

MADARP: A Flexible Agent Architecture for Passenger Transportation

Claudio Cubillos
Pontificia Universidad Católica
de Valparaíso
Av. Brasil 2241
Valparaíso, Chile
claudio.cubillos@ucv.cl

Franco Guidi-Polanco
Pontificia Universidad Católica
de Valparaíso
Av. Brasil 2241
Valparaíso, Chile
franco.guidi@ucv.cl

Claudio Demartini
Politecnico di Torino
Cso. Duca Degli Abruzzi 24
10129, Torino, Italia
demartini@polito.it

ABSTRACT

This work describes the development of a distributed agent-based application devoted to the flexible transportation of passengers. The system considers the planning and control of trip requests coming from clients. The underlying 3-tier agent architecture, named MADARP, provides a set of base agents organized around an interface, a planning and a service layer. Interface agents provide GUIs for the interaction with customers and drivers, while planning agents make use of a distributed version of an improved insertion heuristic called ADARTW for the scheduling of passengers' trips. They make use of the contract-net protocol as base coordination mechanism plus a specific transportation domain&communication ontology. The agent application was designed with the agent-oriented software engineering methodology (AOSE) PASSI and implemented over the JADE agent platform. Performance tests were carried out, to evaluate the application's planning capabilities, by analyzing diverse clients' arrival rates while varying the number of hosts.

1. INTRODUCTION

The need to cover more diffuse travel patterns, varying periods of low demand, city-peripheral journeys, as well as commuting trips often make conventional public transport systems unable to guarantee the level of service required to address the user needs. The use of Demand-Responsive Transport services (DRTS), where routes, departure times, vehicles and even operators, can be matched to the identified demand allows a more user-oriented and cost effective approach to service provision.

For this approach to success it is vital an appropriate Information Technology (IT) application that adequately supports the communication and interaction among the diverse involved actors and systems. Under such scenario, software agents leverage as an interesting paradigm to design and implement this kind of software application. Software agents

are defined as autonomous entities capable of flexible behavior denoted by reactivity, pro-activeness and social ability [18]. Multiagent systems (MAS) consist of diverse agents that communicate and coordinate generating synergy to pursue a common goal. This higher level of abstraction has allowed agents to tackle the increasing complexity of nowadays open software systems.

The present work describes the development of an agent-based application for the planning and control of a passenger transportation system under a flexible approach. It gives continuity to our past research [5] on heuristics for solving the scheduling of passenger trips.

In particular the paper covers the description of the overall MAS architecture for then detailing the agents and interfaces involved from the Customer, Vehicle and Transport Enterprise sides. The used underlying optimization problem and scheduling heuristic are depicted together with some performance tests on the planning capabilities of the system.

2. RELATED WORK

Regarding the state-of-the-art research in the field of transportation scheduling, cluster-first and route-second planning techniques have been widely covered. Borndörfer et al. [2] presented such a two-phase approach applied to several instances provided by an operator in Berlin. A constructive method was presented by Madsen et al. [12] in a package named REBUS, to be used by Copenhagen Fire Fighter Services for the transportation of the elderly and handicapped.

Local search techniques have also been applied by Healy and Moll [8] presenting a local search variant based on a strategy called sacrificing, which consists of biasing the search in the direction of solutions with larger neighborhoods of feasible solutions.

In [17] Toth and Vigo have worked on a tabu threshold post-optimization procedure to improve their parallel insertion procedure. A reactive tabu search heuristic for Pickup and Delivery Problem with Time-Windows (PDPTW) was developed by Nanry and Barnes [13] where solutions that violate time-window and vehicle capacity constraints are allowed during the search. More recently, Cordeau and Laporte [4] have developed a tabu search heuristic for the multi-vehicle Dial-a-Ride Problem (m-DARP).

On the other hand, Agent research in the transportation domain has deserved an increasing interest. A Bus-holding control system was proposed by Jiamin et al. [10], which tackles the coordination of multiple lines of fixed-route buses and the different stops, seeking the global optimality. In their approach a MAS negotiation between a Bus Agent and a Stop Agent was conducted based on marginal cost calculations.

In Urban Traffic Control (UTC) systems, Ou [14] presented a UTC which adopted MAS technology based on recursive modeling method (RMM) and Bayesian learning. Ferreira et al. [7] presented a multi-agent decentralized strategy where each agent was in charge of managing the signals of an intersection and optimized an index based on its local state and "opinions" coming from adjacent agents. Agent-based systems devoted to Vehicle Routing (VRP) are presented in [11] and [15]. Both make use of the Contract-Net Protocol (CNP) for the assignment of rides. In addition, [11] uses a stochastic post-optimization phase to improve the result initially obtained. In [15] is presented the Provisional Agreement Protocol (PAP), based on a Extended CNP and de-commitment techniques.

Finally, none of the above solutions tackles the passenger transportation problem under a flexible approach and at the best of our knowledge no similar systems have been found in literature.

3. FLEXIBLE PUBLIC TRANSPORT SERVICES

Flexible or Demand Responsive Transport (DRT) services can be seen as an element of a larger intermodal service chain, providing local mobility and complementary to other conventional forms of transportation (e.g. regular buses and trams, regional trains). In this context, DRT provides a range of Intermediate Transport solutions, filling the gap between traditional public bus services and individual taxis.

The DRT service can be offered through a range of vehicles including regular service bus, mini-bus, maxi-vans, buses&vans adapted for special needs and regular cars. The use of each vehicle type depends on the transport service to offer, the covered area and the target users. The aim is to meet the needs of different users for additional transport supply. The use of flexible transport services, where routes, departure times, vehicles and even operators, can be matched to the identified demand allows a more user-oriented and cost effective approach to service provision. The adaptation of transport services to match actual demand enables cost savings to the operators, society and passengers.

With respect to process implementation and management, the flexibility of the system is expressed along two main directions: on one hand, users of DRT systems must be provided with user-friendly instruments for accessing the services (such as information, reservation, query update) in several different flexible ways (the so called "anywhere and anytime" access). On the other hand, the organization providing flexible services must be itself flexible, with the capability of managing dynamic relationships in a pool of transport resources (vehicles), which may sometimes have

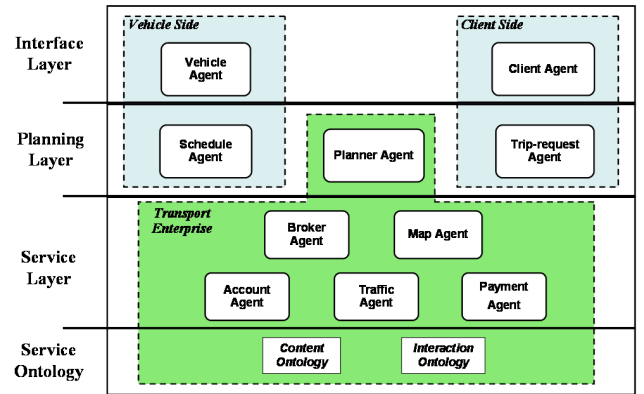


Figure 1: The multiagent transportation architecture

to change to better adapt the transport supply to the dynamic demand.

4. THE AGENT-BASED TRANSPORTATION SYSTEM

In Figure 1 is summarized the agent architecture. A first view shows how the underlying agents are grouped around functional layers to provide a coherent service. While another view shows the architecture from the perspective of the different actors involved, identifying the agents pertaining to each one.

Turning back to the first view, the diagram shows four layers that group the agents and structures according to the functionality provided. The Interface layer connects the system with the real world, providing agents capable of connecting the different actors (clients, vehicle operators) with the system. The Planning layer contains the agents devoted to perform the trips processing and planning in a distributed way. The Service layer supports the above layer providing different complementary functionalities needed for managing a complete transportation service. Finally at the bottom, the Service Ontology provides a means to integrate and make interacting the different agents and actors from the upper layers in a transparent and coherent way.

In the architecture, the control is distributed across the different layers. In general terms, the interface agents provide the input and monitoring signal, for the planning agents to adjust the vehicle's planification. The service agents support these procedures providing with the required information for the re-planning process and the ontology offers the concepts and formalizations for carrying out the control interactions.

The second view provided by Figure 1 shows the three main actors involved in the transportation chain. They correspond to the vehicles, the clients and the transportation enterprise. Each of them is modeled in terms of agents. Consequently, each vehicle actor is represented by a Vehicle agent and a Schedule agent. In a similar way, each client is characterized by a Client agent and a Trip-request agent. In both cases, the pair of agents is tightly coupled as they are modeling different aspects of the same real entity. The

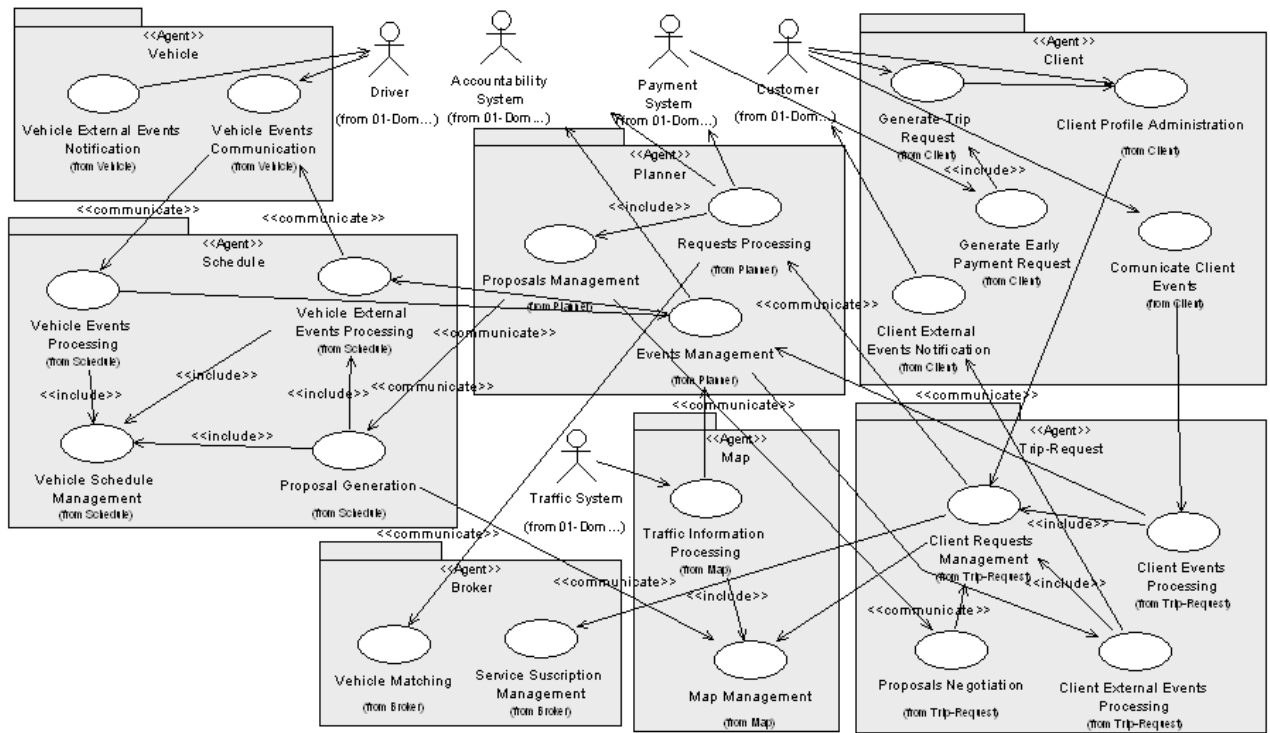


Figure 2: Portion of the Agents' Identification Diagram

third actor is the transport enterprise, which is built up by a series of agents and structures that provide support to diverse services related with the planification and control of the passenger transportation service.

The system was designed following the Process for Agent Societies Specification and Implementation (PASSI) which is made up of five models containing twelve steps in the process of building a multiagent system. Please refer to [3] for a more detailed description on the whole PASSI methodology. Models were developed with the PTK (Passi Toolkit add-on for Rational Rose) and was implemented over the Jade Agent Platform[1], which provides a full environment for agents to work.

The PASSI methodology starts capturing the system's requirements through use cases, for then grouping them together to conform the agents. The diagram in Figure 2 shows the part of the use cases and agents involved in the system. Due to space restrictions some service layer agents were expressed as actors.

By starting from the transport operator side, we find the Vehicle, which is an interface agent (with a GUI) in charge of providing the monitoring of the route-schedule planned for the vehicle. In addition, it can inform the Driver about any changes to the initial schedule and can be used by him to inform any eventuality (e.g client no show, delay, detour, etc) that may happen regarding the trip and the customers. In particular its interface has been designed to work on-board the vehicle through a touch screen. This agent will be further detailed on a next section.

The Schedule agent is the one in charge of managing the trip plan (work-schedule) of the vehicle. In addition, the agent is also responsible of making trip proposals upon Planner request and in case of winning will have to update its actual plan to include the new trip. Upon changes (due to vehicle or client events) informed either by the Vehicle or the Planner agent, the Schedule agent will update the plan and reschedule the remaining requests. The Schedule agent encapsulates the underlying optimization algorithm for scheduling the trips of the vehicle. In our system Schedule agents implement a distributed version of a well known greedy insertion algorithm called ADARTW (Please refer to [6] for further details).

The Client is the second interface agent with a GUI, providing the connection between the end-user (Customer) and the transportation system. Through it, the Customer can request a trip by giving a description of the desired transportation service by means of a *Trip Request Profile*.

Other relevant agent is the Trip-Request, which acts as a proxy representing the Customer in the process of contracting a transportation service. In fact, the trip-request agent is involved in all the interaction of the Customer (through the interface agent) with the transportation system. Its activities regard the management of the client transportation requests, including any negotiation or selection of proposals coming from the Planner, together with processing any events generated by the Customer or by the system. As residing on a device with more processing power (such as a PC), this agent may have diverse degrees of autonomy for taking decisions on the trip proposal to choose and how to

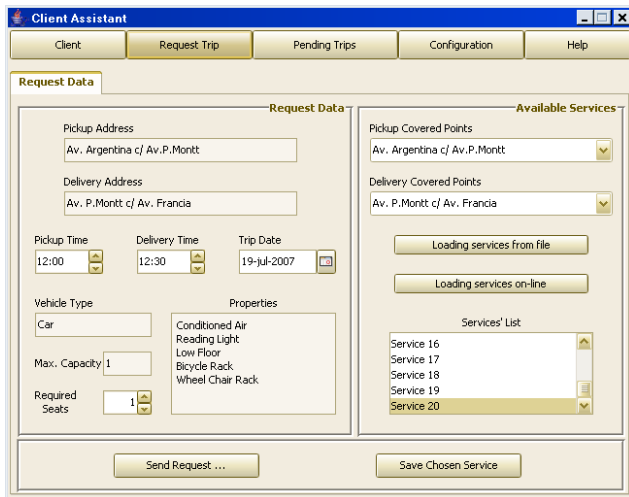


Figure 3: Client agent GUI showing the "Request Data" tab in the "Request Trip" menu

react when faced to eventualities.

Finally, the Planner agent processes all the customers' requests coming through their Trip-request agents. It initiates a contract-net (CNP) [16] with the Schedule agents and manages all the arrived proposals. It is also in charge of managing events that may affect the trip services already contracted and scheduled. The remaining actors correspond to supporting agents or sub-systems from the service-layer that interact with the agent society, such as the broker, responsible for the initial service matching, and the map, providing times and distances, among others.

4.1 The Customer Side

As stated before, the Client is an interface agent devoted to the Customer-System interaction. In principle, this trip-client assistant may reside on diverse devices (e.g PC, PDA, mobile phone) in order to allow a more flexible and pervasive access to the transportation system. In our prototype, has been developed a Client agent for PC, remaining the versions for more restricted devices as future work. In this sense, it is important to highlight that all the complex processing or decision-making (if delegated by the Customer) has been attached to the Trip-request agent in order to lightweight the Client (the interface agent). In the following Figure 3 a screenshot of the Client agent GUI is shown, detailing the tab that appears when initiating the request of a trip. In the "Request Data" area, on the left, is asked all the information necessary to detail a transport service request under the demand-responsive considered scenario. This regards the date, the pickup and delivery points (addresses), the corresponding times and other specific information such as the required seats and diverse vehicle characteristics.

On the right hand, the available transport services are deployed, showing for each selected service the covered area in terms of street intersections. The services' list can be imported from the system (on line) or from a local file. At the bottom, the Customer can send the trip request and save the services' list.

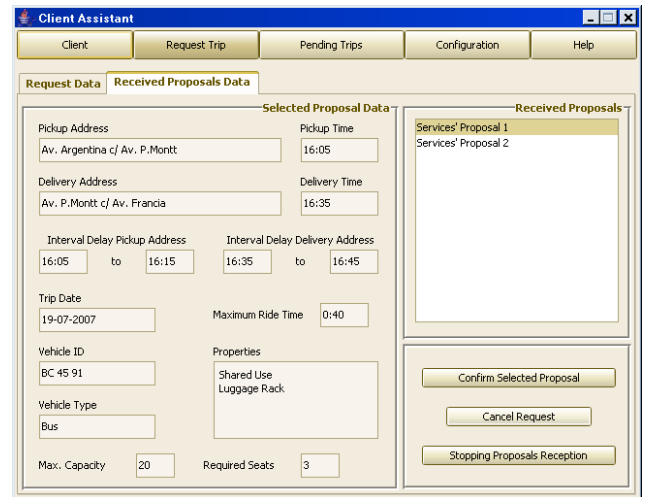


Figure 4: Client agent GUI showing the "Received Proposals Data" tab in the "Request Trip" menu

The following Figure 4 corresponds to a screenshot of the Client agent GUI detailing the "Received Proposals Data" tab which appears as an answer after sending the request for a trip. In this form are displayed all the transportation alternatives found to be capable of performing the service. The list of alternatives is on the right-hand box and by selecting on each of them the left area (Selected Proposal Data) displays the details of such proposal. The data involved concerns the address and requested time for pickup and delivery. In addition, a time window is specified for the pickup and for the delivery in order to make more flexible the service and tackle possible differences with the original schedule.

Other relevant data provided regards the vehicle ID and type, the required seats, together with diverse specific properties, such as the capacity, bicycle rack, shared/individual use, among others. It is important to mention that all the concepts involved in the specification of services make part of a Service Ontology specific for this transportation domain (for further details on the ontology please refer to [6]).

The PASSI methodology used for the modeling considers a *Task Specification* step. In this activity the scope is to focus on each agent's behavior, decomposing it into tasks which usually capture some functionality that conforms a logical unit of work. Therefore for each agent an activity diagram is developed, containing what that agent is capable of along the diverse roles it performs. In general terms, an agent will be requiring one task for handling each incoming and outgoing message.

In the following Figure 5 a portion of the Task Specification Model for the Client Agent is depicted. The diagram shows six tasks that constitute the main Client agent capabilities devoted to the process of requesting a transportation service. The *SendQueryAvailableService* task handles the request from the Customer to search for available services and triggers the *ManageClientQuery* task of the Trip-Request agent which is in charge of requesting the Broker for pos-

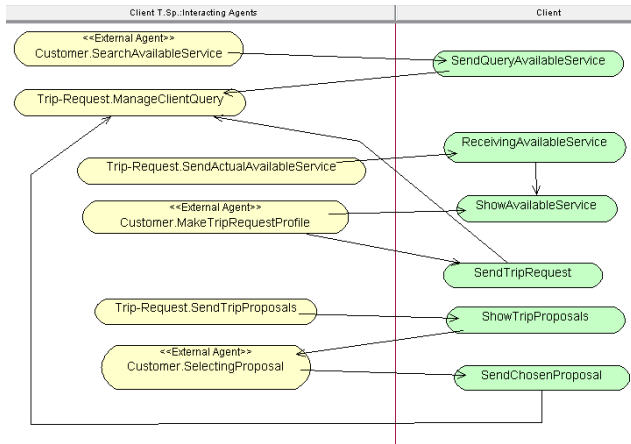


Figure 5: Part of the Task Specification Model for the Client Agent, showing the flow of tasks involved in the trip request processing

sible transportation services available. These are returned by the *SendActualAvailableService* task of the Trip-request and is received by the *ReceivingAvailableService* task of the Client which processes and decodes the ACL message and forwards the services' list to the *ShowAvailableService* task responsible for displaying the list in the proper form as already shown in Figure 4 right-hand box.

The Customer, when making a trip Request Profile (see Figure 3), can browse on the available services (after loading them) in the right-hand area calling to the *ShowAvailableService* task or can send the request (by pressing the button) after filling the left-hand information, calling the *SendTripRequest* task. This Client's task is responsible for sending the *Request Profile* to the Trip-Request, being handled by its *ManageClientQuery* task, which on its turn will forward the request to the Planner.

The Trip-request agent will receive from the Planner the trip proposals coming from the different Schedule agents of the vehicles and its *SendTripProposals* task will send them to the Client. On its turn, the Client will receive and handle the proposals through its *ShowTripProposals* task, also responsible for displaying them on the appropriate form area as already shown on Figure 4.

In this way, the Customer will be able to select the best alternative according to his preferences and will click the "Confirm Selected Proposal" button on the GUI (see Figure 4 first button lower-right corner). This action will call the *SendChosenProposal* task of the Client responsible of forwarding the selected proposal to the Trip-request agent, which on its turn will forward it to the Planner, who is in charge of communicating the proposals' acceptance/rejection to the diverse Schedule agents that made bids.

4.2 The Vehicle Side

As mentioned earlier, the Vehicle agent constitutes an interface agent for the Driver - Transportation System interaction. From the Agents' Identification Diagram of Figure 2, it is possible to see that the Vehicle is responsible for allowing



Figure 6: Vehicle agent GUI showing the main screen with the itinerary

the communication of incoming events to the Customer and of vehicle events to the transportation system.

The following Figure 6 shows a screenshot of the Vehicle agent GUI, detailing the actual vehicle itinerary with expected times. On a first view, we can realize that the layout is minimalistic with simple shapes as buttons. This is because the Vehicle agent is intended to be on-board the transportation vehicle (car, van, maxi-taxi, etc.). Hence, the interfaces were developed to be used in touch screens.

On the left side appears the timetable, providing the expected times (e.g. 10:00, 10:30, 11:00 and so on) of the places to visit (either pickup or delivery points). These can be scrolled up or down with the square buttons in the lower part.

On the right hand the interface is divided in three. A header on the top, showing the present date and time plus a square led that blinks when a change to the itinerary is carried out by the system and needs to be communicated to the driver. On the middle-right are shown the details of the entry selected on the left hand (the 10:00 in this particular case). It provides the destination address to reach, the time limit for departure in that place (10:05) for not arriving late to the next destination (at 10:30 in this case) and the number of passengers that go up and down in this stop. Additionally, it is provided the best route in order to arrive from the actual position of the vehicle to the required destination.

Finally on the footer part, the interface deploys three touch buttons; the first allows to confirm passengers presence at the stop, the second to inform a delay or anticipation with respect to the schedule and the third one to turn back to the main menu.

The Task Specification Model for the Vehicle agent is depicted in Figure 7. This activity diagram contains five tasks that specify the labour carried out by the agent. The task *ObtainEventData* manages the notification of events received from the Driver when touching the *Inform Event* button

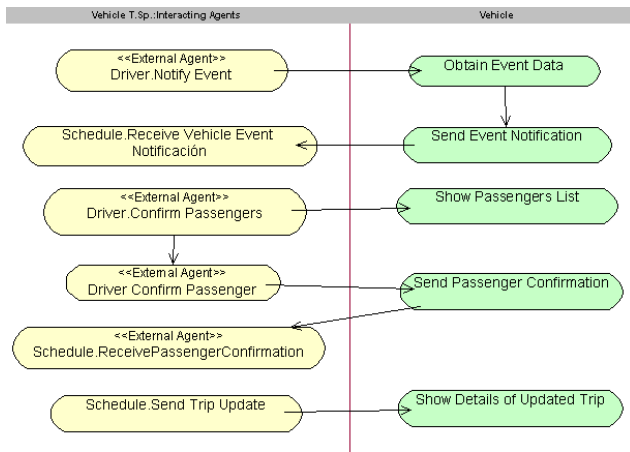


Figure 7: Task Specification model for the Vehicle Agent

of the Vehicle GUI (on Figure 6). The task handles the GUI processing of the notification at low-level, the notification details and forwards them to the *SendEventNotificación* task. This task is responsible for translating the event notification and related data into an ACL message in alignment with the transportation domain ontology and finishes by sending to the Schedule agent the ACL message with the given event notification. The *Inform Event* button displays another screen that allows to notify a delay due to a detour, a traffic jam, or a vehicle breakdown.

On each stop the driver must confirm to the transportation system the presence or absence of clients (*Confirm Passengers* button on Figure 6). For this, the driver's passenger confirm action is managed by the *ShowPassengersList* task which is responsible for displaying other tab with a detailed list of the inbound and outbound passengers. After the list is displayed, in the case of inbound customers the driver can confirm the presence or absence on each particular case. This action is managed by the *SendPassengerConfirmation* task which takes the responsibility of taking the client details and sending the Schedule agent an ACL Message with the notification.

Finally, the schedule through its *SendTripUpdate* task informs the Vehicle agent about changes in the itinerary. These messages are handled by the *ShowDetailsOfUpdatedTrip* task of the Vehicle agent. It is in charge of notifying the driver about a change by blinking the upper-right square led on the screen (on Figure 6) together with updating the timetable in order to reflect the changes.

4.3 The Transport Enterprise Side

The Planner is a key agent in the system's architecture. It processes all the clients' requests coming through their Trip-request agents. Therefore it manages all the proposals for a given trip request coming from the diverse Schedule agents representing each available vehicle. It is also in charge of managing inbound and outbound events coming from vehicles and customers. Such events regard the monitoring of vehicles and the modification or cancellation of a trip request.

Upon differences in the planning (due to breakdowns, traffic-jam, etc) the Schedule agent re-plans. In the case of having an infeasible trip request (mainly due to the time-window restrictions), it informs the Planner agent about the situation. The Planner makes a call for trip-proposals to try reallocating the request in other available vehicle. In any case, the result is informed to the corresponding Trip-request agent, which depending on its degree of autonomy will process the alternatives and take a decision or will inform the client about the change. This change may imply a different vehicle processing the trip only or also a delay or an anticipation of the pickup and delivery times defined previously.

Besides the Planner Agent, there is a whole set of service agents collaborating to give support to the different required functions, such as the matching of request to vehicles, the geographical data access, the accountability of the transactions and the service payment among others. From them, the most critical ones from the planning and control point of view are the Broker and the Map agents.

The Broker's main role is to know which transportation services are available and their characteristics. In addition is able to analyze those service characteristics upon planner request. It provides a publish/subscribe infrastructure that allows vehicles to enter or leave the system freely and allows clients to query the system for available transport services.

It is important to mention that the service profile gives a static description, that is, the description does not take into account the actual state of the vehicle while working. Some characteristics declared in the profile depend on the state (schedule) of the vehicle and hence, are not updated. For example, the service profile can specify that the vehicle has 4 places for wheel chair use. But this information needs to be checked, as it is possible to have all of them used during a route interval. As this information is dynamic, because it depends on the actual vehicle schedule, a further check needs to be performed afterwards in the planning layer, by the schedule agent when making a bid.

The Map agent represents the geographic area being considered where it can be a zone, a city or a part of it. The Map provides the enterprise with a series of information regarding the actual zone being covered such as localization of addresses and stops, street names and distances between localizations, among others. The map agent can be connected to a Geographic information system (GIS) to provide such information.

In our system the Map stores a graph with nodes representing the geographic area under coverage. The distance (km) and time (min) required to go from one node to each other is registered. The measure as distance is likely invariant, while the time measure representation can vary greatly along the day, specially on rush hours. Therefore the map agent can receive updated data from a traffic agent or other external sources of traffic information, allowing the Map to notify the Planner about changes on estimated travel times.

5. THE SCHEDULING PROBLEM

As stated before, during the planning process schedule agents make proposals of trip insertions which are managed by the

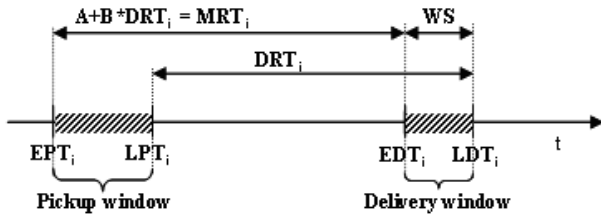


Figure 8: Time-Windows model for clients pickup and delivery intervals

planner. Therefore, each of these agents contains a scheduling heuristic to search in the state space for suitable alternatives. The underlying optimization problem and solution heuristic is explained in the following.

From the Operational Research (OR) perspective, the flexible transportation of passengers has traditionally been mentioned in literature under the name of Dial-a-Ride Problem (DARP). In the DARP problem, users formulate requests for transportation from a specific origin (pickup point) to a specific destination (drop-off/delivery point). Transportation is carried out by vehicles that provide a shared service in the sense that several users may be in the vehicle at the same time. The aim is to design a minimum-cost set of vehicle routes serving all requests. Our present implementation of DARP has included the following modelling assumptions:

Passenger Trip Duration. When dealing with passenger transportation is usual to add a limit to the length of the client's journey aboard a given vehicle. This restriction is often mentioned in literature under the name of Maximum Ride Time (MRT) and is usually proportional to Direct Ride Time (DRT), the time needed for the trip but without any deviations (shortest path). Therefore, the MRT has an specific value for each client.

Time Windows. There are different models for constructing the time windows. In our considered variant with Time Windows (DARPTW), customers specify the arriving time, becoming the Latest Delivery Time (LDT). The Latest Delivery Time (see Figure 8) constitutes the upper bound of the delivery time-windows [EDT_i, LDT_i]. The majority of real-life pickup and delivery problems are time restricted in a tight or loose way. In our case this is controlled by the systems' parameters A , B and WS allowing to vary the service Quality level to be provided. The model also considers a pickup time-window, the pair (EPT_i, LPT_i) , where $EPT_i = EDT_i - MRT_i$ and $LPT_i = LDT_i - DRT_i$. In this way, a vehicle serving the customer i must reach the pickup and delivery points within the time-windows specified for i .

Multiple Vehicles. If the service will be done by one vehicle, the corresponding problem is called the single-vehicle variant of the problem (1-DARP). If there is a fleet of vehicles available for the service, the problem is known as the multi-vehicle variant of the problem (m-DARP) which corresponds to our case.

Multiple Depots. In a number of environments, not all

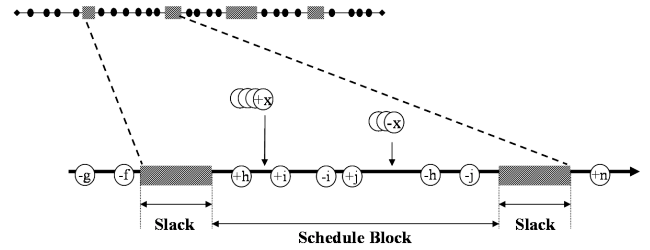


Figure 9: Work-schedule used by vehicles, consisting in sequences of schedule blocks and slacks

vehicle routes start from and end on the same depot, especially when dealing with multiple vehicles.

Vehicle Route Duration. This formulation allows for constraints on the lengths and/or durations of vehicle tours. For example, such considerations arise from constraints on the geographic coverage of a given vehicle, its refueling requirements and restrictions on the drivers' duty day (e.g. different shifts or time-blocks, lunch breaks) among others.

Dynamic Model. Static formulations assume that customer demand is known ahead in time (e.g. models assuming "advance reservations"). In contrast, in dynamic models DARP (D-DARP), new customer requests are eligible for immediate consideration, requiring revisions of already established routes and schedules. In addition, dynamicity can include delays or cancellations due to traffic-jams, accidents, vehicle breakdowns or simply a client no-show situation, all of which imply a re-planning of the original routes.

5.1 Greedy Insertion Heuristic

The scheduling algorithm used by schedule agents is based on an improved and distributed version of the ADARTW algorithm [9], a constructive greedy heuristic. Due to the dynamic nature of the problem being tackled it was necessary to make use of a solver fast enough to provide results within seconds rather than minutes, orienting the choice towards this kind of heuristics.

The algorithm finds all the feasible ways in which a new customer can be inserted into the actual work schedule of a vehicle, choosing the one that offers the maximum additional utility according to a certain objective function. Figure 9 shows a schedule block that serves 3 customers (h, i, j) while evaluating the insertion of a fourth one (customer x). Each of them has their pick-up (+) and delivery (-) stops respectively. The search must include all the schedule blocks contained in the vehicle's work-schedule. In a block with already d stops (2 per customer) there are $(d+1)(d+2)/2$ possible insertions, considering that the customer's pickup must always precede his delivery and that is not possible to pickup a client in one block and deliver him in another (because of the block's definition).

Once we have found that a possibility of insertion is feasible, it is necessary to define the actual times for those events, that is, the Actual Pick-up Time and the Actual Delivery Time. This problem is often mentioned in literature as the

scheduling problem, as once the sequence of trips (route) has been fixed the following step is to define the exact position where the sequence will be placed in time without violating the time-windows defined for each client.

Commonly, there will be a time interval in which can be inserted, meaning that the sequence can be scheduled more early or late in time within that interval. Several authors program the actual times as soon as possible for reducing the travel and waiting times of the customers, reason why our implementation does it in this way.

5.2 Work-Schedule

The model used for the vehicles' work-schedules considers that along the day a vehicle can be in any of these three states: at a depot, in travel or inactive. When the vehicle is at a depot means that it has not started its service period or has just finished it. When the vehicle is in travel, means that it is actually going to pickup or delivery passengers generating schedule blocks. As Figure 9 shows, a schedule block corresponds to a sequence of pickups and deliveries for serving one or more trip requests. A schedule block always begins with the vehicle starting on its way to pick-up a customer and ends when the last on-board customer is discharged. The third state is when the vehicle is inactive or idle generating a slack time. In this case the vehicle is parked and waiting to serve a next customer and then begin another schedule block.

Therefore, a complete vehicle's work-schedule will have periods of vehicle's utilization (schedule blocks) and inactive periods (slacks times) in which the vehicle is available and waiting.

5.3 Time-Windows Feasibility

The time-windows feasibility processing is tightly coupled to the work-schedule model. Within the checking algorithm, different restrictions need to be checked for a given potential solution. The most important ones are the time windows, the capacity constraints (on number and type) and the bounds on the duration of clients' ride and of vehicle routes.

This represented a challenging aspect of the work, as in general is difficult to find in literature the used mechanism for tackling this point. In Jaw et al. [9] is described only in general terms and most research papers state a change from the previous work but not its specific implementation.

For a block X with w events representing either a pickup or a delivery of passengers, Jaw's work presents the following calculations representing how much the events can be anticipated/posticipated in time.

$$BUP(X_i) = \text{Min}(\text{Min}(AT(X_i) - ET(X_i)), SLK_0)$$

$$BDOWN(X_i) = \text{Min}(LT(X_i) - AT(X_i))$$

$$AUP(X_i) = \text{Min}(AT(X_i) - ET(X_i))$$

$$ADOWN(X_i) = \text{Min}(\text{Min}(LT(X_i) - AT(X_i)), SLK_{w+1})$$

SLK_0 and SLK_{w+1} represent the (possible) slack periods immediately preceding and following the block respectively.

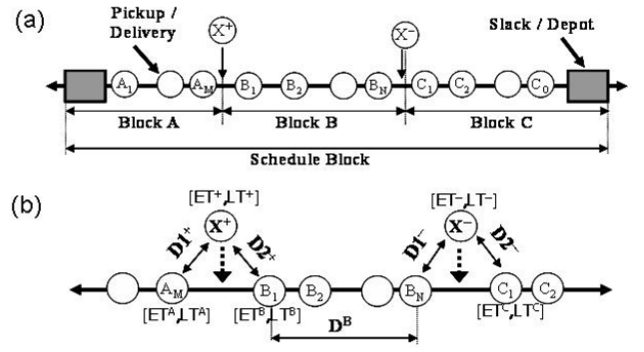


Figure 10: Time-windows feasibility-check procedure

$ET(X_i)$, $AT(X_i)$ and $LT(X_i)$ represent the Early, Actual and Latest Times of the event X_i respectively with $0 < i < w + 1$.

Our developed model is based on the Jaw's calculations on *BUP* and *ADOWN* but adds the important idea of intersecting the time windows restrictions along a piece of route, allowing to simplify the processing of the time windows feasibility check and making it possible to evaluate the insertion of the whole client (pickup and delivery) at the same time.

Therefore, in the case of the implemented insertion heuristic the starting point is the schedule block under which to evaluate the insertion of the new client. The Figure 10(a) shows a detailed view when evaluating the insertion of the pickup (X^+) and delivery (X^-) of a client. Between the pickup and the delivery are one or more events separating them and at the beginning (or ending) of the block is a slack or the bus depot. The approach is to divide the schedule block in three sub-blocks A, B and C for the events before the pickup, in between and after the delivery respectively. A special case is when both events are consecutive meaning that the block B includes only the distance from the pickup to the delivery of the new client.

The Figure 10(b) shows the time windows and distances needed for the evaluation and intersection. The interval $[ET_A, LT_A]$ represents the earliest and latest times to which the event A_M can be shifted (anticipated / posticipated) without violating the time window constraints of all the events within its block. A similar thing happens with intervals $[ET_B, LT_B]$ and $[ET_C, LT_C]$ on the events B_1 and C_1 for the blocks B and C respectively. Therefore, is needed to identify the feasible shift up and shift down for each of the three blocks. For the block A are used the *BUP* and *BDOWN* of the event A_M as they consider the previous events, while for the block C the *AUP* and *ADOWN* of C_1 are needed. For block B is needed the *AUP* and *ADOWN* for B_1 but considering only until B_N and not the events on block C as the normal calculations would. Then, for interval $[ET_A, LT_A]$ we have: $ET_A = AT(A_M) - BUP(A_M)$ and $LT_A = AT(A_M) + BDOWN(A_M)$.

A similar thing happens with $[ET_B, LT_B]$ and $[ET_C, LT_C]$. Distance $D1^+$, $D2^+$, $D1^-$ and $D2^-$ correspond to the distances between the nodes indicated by the respective arrows

in the figure. The next step is intersecting the time intervals of the three blocks and the two time windows coming from the new client’s pickup and delivery events. This intersection needs to consider the distances separating each of the five intervals. For this reason, a point in the schedule is used as reference and all the intervals are translated to that reference obtaining a single time interval $[ET, LT]$. By using the pickup event (X^+) of the new client as reference point and following Figure 10(b) are obtained:

$$ET = \text{Max}(ET_A + D1^+; ET^+; ET_B - D2^+; ET^- - D1^- - D_B - D2^+; ET_C - D2^- - D1^- - D_B - D2^+)$$

$$LT = \text{Min}(LT_A + D1^+; LT^+; LT_B - D2^+; LT^- - D1^- - D_B - D2^+; LT_C - D2^- - D1^- - D_B - D2^+)$$

This $[ET, LT]$ interval represents the feasibility area in which to set the new schedule with respect to the reference point. The actual time for the reference point (X^+ in this case) must be set and hence the actual times for the whole schedule block can be calculated as they depend on the fixed distances between one event and another. Defining the optimal place within this interval corresponds to the scheduling problem mentioned before.

6. EXPERIMENT SETUP

As mentioned earlier, the original architecture’s planning approach is based in the contract-net (CNP) under self-interested agents. Therefore, Vehicle agents pursued the optimization of the travelling costs (utility function with total slack time and total travel time) and Client agents were oriented towards the maximization of the perceived service quality (utility function with excess travel time and waiting time).

All the tests considered the same geographical net and 20 demand scenarios labeled from U1.txt to U20.txt. Each considered 50 trip requests each, distributed uniformly in a two-hour horizon. For each demand scenario 25 runs were done. Regarding the considered distributed environment, the simulations were carried out over PCs with Intel Pentium 4 of 2 GHz. with 256 MB Ram, connected through a 10/100 Mb. Router.

The following tests focus on the planning capabilities of the architecture. In this sense, the simulations consider an agent devoted to the generation of the Trip-request agents and another devoted to generating the Schedule agents. In addition, a Main agent was in charge of managing all the aspects related to the simulation control, specifically centered on the generation of the agents, request of output data and deletion operations along the diverse runs and scenarios.

The following operational decisions were adopted: 1) the same utility function and scheduling algorithm have been used for all the vehicles, 2) all the clients share the same utility function, 3) the available fleet is of 30 identical vehicles with capacity 20, 4) one depot is used for all the vehicles and 5) in all cases the effectiveness measures (utility variables) were weighted with the same value.

The generation of Trip-request agents (and hence the arrival of trip-requests) to the system follows a Poisson distri-

Table 1: Distribution of agents among hosts over the 3 scenarios

2 Hosts	3 Hosts	5 Hosts	Agents
1	1	1	Map Agent
1	2	2	Trip-Request Agents
2	3	3	Schedule Agents
1	1	4	Planner Agent
1	1	5	Broker Agent

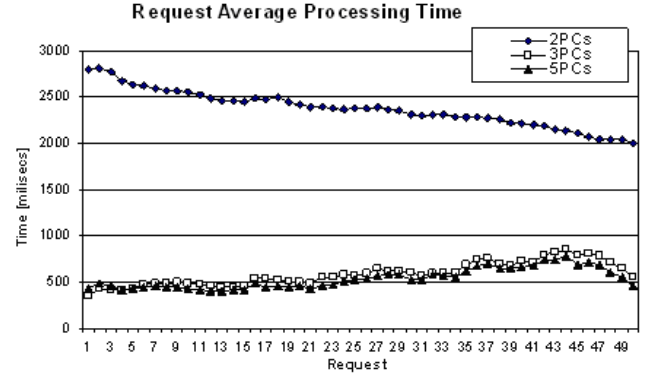


Figure 11: Processing time for trip requests ordered by arrival at Exp(3) under 2, 3 & 5 hosts.

bution. Then, the time between arrivals distributes Exponential, $E(\lambda)$, with lambda in terms of requests per second. The agents involved in the simulations were the three of the planning layer (Trip-request, Planner and Schedule agents) plus two of the service layer (Map and Broker agents) as Table 1 details.

6.1 Results

Figure 11 shows different curves for 2, 3 and 5 hosts’ configuration at a $(\lambda) = 3$ arrival rate. A big improvement exists when changing from 2-hosts to 3-hosts configuration, while little improvement is obtained when changing from 3 to 5 hosts. A closer look on how agents were distributed shows that separating Trip-request and Schedule agents from the rest has a big impact on performance, but separating the Planner, Broker and Map agents on diverse host gets only a small improvement in terms of processing time.

A second round considered contrasting the effect of changing the lambda (λ) coefficient over the performance of the planning system when processing a request. In Figure 12 are compared 2 diverse arrival rates; $\lambda = 3$ and $\lambda = 5$ requests per second for the 3-host and 5-host scenarios. The two curves in the lower part correspond to $\lambda = 3$ scenarios while the other two at the top, to $\lambda = 5$.

In contrast, the average quality of the results for all the 20 demand scenarios considered were similar. In fact the number of vehicles used and cost of the solutions provided are not significantly different among the two trip-request rates even when changing the number of hosts.

These results reflect the fact that the system gets much overloaded at a $\lambda = 5$ arrival rate, suggesting a possible

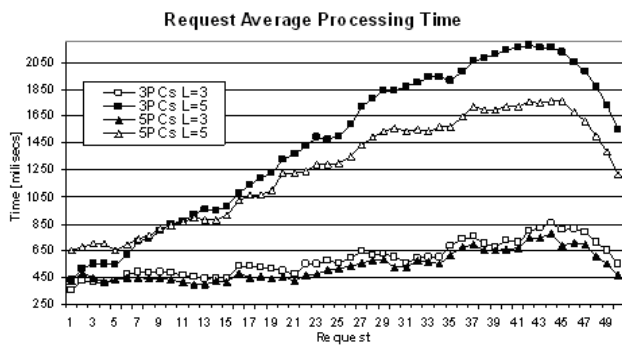


Figure 12: Processing time for trip requests ordered by arrival at Exp(3) and Exp(5) rates, over 3 and 5 hosts.

bottleneck in the architecture. In fact, the Planner Agent constitutes a central point of communication among parties, therefore for bigger-size problems the architecture needs to be scalable.

As a possible solution, the Planner could be replicated on diverse hosts and be organized hierarchically to balance the workload, allowing to improve the overall performance under such scenarios. Of course this remains a matter of further work.

7. CONCLUSIONS

An agent-based software application for managing a flexible passenger transportation systems was described. The underlying architecture provides a transparent and flexible way to make interoperate users, vehicles and support-service providers into a single application.

The methodology used allowed an appropriate level of specification along its diverse phases. The focus was centered on the specification of the main actors involved with the system: Customers, Drivers (Vehicles) and the Transportation Enterprise, providing a concrete idea of interface agents' GUIs while covering the optimization problem involved and the implemented scheduling heuristic.

A project next step involves a field test with a local group of taxis already operating under a shared mode but actually organized in frequency-based route lines. Further work considers to carry out scalability tests while replicating the Planner agent.

8. ACKNOWLEDGEMENTS

Partially funded by the Pontifical Catholic University of Valparaíso (www.pucv.cl), project No. 209.746/2007.

9. REFERENCES

- [1] Bellifemine, F. et al: JADE - A FIPA Compliant Agent Framework. C SELT Internal Technical Report, 1999.
- [2] Borndörfer, R., Grötschel, M., Klostermeier, F., Küttner, C.: Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System. Tech. Report SC 97-23, Berlin, 1997.
- [3] Burrafato, P., and Cossentino, M.: Designing a multiagent solution for a bookstore with the passi

- methodology. In 4th Int. Bi-Conference Workshop on AgentOriented Information Systems (AOIS-2002).
- [4] Cordeau, J.F., Laporte, G.: A Tabu Search Heuristic for the Static Multi-Vehicle Dial-a-Ride Problem. *Transportation Research B*, Vol. 37B, 2003, pp. 579-594.
- [5] Cubillos, C., Rodríguez, N., Crawford, B.: A Study on Genetic Algorithms for the DARP Problem. Mira, J., Alvarez, J.R. (eds.) IWINAC 2007, Part I. Springer LNCS, Vol. 4527,2007. pp. 498-507.
- [6] Cubillos, C., Gaete, S.: Design of an Agent-Based System for Passenger Transportation using PASSI. Mira, J., Alvarez, J.R. (eds.) IWINAC 2007, Part II. Springer LNCS, Vol. 4528,2007. pp. 531-540.
- [7] Ferreira, E. D., Subrahmanian E.: Intelligent Agens in Decentralized Traffic Control. IEEE ITS Conference Proceedings, August 2001, USA, pp. 705 -709.
- [8] Healy, P., Moll, R.: A New Extension of Local Search Applied to the Dial-a-Ride Problem. *European Journal of Operational Research*, Vol. 83, 1995, pp. 83-104.
- [9] Jaw, J. et al.: A heuristic algorithm for the multiple-vehicle advance request dial-a-ride problem with time windows. *Transportation Research*. Vol. 20B, No 3, 1986, pp. 243-257.
- [10] Jiamin Zhao, Dessouky, M., Bukkapatnam, S.: Distributed Holding Control of Bus Transit Operations. IEEE ITS Conference Proceedings, Oakland - USA, August 2001, pp. 976 - 981.
- [11] Kohout, R, Erol, K. Robert C.: In-Time Agent-Based Vehicle Routing with a Stochastic Improvement Heuristic. AAAI/IAAI Int. Conf. Orlando, Florida, 1999, pp. 864-869.
- [12] Madsen O.B.G., Ravn H.F., Rygaard J.M.: A Heuristic Algorithm for a Dial-a-Ride Routing and Scheduling Problem with Time Windows, Multiple Capacities, and Multiple Objectives. *Annals of Operations Research*, Vol. 60, 1995, pp. 193-208.
- [13] Nanry, W.P., Barnes, J.W.: Solving the Pickup and Delivery Problem with Time Windows using Reactive Tabu Search. *Transportation Research B*, Vol. 34B, 2000, pp. 107-121.
- [14] Ou, H. T.: Urban Traffic Multi-Agent System based on RMM and Bayesian Learning. Proc. American Control Conference 2000, pp. 2782-2783.
- [15] Perugini, D., Lambert, D., et al.: A distributed agent approach to global transportation scheduling. IEEE/WIC Int. Conf. on Intelligent Agent Technology, 2003, pp 18-24.
- [16] Smith, R. G. and R. Davis: Distributed Problem Solving: The Contract Net Approach. Proc. 2nd National Conference of the Canadian Society for Computational Studies of Intelligence. 1978.
- [17] Toth, P., Vigo, D.: Heuristic Algorithms for the Handicapped Persons Transportation Problem. *Transportation Science*, Vol. 31, No. 1, February 1997.
- [18] Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Massachusetts, USA. 1999.