theme parks, as in larger areas such as local, national and international road systems.

GATS aims to provide an automated road-vehicle transport system which do provides an answer to known traffic problems. This driver-less automated transport system has the following features:

- Fully automatic driving of vehicles on all kinds of roads and all kinds of intersections
- Virtually unlimited volume of traffic, potential of system growth and geographical expansion
- Decentralized, hierarchical modular architecture
- Integrated, real-time communications and control among all system components
- Real-time personalized vehicle identification, localization and control
- Real-time, automatic sensing of driving conditions anywhere and of the functioning and safety conditions of each individual vehicle
- Real-time optimal navigation in accordance with the dynamically changing requirements of passengers, vehicles and road conditions

F. Versteegh
University of Twente, Enschede, The Netherlands
e-mail: f.c.versteegh@student.utwente.nl

M. A. Salido · A. Giret (✉)
Universidad Politécnica de Valencia, Valencia, Spain
e-mail: agiret@dsic.upv.es

M. A. Salido
e-mail: msalido@dsic.upv.es

– Management of an entire regional or continental traffic system.

The main contribution of the paper is the definition of a distributed architecture for modelling GATS and the underline distributed algorithm for solving it. In section "Terminology and integrated functioning of GATS" we explain the basic terminology and integrated functioning of GATS. Section "Problem requirements" overviews the problem requirements. A holon-based distributed architecture for GATS is detailed in section "Holonic systems". Section "A distributed architecture based on Holons for GATS" describes the distributed algorithm and its complexity. Section "The underline distributed algorithm" overviews some related works and finally, we present some conclusions and further works in section "Related works".

Terminology and integrated functioning of GATS

In this section we briefly summarize the internal structure of GATS. In the centre of a traffic lane, 10–15 cm under the road surface lies an "intelligent cable". This cable comprises tiny Road-Units (RU's) located at fixed distances (approximately 3 m) from each other. While driving, the vehicle sends short radio transmissions down towards the RU's at regular time intervals. The RU receives a transmission, processes it and responds with a radio transmission back to the vehicle. As the vehicle moves along the road, it communicates with the RU's one after the other instantaneously, so the vehicle has continuous radio communication with the RU's. The RU's are connected to each other by two means: directly, by Serial Buses (tiny lines in Fig. 1), and indirectly, by Parallel Buses (bold lines in Fig. 1).

The memory of each RU stores the specifications of the RU and individual driving instructions that it will transmit to each vehicle above it. Several hundred consecutive RU's constitute a Segment, whose functions are administered by a Segment Controller. The Segment Controller is connected to its RU's through the Parallel Buses and is responsible for "driving" the vehicles passing through its domain, performing routine maintenance checks on the components in its Segment and monitoring and regulating their mutual performance. At level m, a group of adjacent Segment Controllers has a superior Controller: Level m-1 Controller, which coordinates and controls their individual and mutual functions. A group of adjacent Level m-1 Controllers has a superior Controller: Level m-2 Controller. This goes on hierarchically (Fig. 2). The Level 0 Controller coordinates and controls the functions of the whole system. There is virtually no limit to the number of levels or the size of the system's geographic domain.

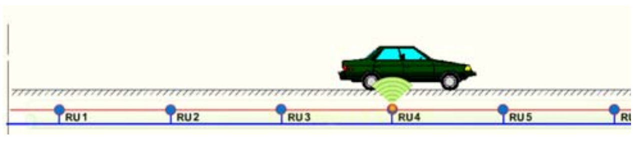


Fig. 1 RU's are connected by serial buses and parallel buses

Let's assume a vehicle is in a parking lot above a RU. The passengers turn the vehicle on, which begins to send short radio transmissions down towards the road. The RU detects those transmissions and responds with radio transmissions back to the vehicle. The RU initiates a communication session with the Segment Controller, in order to inform it about the new event. The passengers in the vehicle enter their requirements such as destination, priority, preferred routes etc. The vehicle's processor sends a message to the Segment Controller which includes the requirements, the exact location of the vehicle relative to the RU and its own specifications. The Segment Controller processes the request while considering additional inputs from other RU's in the Segment and from its superior Controller. Finally it prepares a driving instruction message for each RU in the Segment. The RU's will send these instructions to the vehicle when it passes above them. Each message includes an addressee (RU1 etc.), a vehicle ID, the expected arrival time of the vehicle at the RU, the speed that the vehicle should travel at and the driving direction. When the vehicle is driving from one RU to another, the active RU uses the Serial Bus to inform the next and the previous RU's in the sequence of the exact timing, the ID number and other specifications of the moving vehicle. If the RU's detect intolerable deviation from the plan they can initiate a so-called Emergency Braking Procedure. The active RU uses the Parallel Bus to inform the Segment Controller of the same information as the moving vehicle.

Problem requirements

In this section we present the problem requirements for the architecture we will develop later.

The GATS controller hierarchy (Fig. 2) can be viewed as a network of processors whose computational tasks are accomplished in a distributed manner. The cars are "objects" that have to be transferred from a source to a destination. In graph theory terms, the planar graph on which the cars travel is partitioned into a set of sub-graphs, each managed by a subset of the controllers. The partition is refined recursively as we go down in the controllers' hierarchy tree.

Our aim is to develop a general algorithm, which will supervise the vehicles in any segment and at any level. Thus, the algorithm is scalable and it can be applied to any level. The origin and destination of a vehicle determine which controllers are required to calculate the shortest path. The task of every controller at any level is the same, namely, calculating the journey between two controllers or segments at the level

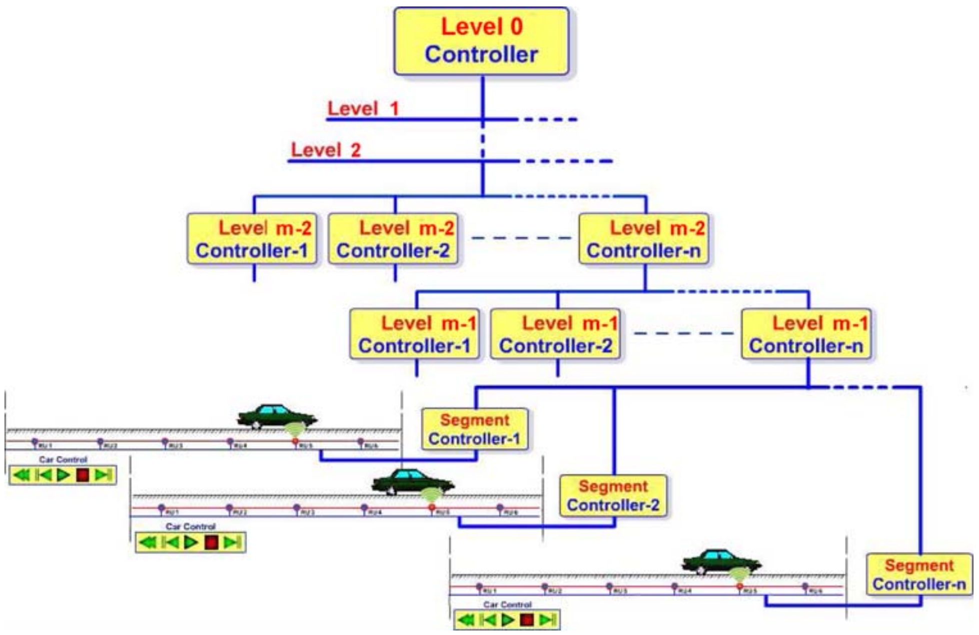


Fig. 2 Hierarchical structure of the system

below, until the lowest level of segments is reached. As the level of controllers is virtually unlimited and to reduce costs, the algorithm should be the same for every controller at every level. This requirement calls for the use of a distributed and scalable algorithm.

To be able to approach the journey of each vehicle in real-time as closely as possible, several other factors, along with distance and maximum speed, need to be taken into account. The choice of path and driving speed for a vehicle should be adapted to its individual characteristics, including vehicle and driver related factors such as driver experience and vehicle condition. As well as these static factors, the algorithm should be able to take into account dynamic factors such as weather and traffic density. Upon the entry of a vehicle in a segment, the Segment Controller will have to determine the vehicle's probable point and time of exit. It will then have to set a plan accordingly. This plan is susceptible to changes underway since it will be influenced by other vehicles using the same segment. This means that the route has to be recalculated along the way to be sure that the best schedule is chosen.

The hierarchical structure, as explained in section "Terminology and integrated functioning of GATS", implies that every controller only has information about the segments it controls on one level below. It does not have any information about what happens in other segments or in other controllers. A level of privacy between controllers and segments has to be maintained by the algorithm.

Holonic systems

In this section we summarize the fundamental features of holonic systems in order to detail the advantages of using

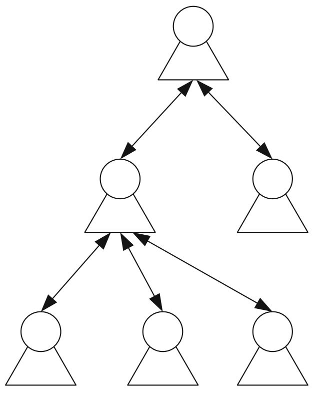


Fig. 3 A hierarchical structure

such a distributed architecture for solving complex and large-scale problems.

A hierarchical structure is a horizontal structure in which there is a central head or controller (Fig. 3). The only type of communication allowed is between the controller and its controlled entities. This yields a deterministic system in which control is achieved by means of the use of global information. Nevertheless, disturbances and changes cannot be coped with.

A heterarchical structure is a flat structure composed of independent entities (agents) where there is no central authority (Fig. 4). The allocation of tasks to these agents is done using dynamic market mechanisms. This yields a simple and fault-tolerant system since none of the agents need a-priori information about the other agents. As a consequence, several disturbances and changes can easily be coped with. Nevertheless, the basic assumption of this architecture paradigm gives rise to its principle drawbacks: the independence

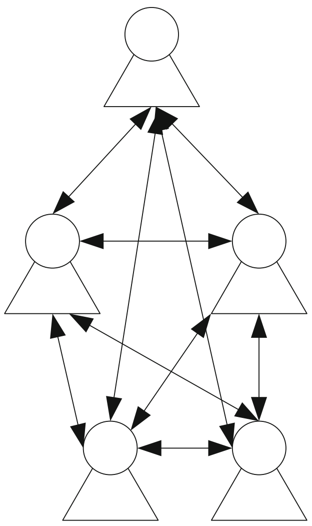


Fig. 4 A heterarchical structure

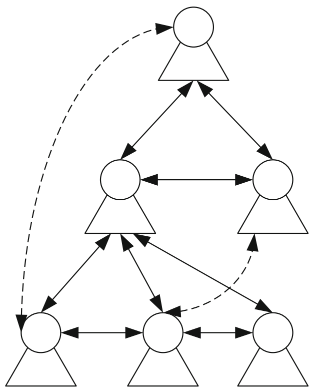


Fig. 5 A holarchy structure

of agents prohibits the use of global information. Therefore, global system performance is very sensitive to the definition of the market rules; the system cannot guarantee a minimum performance level in case of unforseen circumstances.

A holarchy structure is a combination of the two previous ones. Within it, there are temporal and relaxed hierarchies that are dynamically created and deleted as and when required (Fig. 5). Moreover, the participating entities, called holons, are autonomous and cooperative entities that can be seen as a whole and a part. Therefore, a holarchy is a group of basic holons and/or recursive holons that are themselves holarchies. The holonic concept was developed by the philosopher Arthur Koestler (Koestler 1971) in order to explain the evolution of biological and social systems. The strength of a holonic organization, or a *holarchy*, is that it enables the construction of very complex systems which are efficient in the use of resources, highly resilient to disturbance (both internal and external), and adaptable to changes in the environment in which they exist. Within a holarchy, holons may dynamically create and change hierarchies. Moreover, holons can participate in multiple hierarchies at the same time. Holarchies are recursive in the sense that a holon may itself be an entire holarchy that acts as an autonomous and cooperative unit in the first holarchy.

The stability of holons and holarchies stems from holons being self-reliant units, which have a degree of independence and can handle circumstances and problems on their particular level of existence without asking higher level holons for assistance. Holons can also receive instruction from, and to a certain extent be controlled by, higher level holons. This self-reliant characteristic ensures that holons are stable and able to survive disturbance. The subordination to higher level holons ensures the effective operation of the larger whole.

Perhaps there is some misunderstanding about the relationship between a holonic system and a MAS (Multi-Agent System) (Wooldridge and Jennings 1995). The main difference derives from the recursive definition of holon. A holarchy is composed of a set of holons, but a holon can also be divided into a set of holons again. However, a MAS is composed of a set of agents, but an agent is an atomic entity and therefore indivisible. Thus, we consider a holonic system that involves a MAS. The scalability of holonic systems makes them very useful for managing large-scale problems. Given a network, it can be part of a larger one, or it can be divided into smaller ones.

In summary, a holonic architecture is committed to organizing entities—holon or agent (Giret and Botti 2004a)—which are responsible for solving each subproblem. In order to unify the concepts of holon and agent, we will use the Abstract Agent notion (for short, A-Agent) (Giret and Botti 2004b) to refer to any of these entities indistinctly.

A holonic architecture seems appropriate for the problem we are trying to solve, due to the modular nature of GATS and the need for deterministic control over it. Holarchies are temporal "relaxed" hierarchies, in which the hierarchical structures are created and destroyed on-demand, based on the current system requirements. Communication among the entities of the same levels is not forbidden. This structure will allow autonomous and cooperative behavior among the segment controllers; the distributed solving of the road assigning per segment (divide and conquer); the on-demand composition of complex structures (recursive decomposition into simpler or complex segments); and deterministic distributed management of the whole transport system.

A distributed architecture based on Holons for GATS

Traditional centralized techniques fail to model and implement problems of this type due to their large and complex nature. Due to the decentralized and modular nature of the architecture, the algorithms for calculating the scheduling of each vehicle must be distributed.

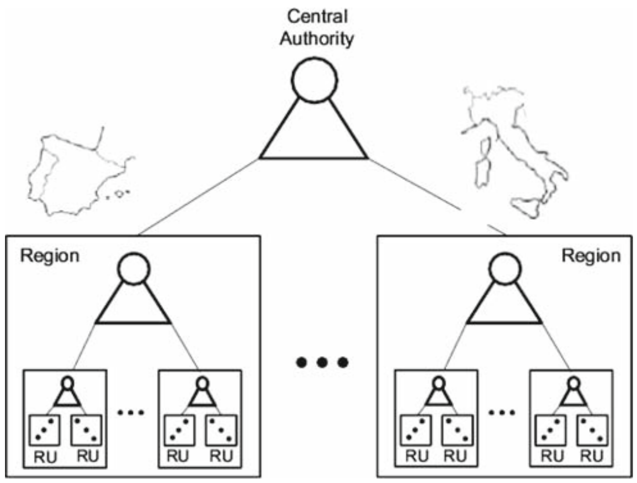


Fig. 6 Distributed model with a central authority

Briefly, the system is composed of a network, where nodes are locations and arcs are roads. Depending on the granularity, nodes are points in the road or regions in a country. At the lower level of the system, each RU is represented by a node and arcs are roads connecting RUs (see Fig. 6). The system may be composed of millions of RUs. In the specialized literature, there are many works about distributed CSPs (DisCSPs). In (Yokoo and Hirayama 2000), Yokoo et al. present a formalization and algorithms for solving distributed CSPs. These algorithms can be classified as synchronous backtracking or asynchronous backtracking. The work of Yokoto on DisSCP, and many works by other researchers, are based on the same principle: the problem is modelled as a Multi-agent system (MAS) using the mapping variable → agent (Wooldridge and Jennings 1995). However, to solve our problem efficiently, we cannot maintain a variable per agent (Salido 2007), due to the fact that it can be composed of thousand of variables and constraints. By using these approaches, the problem exploits into thousands of agents, an exploitation of the number of messages passing in the interaction scenarios. This makes the resulting DSCP model and implementation almost unmanageable.

To overcome this weakness, we use the distributed model presented in (Salido et al. 2007) in which the problem is partitioned into subproblems which represent regions, countries, etc (see Fig. 6). We use a Holonic architecture (HMS 1994; Koestler 1971) to organize the entities (holon or agent (Giret and Botti 2004a) responsible for solving each subproblem. We can observe that a country can be seen as a whole and a part: as a whole if the network is not increased and only optimal paths are calculated into the country, but as a part if it is connected to other countries to calculate international paths. Hence, a country can be a holon (country-holon). A set of country-holons compose a holarchy (continent-holon). Furthermore, a country-holon is composed of a set of holons (region-holon). Each region-holon represents its own network with its own rules, traffic policies, etc. However, they do not represent independent problems, since they must cooperate to achieve a global solution.

Each problem must be solved by a CSP solver in a distributed framework, where cooperation, flexibility, autonomy, reactiveness and pro-activeness is necessary to achieve a global and sub-optimal solution. The problem solver (CSP) is encapsulated into a controller holon, and this holon is responsible for the management of the traffic flow throughout the segment under its control. The main benefit of the distributed management of the problem is that the original and constrained problem can be divided into a set of solvable problems. Each controller can maintain its own privacy for confidential data. Every subproblem can be solved concurrently to achieve a global solution in a reasonable time. In order to do this, every controller holon: (i) has the intelligent processing capabilities necessary for planning and assigning roads to vehicles, in a cooperative way; (ii) participates in negotiation scenarios in order to connect the traffic of its segment to other segments by means of intersections; (iii) participates in cooperation scenarios for ensuring security levels based on the priorities and individual features of the vehicles on the different roads, and; (iv) participates in cooperation scenarios for controlling and resuming from unforseen circumstances.

The distributed model generated for this scheduling problem follows the following guidelines.

– The number of subproblems depends on the size of the system. A holon can represent a track between two traffic lights or a region or country. Figure 6 shows two holons which represent two countries, Spain and Italy. Each of them is composed of a set of sub-holons which represent regions, and each sub-holon is composed of new sub-holons which represent sub-regions and so on. The base case is composed of individual variables that represent RUs. Without loss of generality we will call every problem/sub-problem controller holon or agent indistinctly (Giret and Botti 2004a).
  In Fig. 7 we can see the different types of A-Agents that make up the holonic architecture for GATS. The *Vehicle* A-Agent represents the controlling entity of the vehicle which interacts with the user. This holon stores information (Beliefs) about the *Vehicle and User Parameters, Destination* and *Preferences* for a journey. The *Vehicle* A-Agent has the following goals: *Follow the optimal path* and *Go from origin to destination*. On the other hand, the *RU* A-Agent attempts to *Control running vehicles* above its zone, *Cooperate with adjacent RUs* in order to connect zones, and *Follow SC orders* to guide the vehicle running. The *RU* A-Agent stores the following beliefs: the list of the *Running Vehicles* above its zone, the *SC* which con-

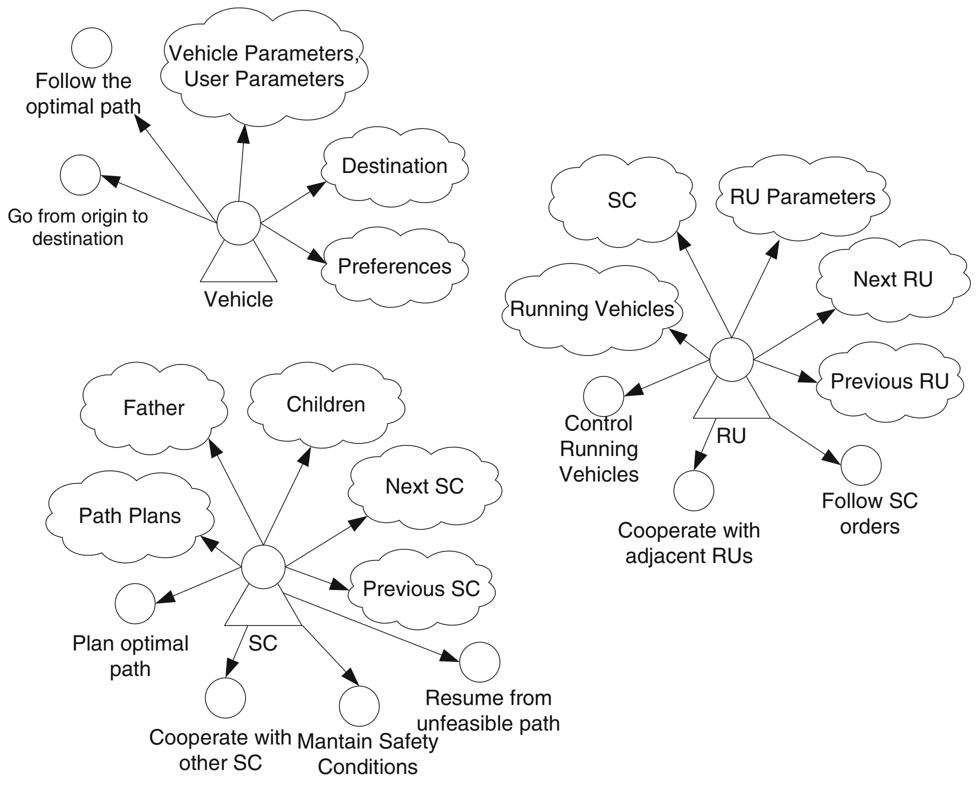


Fig. 7 Holons of the holonic architecture for GATS

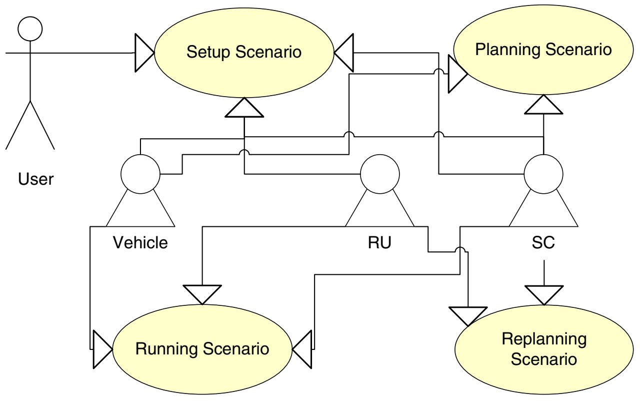


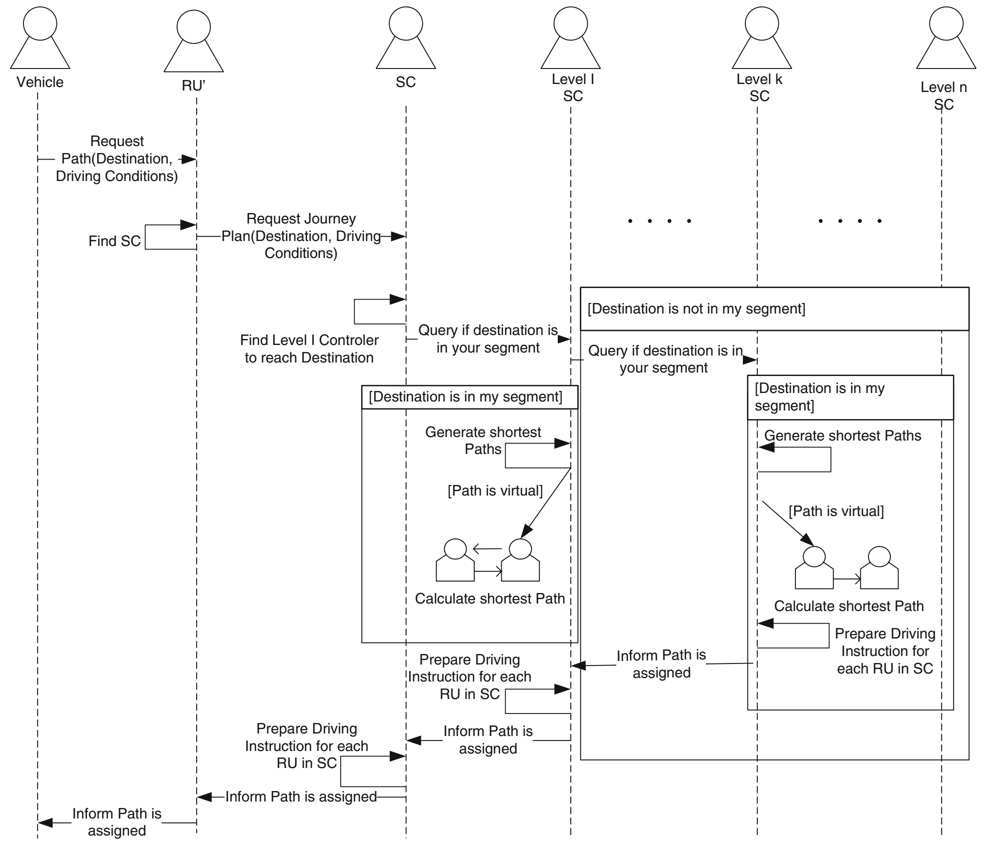
Fig. 8 Cooperating Scenarios for the GATS problem

trols it, *the RU Parameters*, and its adjacent RUs (*Next RU* and *Previous RU*). Finally, the *SC* A-Agent is the holon which implements a Segment Controller. This holon has the following goals: *Plan optimal path, Resume from unfeasible path, Maintain Safety Conditions* during the running of the vehicle, and *Cooperate with other SC* to plan a journey path in a distributed fashion.
These holons participate in four cooperation scenarios in order to support the GATS problem. The scenarios are depicted in Fig. 8. In the *Setup Scenario* the User specifies its journey *Preferences* and *Destination* to the *Vehicle* A-Agent. The *Planning Scenario* is committed to calculating the journey plan in a distributed fashion. The *Running Scenario* is executed when the vehicle is running to its *Destination*. This scenario is committed to control and guide the vehicle throughout its journey path. Finally, the *Replanning Scenario* is executed whenever a journey path becomes unfeasible (due to heavy traffic in a segment, RU or SC failures), and an alternative path has to be calculated.

– The execution of the subproblems is carried out in two steps. Firstly, given the requirements of the passenger, (the destination is the most important requirement), the central authority is the *Level i Segment Controller* which involves both origin and destination. This *Level i Segment Controller* is committed to solving the shortest path in a high level problem (each node is a region). This path is only a first approach that guides us to find the real shortest path. Thus, *Level i Controllers* are executed first, then all *Level i+1 Controllers* are executed concurrently, and so on. Depending on the size of the journey, several hierarchical levels are necessary. Finally, the calculated route is sent to the *Segment Controllers* involved.
Figure 9 depicts a sequence diagram[1] for the *Planning Scenario*. In this figure we can see the participating entities in the cooperation scenario and the message passing among them. Firstly, the *Vehicle* A-Agent requests a journey plan from the *RU'* holon. In order to do so, *RU'* sends

[1] A sequence diagram depicts the interacting entities, the message passing and the time. The time is as a vertical line that represents the lifecycle of the interacting entity. A conditional sequence is represented as a box with a condition label.

Fig. 9 Planning scenario


Vehicle
RU'
SC
Level I SC
Level k SC
Level n SC
Request Path(Destination, Driving Conditions)
Find SC
Request Journey Plan(Destination, Driving Conditions)
Find Level I Controler to reach Destination
Query if destination is in your segment
[Destination is not in my segment]
Query if destination is in your segment
[Destination is in my segment]
Generate shortest Paths
[Path is virtual]
Calculate shortest Path
[Destination is in my segment]
Generate shortest Paths
[Path is virtual]
Calculate shortest Path
Prepare Driving Instruction for each RU in SC
Inform Path is assigned
Prepare Driving Instruction for each RU in SC
Inform Path is assigned
Prepare Driving Instruction for each RU in SC
Inform Path is assigned
Inform Path is assigned

the request to its *SC*. The *SC* holon finds out if the journey *Destination* falls into its controlled segment. If so, *SC* prepares the driving parameters for each RU under its control. If not, *SC* queries its superior *SC*s in order to find a *Level i SC* which controls the segment the *Destination* is in. When such a *Level i SC* is found the *Level i SC* calculates a shortest path to *Destination*. If this shortest path is virtual (that is, it connects segments by meta-RUs) a new *Calculate Shortest Path* (see Fig. 10) scenario is started. This sequence is repeated until a real shortest path is found (that is, every node in the path is a RU).
The *Calculate Shortest Path* cooperation scenario depicted in Fig. 10 represents the interaction scenario among a SC called *Father SC* and its sub-SCs (SCs of lower level that are controlled by *Father SC*). Every sub-SC is called *Child i SC*. This scenario is a recursive one since, whenever the calculated path is virtual, a new *Calculate Shortest Path* scenario is initiated. This recursive process is executed until all of the calculated paths are real (that is, every node in the path is a RU). When *Father SC* receives all of the shortest sub-paths from its children, it composes the final shortest path. Finally, *Father SC* informs its children of the final shortest path, and asks them to prepare the driving instructions for each RU in their segment.
Figure 11 shows the cooperation scenario executed when the vehicle runs along its journey path. The *RU* in charge of the vehicle running zone is responsible for guiding it. To this end the *RU* receives radio signals with the position data of the vehicle. The *RU* continuously calculates whether the vehicle is going to leave its control. When that happens the *RU* informs its *SC* of the situation in order to connect the vehicle to the next *RU* in the path. The *SC* sends driving parameters to the *Vehicle* via the *RU*. Also, *SC* checks when a new connecting *SC'* has to be contacted in order to pass it the *Vehicle* running control. When the *Vehicle* is no longer in a given zone or segment it is deleted from the list of running vehicles for that segment.

– Due to the dynamic structure of the problem, some parts of the system may change and new schedules must be calculated. The rescheduling is only calculated from the incidence to the destination. At the same time, two regions (say “A” and “E”, see Fig. 12) not connected previously, can be connected if a new road is available at a given time. In this case, controller holon “A” and controller holon “E” will be part of a higher level controller holon (say “C”) to manage the new region which connects “A” and “E”. Also, previously connected regions (“C”) may be disconnected at a given time due to extreme weather conditions, problems with the roads, etc. In these cases, the controller holon which manages the connected regions (“C”) will no longer exist until the regions (“A” and “E”) are connected again.

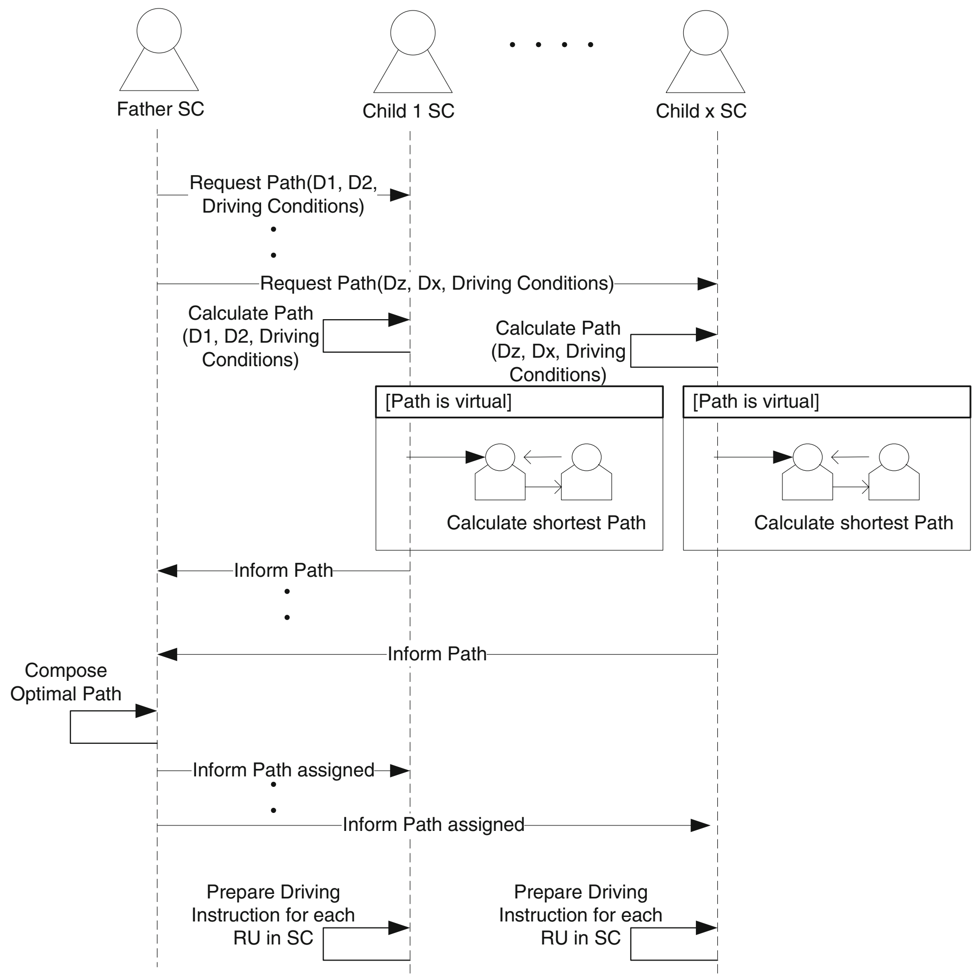

Father SC
Child 1 SC
Child x SC
Request Path(D1, D2, Driving Conditions)
Request Path(Dz, Dx, Driving Conditions)
Calculate Path (D1, D2, Driving Conditions)
Calculate Path (Dz, Dx, Driving Conditions)
[Path is virtual]
Calculate shortest Path
[Path is virtual]
Calculate shortest Path
Inform Path
Inform Path
Compose Optimal Path
Inform Path assigned
Inform Path assigned
Prepare Driving Instruction for each RU in SC
Prepare Driving Instruction for each RU in SC

Fig. 10 Calculate shortest path interaction

– The nature of the system makes a central authority necessary. However, due to the scalability of the system, the central authority behaves similarly to a segment controller. The central authority is the minimal level controller which involves both origin and destination. This level depends on the problem instance.

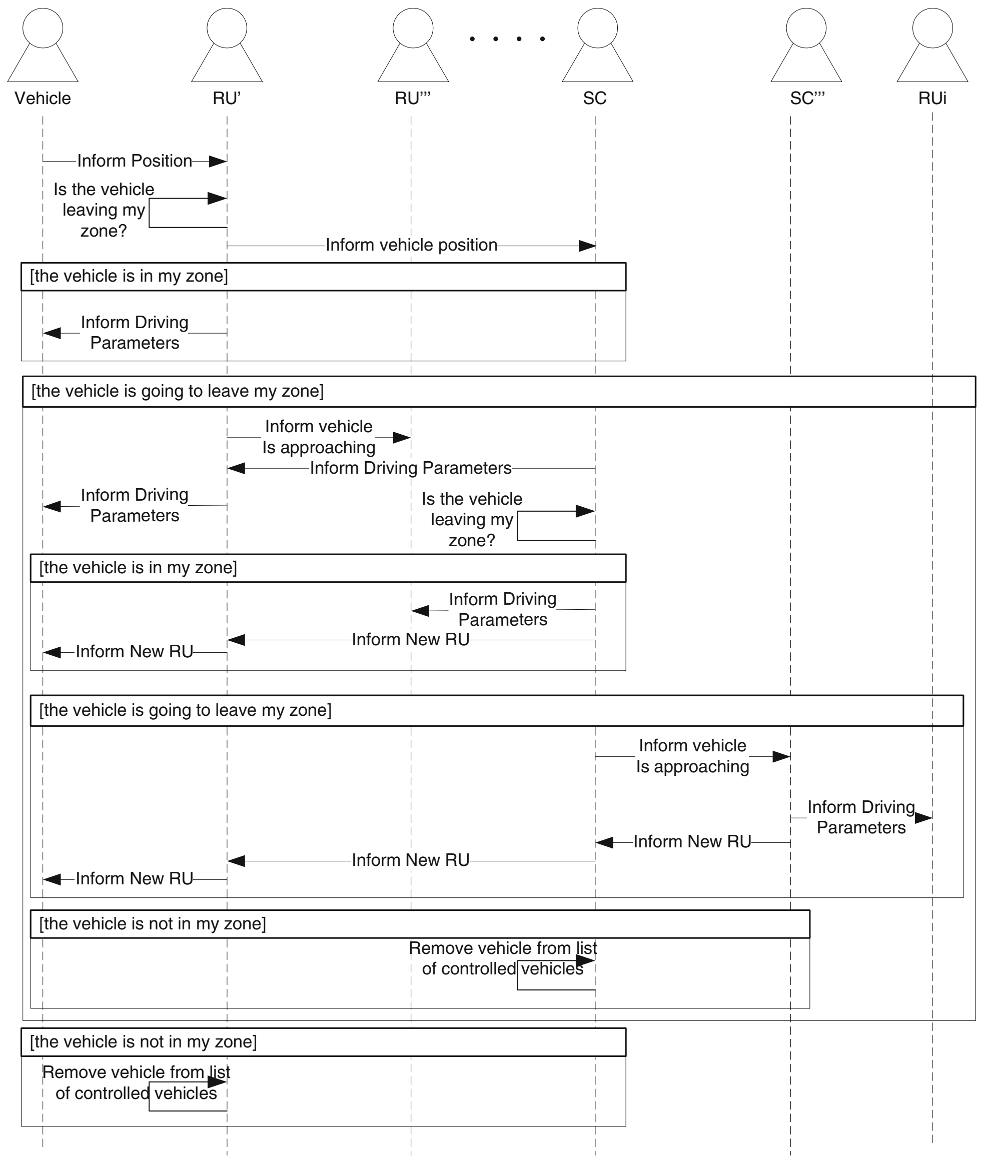
The underline distributed algorithm

An underline algorithm is needed to establish the controlling holonic structure for each vehicle journey. This algorithm is composed of two steps. In the first step, the algorithm determines a high level path in which a set of regions are selected to build up the holonic structure. In the second step, the algorithm calculates the low level path corresponding to the sup-optimal path. The detailed pseudocode of the described procedures that compose the algorithm can be found in (Versteegh and Salido 2007).

The *complete distributed algorithm* we propose (Fig. 13) uses the procedures described below to calculate the shortest path between the origin node and the destination node the user wants to reach. Once the user inputs the desired destination, the procedure DetermineHighLevelPath is executed. If a path exists from the begin region to the end region, in other words if the predecessor of the end region is unequal to −1 and the number of iterations made in the procedure is smaller than or equal to the number of nodes, the algorithm continues. If not, the message *“there is no possible route”* will be shown. If the algorithm continues the DetermineLowLevelPath procedure is executed, followed by the procedure MakeFinalPath. Finally, the total distance and the path through regions are shown (see Fig. 14).

In the *DetermineHighLevelPath* procedure, the shortest path from start region to end region at level 0 is determined. This means that the path is only made up of regions; no nodes inside regions are taken into consideration. The result of this procedure is a list composed of the regions through which the shortest path will be planned. In this procedure, the file with data of level 0 is loaded first. For every two neighbor regions the file concerning level 0 contains the distance between them. If no direct path exists the distance is set to −1.

The procedure *DetermineLowLevelPath* determines the shortest path for every region selected by DetermineHighLevelPath. Holon 0 gives all involved holons at level 1 the order to calculate the shortest path from all frontiers with the


Vehicle
RU'
RU'''
SC
SC'''
RUi
Inform Position
Is the vehicle leaving my zone?
Inform vehicle position
[the vehicle is in my zone]
Inform Driving Parameters
[the vehicle is going to leave my zone]
Inform vehicle Is approaching
Inform Driving Parameters
Inform Driving Parameters
Is the vehicle leaving my zone?
[the vehicle is in my zone]
Inform Driving Parameters
Inform New RU
Inform New RU
[the vehicle is going to leave my zone]
Inform vehicle Is approaching
Inform Driving Parameters
Inform New RU
Inform New RU
Inform New RU
[the vehicle is not in my zone]
Remove vehicle from list of controlled vehicles
[the vehicle is not in my zone]
Remove vehicle from list of controlled vehicles
Fig. 11 Running scenario

precedent province to all frontiers with the next province. For the beginning region the shortest path from the begin node to all frontier nodes with the next province is calculated. For the end region the shortest path from all frontier nodes with the precedent region to the end node is calculated. The sequence of provinces is given by the shortest path calculated by holon 0. This procedure is only executed if the predecessor of the end province is unequal to -1, which means that a path to this province is found. For every two adjacent provinces frontier nodes are determined. In general we describe a frontier node as an artificial point that is shared by two regions.

The MakeFinalPath procedure has two different parts. In the first part the shortest path from begin node to end node via frontier nodes is determined, using the paths from frontier node to frontier node that are calculated in the Determine-LowLevelPath procedure. In the second part of the Make-FinalPath procedure the complete path through every region is determined. We calculate this path using a modified form of *Dijkstra's shortest path algorithm*. In this procedure not only the visited status, distance from start and predecessor of a node are saved, but all of these data are saved for region-node combinations.

Complexity

The worst case complexity of the algorithm is $O(N^2R^2)$, in which R is the number of regions in the system and N is the number of nodes per region. The complexity depends on the number of regions in the final path, p, and the maximum number of frontier nodes per frontier, f. The MakeFinalPath procedure accounts for the worst part of the complexity. The

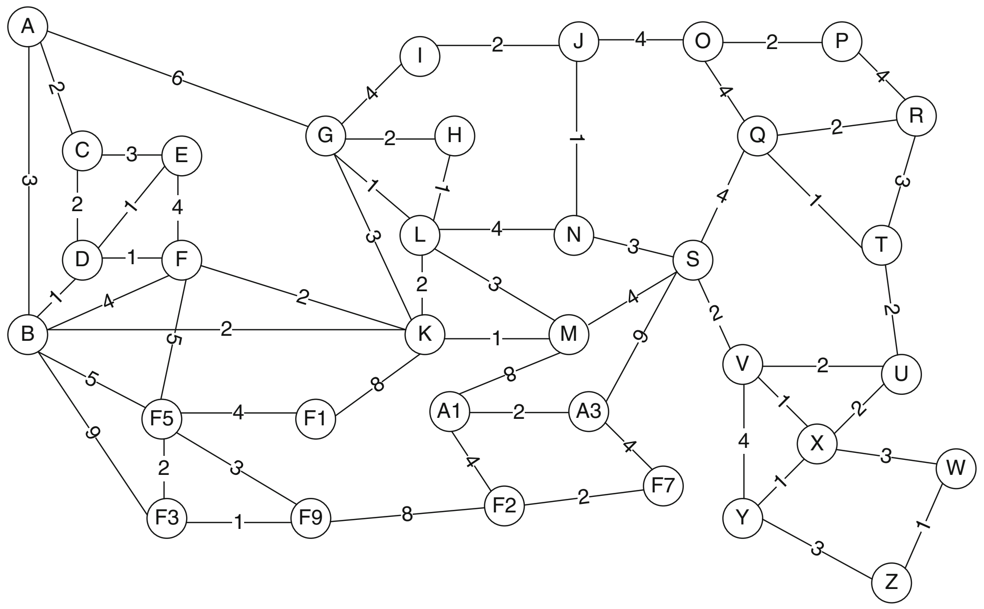


Fig. 12 Initial weighted graph that represents locations (nodes) and best time to go from one location to another one (arcs)

```
Distributed Algorithm

begin
    Ask the user for the destination
    DetermineHighLevelPath
    node=predecessor(end region)
    if number of iterations<=number of nodes and node<>-1 then begin
        DetermineLowLevelPath
        MakeFinalPath
        Show total distance in interface
    end
    else
        show message: "there is no possible route"
    end
end
```

Fig. 13 The complete distributed algorithm

complexity of this procedure is $O(pN^2R)$. The procedure DetermineHighLevelPath depends on both f and p. Its complexity is $O(fpN^2)$.

Related works

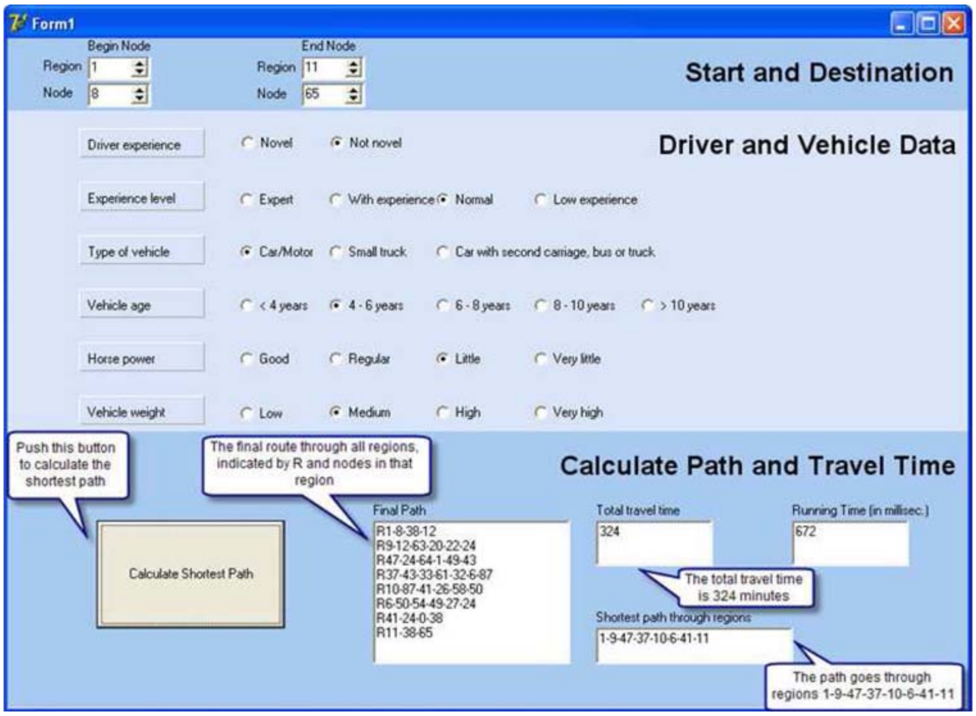
Often in distributed shortest path algorithms the individual nodes need to be able to locally compute shortest paths. In a lot of cases this is done by a variation of *Dijkstra's Algorithm*, the most well-known and fastest basic algorithm for determining the shortest path from one location to all other possible locations in a network (Dijkstra 1959). The use of *Dijkstra's Algorithm* alone is not sufficient for GATS as we need a distributed algorithm and Dijkstra only provides a local shortest path algorithm. In this section we discuss some of the existing distributed shortest path algorithms and their possible usefulness for GATS.

The *Link-State Algorithm*, the *Distance-Vector Algorithm* and other derived algorithms are distributed shortest path algorithms in which each node is responsible for calculating a shortest path to all other possible destinations (Hoffman 2004). Although these algorithms are called distributed algorithms, they are distributed in a different way to those used in this paper. No agents are used, therefore the privacy of the controllers cannot be guaranteed in these algorithms.

The well-known commercial navigation systems use, according to Sanders and Schultes (Sanders and Schultes 2005), heuristics to compute paths. The basic idea behind these heuristics is the observation that shortest paths in general use small roads only locally, more specifically, at the beginning and at the end of a path. This heuristic algorithm only performs a local search around start and destination and then switches to search in a network that is much smaller than the complete graph and which only consists of the most important roads. This heuristic algorithm does not take into account real-time and vehicle and driver specific data and so is not useful for GATS.

The concept of *highway hierarchies* (Sanders and Schultes 2005) is a multi-level highway network in which iteratively every path consisting of nodes with degree two are replaced by a single edge. Although this approach has a lot in common with the system needed for GATS, it is not useful as the privacy of regions is not guaranteed, nor is it possible to involve real-time data.

None of the shortest path algorithms described above meet all of the requirements for GATS as found earlier in this section. This motivates us to propose a distributed shortest path algorithm in which privacy is maintained, dynamic fac-


Form1
Start and Destination
Driver and Vehicle Data
Calculate Path and Travel Time
Calculate Shortest Path
Push this button to calculate the shortest path
The final route through all regions, indicated by R and nodes in that region
The total travel time is 324 minutes
The path goes through regions 1-9-47-37-10-6-41-11

Begin Node: Region 1, Node 8
End Node: Region 11, Node 65

Driver experience: Novel [✓ Not novel]
Experience level: Expert, With experience, [✓ Normal], Low experience
Type of vehicle: [✓ Car/Motor], Small truck, Car with second carriage, bus or truck
Vehicle age: < 4 years, [✓ 4 - 6 years], 6 - 8 years, 8 - 10 years, > 10 years
Horse power: Good, Regular, [✓ Little], Very little
Vehicle weight: Low, [✓ Medium], High, Very high

Final Path:
R1-8-38-12
R9-12-63-20-22-24
R47-24-64-1-49-43
R37-43-33-61-32-6-87
R10-87-41-26-58-50
R6-50-54-49-27-24
R41-24-0-38
R11-38-65

Total travel time: 324
Running Time (in millisec.): 672
Shortest path through regions: 1-9-47-37-10-6-41-11

Fig. 14 Result of calculating the shortest path

tors are taken into account and in which the algorithm for each controller is the same. In our approach we combine multi-agent theory (Wooldridge and Ciancarini n.d) with *Dijkstra's Algorithm* to come to an algorithm that is useful for GATS.

Conclusions and further research

In this work, we present a distributed architecture for the Global Automated Transport System (GATS) (Zelinkovsky 2008). This architecture is modeled as a holonic system. It is appropriate due to the modular nature of GATS and the need for deterministic control over it. The underline distributed algorithm for solving the global road transportation system has to satisfy some requirements as described in Sect. "Problem requirements". We define an algorithm that guarantees privacy for every holon. Every holon, which controls a region or a segment of regions, only has information about what happens in the area it controls. The holon above can ask other holons for information about distances, but will never know directly what happens in other regions. The algorithm we propose is scalable due to the fact that the architecture uses the same algorithm for each controller on every level, every controller determines the shortest path in the region below, once this path is determined the controller in the level below does the same thing, using the information the controller above gathered in its search.

In further work, we will study the dynamic part of the algorithm. What happens when the travel time on a track becomes longer? Will the whole route be recalculated or only the region in which this track lies? Another possibility might be to calculate in the first step not only the shortest route at that moment, but also the best alternative. The difference between the best case of the alternative and the current route could be used as buffer time to determine whether a delay in the current route is large enough to start recalculating the shortest path.

Acknowledgements We appreciate the assistance of Reuven Zelinkovsky for his kind help in obtaining all the information regarding GATS. GATS core concept was conceived, invented and patented by Reuven Zelinkovsky. This work has been partially supported by the research projects TIN2007-29666-E and TIN2007-67943-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER), CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, FOM- 70022/T05 (Min. de Fomento, Spain), GV/2007/274 (Generalidad Valenciana), PAID-06-07/3191 (Universidad Politecnica de Valencia) and by the Future and Emerging Technologies Unit of EC (IST priority—6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

References

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271.
Giret, A., & Botti, V. (2004a). Holons and agents. *Journal of Intelligent Manufacturing, 15*, 645–659.
Giret, A., & Botti, V. (2004b). Towards an abstract recursive agent. *Integrated Computer-Aided Engineering, 11*, (2), 165–177.
HMS. (1994). Press release. 1994. *HMS requirements*. http://hms.ifw.uni-hannover.de/:HMSServer.
Hoffman, A. C. (2004). *Multiple approaches for distributed routing algorithms* (Tech. rept).
Koestler, A. (1971). *The ghost in the machine*. Arkana Books.

Salido, M. A. (2007). Distributed CSPs: Why it is assumed a Variable per Agent? In *Proceeding of the Seventh Symposium on Abstraction, Reformulation and Abstraction (SARA'07), LNAI* (Vol. 4612, pp. 407–408).

Salido, M. A., Abril, M., Barber, F., Ingolotti, L., Tormos, P., & Lova, A. (2007). Domain-dependent distributed models for railway scheduling. *Knowledge Based Systems. Ed. Elsevier Science, 20*, 186–194.

Sanders, P., & Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. In *Proceeding of ESA 2005, LNCS* (Vol. 3669, pp. 568–579).

Versteegh, F. C., & Salido, M. A. (2007). *A Distributed shortest path algorithm for global automated transport system* (Tech. rept). Technical Report DSIC-II/17/07, Department of Information Systems and Computation, Technical University of Valencia, Spain.

Wooldridge, M., & Ciancarini, P. *Agent-Oriented Software Engineering: The state of the art* (Tech. rept). Department of Computer Science, University of Liverpool.

Wooldridge, M., & Jennings, R. (1995). Agent theories, arquitectures, and lenguajes: A survey. *Intelligent Agents, LNCS* (Vol. 890, pp. 1–22).

Yokoo, M., & Hirayama, K. (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems, 3*, 185–207.

Zelinkovsky, R. (2008). Global automated transport system. http://www.global-transportation.com.