Personal Travel Assistant: Intermodal Route Planning

Jürgen Lind
jli@agentlab.de

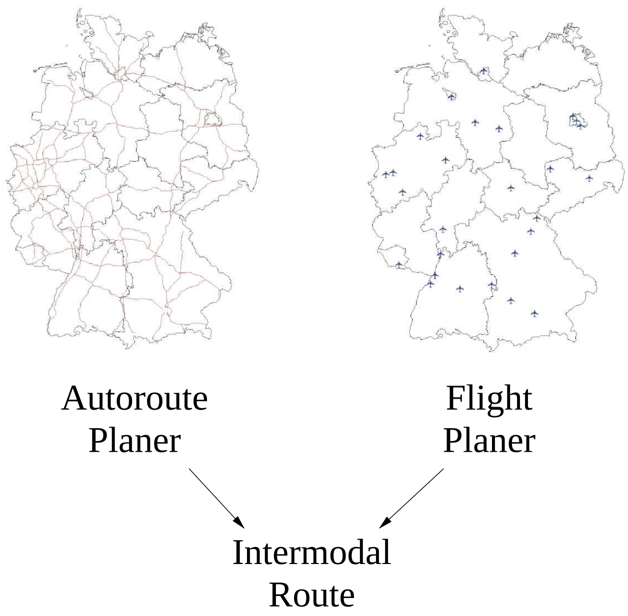
August 20, 2001

The Personal Travel Assistant (PTA) (FIPA, 1997) is a scenario described by the FIPA Standardization Organization to evaluate various aspects of the standard. The FIPA Organization is an international consortium of about 50 industrial and academic institutions whose goal is to define standards for communication among agents to ensure interoperability in industrial applications.

The agents in the PTA domain operate on behalf of their users and provide assistance in the pre-trip planning phase as well as during the on-trip execution phase. In order to accomplish this assistance, the PTA interacts with the user and with other agents that represent the available travel services. Besides the core competences such as configuration and delivery of trip planning and guidance services, the PTA also provides added-value services according to personal profiles, e.g. interests in sports, theater, or other attractions and events.

The focus of the services provided by the PTA, however, is on the basic requirements of the trip which is first of all to find the best combination of travel services that take the PTA user from his or her desired starting point to the destination. This route planning service is provided by a special agent, the *Intermodal Route Planner (IMRP)*. The IMRP is responsible for selecting a group of transport carriers that provide the necessary services at the quality required by the user. Figure 1 illustrates the basic idea in combining the services of several transportation carriers such as car and plane into a single route plan. Each of the different transportation services is encapsulated by an agent that provides a planning interface to the service. In the example in Figure 1, the user agent will query the autoroute planner for the best way from the users starting point to the airport and the air plane agent for a flight to the destination. Obviously, these two plans are not independent of each other as the flight can only be taken after the trip to the airport is complete. Additionally, the person who is traveling will not want to waste too much time because of an early arrival at the airport.

Figure 1 IMRP Example



Hence, it is the task of the PTA to find partial plans that satisfy the users constraints and to integrate these plans in order to guarantee a smooth trip.

In the following sections, I will describe a solution for the intermodal route planning process that was developed in the MoTiV-PTA project (BAYRISCHE LANDESREGIERUNG, 1996). Due to the limited space, however, I will only be able to provide the basic ideas of the design that was used to implement the final system.

1 Environment View

The environment view of the agents within the IMRP domain is determined by the standards that are set by the FIPA. The FIPA97 (FIPA, 1997) specification provides technical standards for *Agent Management* (administration of agent systems, yellow page services, firewall technology etc.), *Agent Communication Language* (FIPA ACL based on speech-act theory, protocol specification etc.) and *Agent Software Integration* (integration of existing SW systems in agent systems (agent wrappers) etc.). This set of standards is extended in the FIPA98 (FIPA, 1998) specification by defining standards for *Agent Management Support for Mobility* (specification, configuration and handling of mobile agents), *Agent Security Management* (secure communication support, agent authentification, trusted platform specification), *Ontology Service Support* (support for the specification of the semantics of domain-specific message contents) and *Human/Agent Interaction* (interface specification, user profiling, learning about preferences).

Since the PTA domain was created to evaluate the FIPA specifications, it is clear that the resulting application must conform to the standards described in (FIPA, 1997) and (FIPA, 1998).

The FIPA standard requires a minimal framework that provides primitive services for the agent inter-operation based on the following entities. The *Agent Management Service (AMS)* provides the services for the management and administration of agents e.g. by offering a name service, the *Directory Facilitator (DF)* agent contains yellow pages with service descriptions and services and several *Agent Communication Channels (ACCs)* exist for platform-independent communication, secure communication and communication through firewalls. This minimal framework is implemented by the MECCA agent plattform (GERBER ET AL., 1999) that was used in the IMRP project. The general development environment for the target system is based on three platforms (Solaris, Linux and NT) and uses Java (SUN MICROSYSTEMS, 1999) as the implementation language. The programming model is parallel because the individual route planer agents are supposed to be physically distributed and the agents will use network communication services to exchange messages.

2 Task View

The Task view describes the task decomposition within the target system and defines the basic concepts that were used to model the problem domain. In the IMRP domain, the overall problem to be solved is to find a trip from a given start point to the destination while preserving additional constraints such as available time windows or preferred transportation means. Before we proceed to the task decomposition of the target sysetem. however, we will first develop a formalization of the IMRP domain in order to obtain a tool for writing precise requirement specifications and solution outlines.

First of all, we define the basic data structures that occur in the problem domain and that must be represented within the target system. The most important data structure for the formalization of the IMRP domain is used to represent the individual travel services. Each of these services operates on a graph structure in which the nodes represent the service access points and the weights of the links represent some abstract cost measure that is associates with traveling a particular link.

DEFINITION 2.1 (GRAPH) *A* **Graph** $G = (V, L, \alpha, \omega, \gamma)$ *consists of a set* V *of vertices, a set* L *of links, a mapping* $\alpha : L \to V, \alpha(l) = v$ *where* $v \in V$ *and* v *is a starting node of* l*, a mapping* $\omega : L \to V, \omega(l) = v$ *where* $v \in V$ *and* v *is a terminal node of* l *and a weight function* $\gamma : L \to \mathbb{R}$ *that assigns a weight to each link.*

l *is a link between two vertices* v_1 *and* v_2 *iff* $\alpha(l) = v_1 \wedge \omega(l) = v_2$*. The weight of the link is* $\gamma(l)$*. In the context of multiple graphs, let* V_G *and* L_G *denote the set of vertices and links for a graph* G*, respectively. In the following, we will use the terms vertice and node synonymously.*

Another data structure that is often used in the formalization is a list over a particular set of ground elements. The following definitions describe the basic operations that can be performed on such lists.

DEFINITION 2.2 (LIST, LIST OPERATIONS) *Let* $L = [e_1, \ldots, e_n]$ *be a* **List** *of elements over a set* E*, i.e.* $e_i \in E \quad \forall i \in \{1, \ldots, n\}$*. We write* $L = [H|T]$ *with*

- $H = e_1$
- $T = e_2, \ldots, e_n$

Let $LISTS(E)$ *be the set of all sets over* E*. The the following operations are defined on* $LISTS(E)$*:*

- $length : LISTS(E) \to \mathbb{N}$ *where* $length([e_1, \ldots, e_n]) = n$ *returns the* **length** *of a list.*
- $head : LISTS(E) \to E$ *where* $head([H|T]) = H$ *returns the* **head** *element of a list.*
- $tail : LISTS(E) \to LISTS(E)$ *where* $tail([H|T]) = [T]$ *returns the* **tail** *element of a list.*

Furthermore, we define

- $heads : LISTS(LISTS(E)) \to LISTS(E)$ *where*
  $heads([[H_1|T_1], \ldots, [H_n|T_n]]) = [H_1, \ldots, H_n]$
- $tails : LISTS(LISTS(E)) \to LISTS(LISTS(E))$ *where*
  $tails([[H_1|T_1], \ldots, [H_n|T_n]]) = [[T_1], \ldots, [T_n]]$

After these basic data structure definitions, we can now turn to the actual formalization of the concepts of the IMRP domain. First of all, we formalize the services that are restricted to individual travel services.

DEFINITION 2.3 (UNIMODAL MAP) *A* **unimodal map** *is a graph G according to definition 2.1, $UMAPS$ is the set of all unimodal maps.*

We define a predicate $valid : UMAPS \rightarrow \{true, false\}$ on the set of unimodal maps that decides the **validity** *of a map in a given context.*

The validity of a route is a personalized function that can be used to express user preferences with respect to a particular travel service. We will not define this function as it is only an interface to higher level functionality of the PTA that is not of interest in the IMRP domain. Next, we define how a particular route is represented such that it can be uniquely identified within a intermodal route.

DEFINITION 2.4 (UNIMODAL ROUTE) *A sequence $R = (v_1, \ldots, v_n)$ is a* **unimodal route** *in a map M iff*

- $v_i \in V_M \quad \forall i \in \{1, \ldots, n\}$ *and*
- $v_i \neq v_j \quad \forall i, j \in \{1, \ldots, n\} : i \neq j$ *and*
- $\exists l \in L_M : \alpha(l) = v_i \wedge \in \omega(l) = v_{i+1} \quad \forall i \in \{1, \ldots, n-1\}$

We call a tuple $((v_s, \ldots, v_d), G)$ with $G = \sum_{l \in L(v_s, \ldots, v_d)} \gamma(l)$ a **weighted (unimodal) route** *and $U = (M, R, G)$ a* **named weighted (unimodal) route** *in M.*

Furthermore let $Routes(M)$ denote the Set of all Routes in M and the set of all links of a particular route is $Links : Routes(M) \rightarrow L_M^n$ with

$$Links(R) = \{l | l \in L_M \wedge \exists v \in V_M : (\alpha(l) = v \vee \omega(l) = v)\}$$

$\Gamma : Routes(M) \rightarrow \mathbb{R}$ *is defined as*

$$\Gamma(R) = \begin{cases} MAXWEIGHT & if \quad v_s = v_d \quad or \quad R = () \\ G & otherwise \end{cases}$$

Note that we have defined the weight of a route to be $MAXWEIGHT$ in the case that the start and the goal node are the same. This trick allows us to simplify the algorithms given below as it automatically rules out empty routes in an intermodal route without an extra check simply because of the high costs that are associated with any intermodal route that contains empty unimodal routes.

Now that the individual travel services have been formalized, we must define how the PTA user can switch from one travel modality to another. Thus, we must define how we describe access points that are shared by several travel services.

DEFINITION 2.5 (JUNCTION POINTS) *The* **junction points** *between two maps* M_1 *and* M_2 *are defined as*

$$J(M_1, M_2) = \{(v_1, v_2)|v_1 \in V_{M_1} \wedge v_2 \in V_{M_2} \wedge samePlace_{M_1 M_2}(v_1, v_2) = true\}$$

Where the predicate $samePlace_{M_1 M_2} : V_{M_1} \times V_{M_2} \to \{true, false\}$ *decides whether two nodes occupy the same position or not.*

Furthermore let

$$J_{M_1 M_2}(v) = \{w|v \in V_{M_1} \wedge w \in V_{M_2} \wedge (v, w) \in J(M_1, M_2)\}$$

be a function that returns all junction points for a given node and fixed source and destination maps and let $Alias : V \to V$

$$Alias(v) = \{w|w \neq v \wedge \exists M_1, M_2 : samePlace_{M_1 M_2}(v, w) = true\}$$

be a functions that returns all nodes that are equivalent (with respect to the $samePlace$ *predicate) to a give node. Note that* v *itself is contained in this set as well.*

The actual decision whether two nodes of different maps represent a junction point is encapsulated in a specific predicate $samePlace$ in order to make it easier to explicitly define junction points. A very straightforward definition of the predicate is to base it on the Euclidean distance of two nodes, but more sophisticated schemes can be implemented as well.

In the next steps in the formalization of the IMRP domain, we will define how several individual travel services can be combined in a single intermodal route. Therefore, we will first of all combine all individual travel services in a joint map in order to ease the reasoning process over several services.

DEFINITION 2.6 (INTERMODAL MAP) *A* **intermodal map** *is a set of graphs* $(G_1, \ldots, G_n)$ *with*

- $L_{G_i} \cap L_{G_j} = \emptyset \quad \forall i, j \in \{1, \ldots, n\} : i \neq j$, *and*
- $V_{G_i} \cap V_{G_j} = \emptyset \quad \forall i, j \in \{1, \ldots, n\} : i \neq j$

Furthermore let $MAPS$ *denote the set of all intermodal maps and let* $Map : \bigcup_{G \in \{G_1, \ldots G_n\}} V_{G_i} \to UMAPS$ *be a function that maps a node to the map that contains that node.*

Now, we can express an intermodal route in terms of this underlying data structure.

DEFINITION 2.7 (INTERMODAL ROUTE) *Let* $U = (M, R, G)$ *be a named weighted unimodal route in* M *and define further*

- $first(U) = first(M, R, G) = first(M, (v_1, \ldots, v_n), G) = v_1$ *and*
- $last(U) = last(M, R, G) = last(M, (v_1, \ldots, v_n), G) = v_n$ *and*
- $V(U) = R$ *and*
- $M(U) = M$.

A sequence of unimodal routes $MMR = (U_1, \ldots, U_n)$ *is called* **intermodal route** *between* v_1 *and* v_2 *iff*

- $v_1 = first(U_1)$
- $(last(U_i), first(U_{i+1})) \in J(M(U_i), M(U_{i+1})) \forall i \in \{1, \ldots, n-1\}$
- $v_2 = last(U_n)$

Let $R(MAPS)$ *be the set of all intermodal routes and* $\Gamma : R(MAPS) \to \mathbb{R}$ *be defined as*

$$\Gamma(MMR) = \begin{cases} MAXWEIGHT & if \quad MMR = () \\ \sum_i weight(U_i) & otherwise \end{cases}$$

The **normal form** $\downarrow: R(MAPS) \to R(MAPS)$ *of an intermodal route* $MMR = ((M, R, G)_1, \ldots (M, R, G)_n)$ *is defined as*

$$MMR \downarrow = ((M, R, G)_{i_0}, \ldots (M, R, G)_{i_m}))$$

with

- $i_0 < i_1 < \cdots < i_m$
- $G_{i_j} \neq \emptyset \forall j$

In the above definition we require for any intermodal route that junction points exist between any two successive unimodal routes that are combined in the intermodal route. We also define a normal form on intermodal routes in order to able to compare two intermodal routes and to rule out such routes that contain empty unimodal routes.

In the last step of the formalization, we will define how to express a request for an intermodal route in terms of the unimodal maps involved.

DEFINITION 2.8 (ROUTING SCENARIO) *A sequence* $(M_1, \ldots, M_n)$ *of maps is called a* **routing scenario** *in the set of unimodal maps* ($UMAPS$) *if the following conditions hold:*

- $v_s \in M_{i1} \wedge v_d \in M_{in_i}$ *for* $i = 1, \ldots m$ *and*
- $\exists v_3, v_4 : (v_3, v_4) \in J(M_{ij}, M_{ij+1}) \, \forall i, j : i \in \{1, \ldots, m\}, j \in \{1, \ldots n_i - 1\}$

Let $RS(UMAPS)$ *denote the set of all possible routing scenarios.*

The function $RS : V \times V \rightarrow RS(UMAPS)$ *returns for any two nodes* v_s *and* v_d *a list of possible routing scenarios* $((M_{11}, \ldots, M_{1n_1}), \ldots, (M_{m1}, \ldots, M_{mn_m}))$.

Furthermore we define $filter : RS(UMAPS) \rightarrow RS(UMAPS)$ *as follows*

$$filter((M_1, \ldots, M_n)) = \begin{cases} (M_1, \ldots, M_n) & valid(M_i) \quad \forall i \in \{1, \ldots, n\} \\ \emptyset & otherwise \end{cases}$$

$filter$ *is extended on a sequence of routing scenarios by defining* $filter : RS(UMAPS)^n \rightarrow RS(UMAPS)^m$ *as*

$$filter(RS_1, \ldots, RS_n) = (filter(RS_1), \ldots, filter(RS_n))$$

The RS function serves as interface to some higher level service that provides all possible routing scenarios between any two nodes of any unimodal travel services and the $filter$ function operates as interface to the user preferences just like the $valid$ predicate defined above.

Nonfunctional Requirements The first nonfunctional requirement for the target system is that its architecture and the services provided must be FIPA compliant. Furthermore, the answering time of the target system should lie within a range that allows the user the on-line planning of a trip, thus the answering time will typically range within a few minutes.

The final nonfunctional requirement is to evaluate two different forms of interaction, the first one of which is a central planning approach that features one intermodal route planner that uses the services provided by the unimodal planners and the other one being a distributed scheme where each unimodal planner can become an intermodal planner upon a user request.

In this view, we have formalized the IMRP domain in terms of functional abstractions that can be used by the subsequent views to develop a particular design for the application. We have also defined the nonfunctional requirements of the target system and developed the basic problem solving capabilities that are necessary in the PTA domain. In the next section, we turn to the environment in which the application will be located.

3 Role View

In the IMRP application, we can identify two functional roles. The *Unimodal Route Planner* is responsible for providing access to a particular travel resource and provides a planning service for this resource as defined below.

DEFINITION 3.1 (UNIMODAL ROUTE PLANER) *A function* $rp : V_M \times V_M \rightarrow R(M)$ *for a map* M *that returns a weighted route for any two nodes* $v_1, v_2 \in M$*, i.e.* $rp(v_1, v_2) = ((v_1, \ldots, v_2), G)$*, is called a* **(unimodal) route planer** *for* M
Furthermore, the following conditions must hold:

- $(v_1, \ldots, v_2)$ *is a route from* v_1 *to* v_2 *and*
- G *is minimal.*

The only atomic problem solving capabilities that is required for this role is the ability to perform unimodal route planning by either planning from first principles or by using an existing service and providing the result to the other agents in the system.

The second roles that occurs in the scenario is the *Intermodal Route Planner* that combines the services of various unimodal route planners in a single plan.

DEFINITION 3.2 (INTERMODAL ROUTE PLANER) *A function* $imrp : V_M \times V_M \rightarrow R(MAPS)$ *for a given set of maps is a* **intermodal route planer** *iff* $imrp$ *returns a list of unimodal maps* $((M, R, G)_1, \ldots, (M, R, G)_n)$ *for any two nodes* v_1 *and* v_2*, i.e.* $imrp(v_1, v_2) = ((M, R, G)_1, \ldots, (M, R, G)_n)$.

The process of combining the services of the unimodal route planners into a single plan works as follows: On the first level of the planning process all possible routes between all alias-nodes for the source node and alias-nodes of the destination node are computed according to the following specification.

$$imrp(v_s, v_d) = \bigcup_{v_1 \in Alias(v_s) \times v_2 \in Alias(v_d)} imrp\_na(v_1, v_2)$$

The implementation is shown in Algorithm 1.

Algorithm 1 $imrp(v_s, v_d)$

for all $v_1 \in Alias(v_s)$ **do**
  for all $v_2 \in Alias(v_d)$ **do**
    $result := result \cup imrp\_na(v_1, v_2)$
  end for
end for
return $result$

Algorithm 2 $imrp\_na(v_s, v_d)$

for all $rs \in RS(v_s, v_d)$ **do**
  route := $imrp\_sc(v_s, v_d, rs)$
  if $route \notin result$ **then**
    $result := result \cup \{route\}$
  end if
end for
return $result$

Alias nodes exist iff the source or the destination node are junction points with another map. In this case, all routes between all combinations of alias nodes must be computed.

On the second level of the planning process, all routes for all routing scenarios between two alias nodes are computed according to the following formula that is implemented by Algorithm 2.

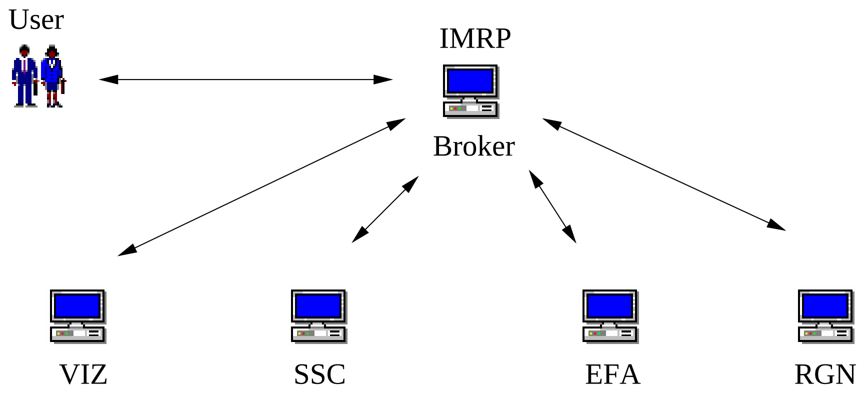
$$imrp\_na(v_s, v_d) = \bigcup_{rs \in RS(v_s, v_d)} imrp\_sc(v_s, v_d, rs)$$

The final design of the PTA/IMRP application uses two versions of role aggregation in order to evaluate the advantages and disadvantages of one approach or the other.

Broker In this version of the IMRP application, the functional roles are assigned to different agents, i.e. there is one IMRP agent in the scenario that processes the user request and then tries to find UMRP agents that provide services that match the users requirements.

Distributed The distributed version of the application does not assign a fixed functional role to an agent, each agent can play both functional roles depending on the context in which it is incorporated into the joint planning process. If the agent is queried by the user, it acts as the IMRP the tries to

Figure 2 IMRP Broker Architecture



integrate services provide by other agents with it own local service in order to generate a route plan. If, on the other hand, it is queried by another agent acting as IMRP, it offers its local service as normal UMRP.

4 Interaction View

The agent interaction that is necessary within the IMRP domain is limited to a rather simple requester-provider protocol. One agent will request a particular service from another agent which will serve the request and return the result to the requester. The application does not feature any market mechanisms or iterative refinements in the agent interaction process and is therefore straightforward to implement.

I have mentioned earlier that the IMRP application uses two different approaches to compute an intermodal route for a given routing scenario. In the following, I will describe how the planning process works for each of these different approaches. The control flow of the broker version is shown in Figure 2: the user request the a route planning service from the IMRP agent that in turn forwards partial requests to the UMRP agents and assembles the results according to the user preferences. The computation uses a recursive algorithm based on the following idea:

- If the source and the destination node are in the same map or if the routing scenario consist of a single entry, the route can be computed by an individual route planner agent. Note that a routing scenario can be of length 1 only if the destination node is a junction point to the map of the source node.

- If the source and the destination node are in distinct maps, the optimal route (wrt. to a given optimization criterion) is the minimum of all routes between the source node and the intermodal route starting at any possible junction point to the next map.

The formal description of the algorithm is given below. The implementation is shown in Algorithm 3.

$$imrp\_sc(v_s, v_d, rs) = \begin{cases} rp_{Map(v_s)}(J_{M_{v_s} M_{v_d}}(v_s), v_d) & \begin{matrix} Map(v_s) = Map(v_d) \\ or \quad |rs| = 1 \end{matrix} \\ \min\limits_{(j_1, j_2) \in J(M_1, M_2)} \begin{matrix} append(rp_{M_1}(v_s, j_1), \\ imrp\_sc(j_2, v_d, tail(rs))) \end{matrix} & otherwise \end{cases}$$

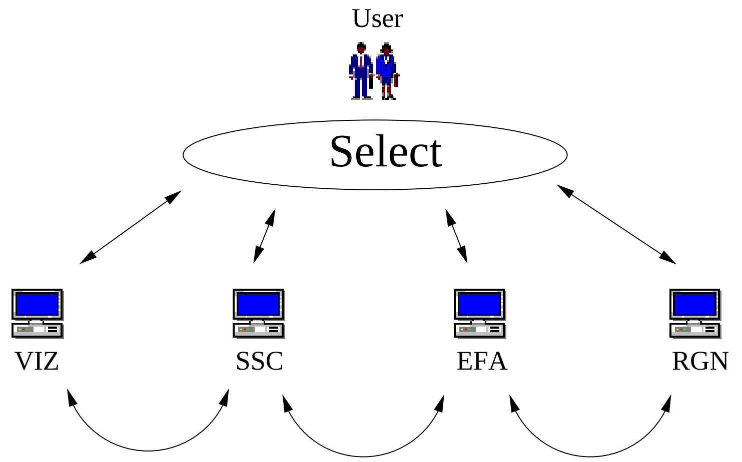
Algorithm 3 $imrp\_sc(v_s, v_d, rs)$ (Broker Architecture)

Require: $rs \in RS(v_s, v_z)$

if $Map(v_s) == Map(v_d) || (|rs| == 1)$ **then**
    $minroute := rp_{Map(v_s)}(J_{M_{v_s} M_{v_d}}(v_s), v_d)$
else
    $minroute := \emptyset$
    for all $(j_1, j_2) \in J(M_1, M_2)$ **do**
        $route := rp_{M_1}(v_s, j_1) + imrp\_sc(j_2, v_d, tail(rs))$
        if $\Gamma(route) < \Gamma(minroute)$ **then**
            $minroute := route$
        end if
    end for
end if
return $minroute$

In the distributed version of the application, the control flow in the application is as shown in Figure 3: The user selects the most appropriate UMRP agent (e.g. the service wrapper for the first service that the user intends to travel with) and places a request. The UMRP will then play the role of the IMRP in subsequent service requests to other UMRP agents and return the result to the user.

Figure 3 Distributed IMRP Architecture



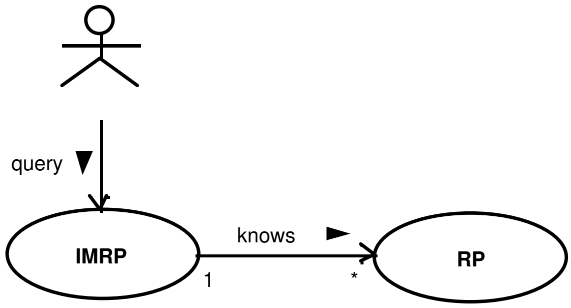
The algorithm that implements this flow of control is specified by the following equation. The implementation is shown in Algorithm 4.

$$imrp\_sc(v_s, v_d, rs) = \begin{cases} rp(v_s, v_d) & Map(v_s) = Map(v_d) \\ \min\limits_{(j_1, j_2) \in J(M_1, M_2)} \begin{array}{l} append(rp(v_s, j_1), \\ RP_{M_2}(j_2, v_d, tail(rs))) \end{array} & otherwise \end{cases}$$

Algorithm 4 $imrp\_sc(v_s, v_d, rs)$ (Distributed Architecture)

if $Map(v_s) == Map(v_d) || (|rs| == 1)$ **then**
  $minroute := rp_{Map(v_s)}(J_{M_{v_s} M_{v_d}}(v_s), v_d)$
else
  $minroute := \emptyset$
  for all $(j_1, j_2) \in J(M_1, M_2)$ **do**
    $route := rp_{M_1}(v_s, j_1) + RP_{M_2}(j_2, v_z, tail(RS))$
    if $\Gamma(route) < \Gamma(minroute)$ **then**
      $minroute := route$
    end if
  end for
end if
return $minroute$

Figure 4 Basic Society Structure



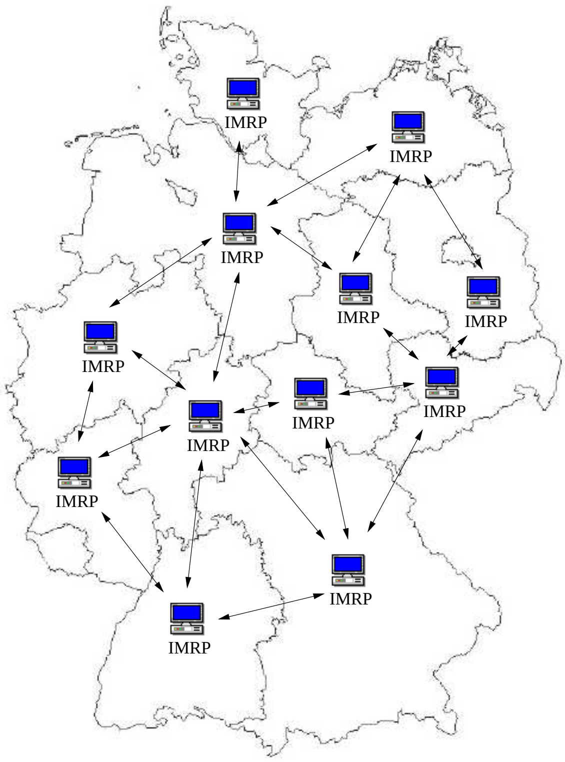
5 Society View

The society that operates in the IMRP domain is rather simple and consists of two or three different entities, depending on which of the alternatives (broker or distributed version) is considered. In either case, however, the structure of the society is initially flat as shown in Figure 4. The user connects to the IMRP agent which in turn calculates an intermodal route for the user request by connecting the individual travel services and then returns the intermodal travel plan to the user.

This simple form of the society structure is adequate as long as the number of individual travel services is relatively small. However, in a scenario with a large amount of individual travel service, e.g. when each UMRP represents some local public transport facility, this society structure may no longer be adequate. The IMRP agent cannot compute an intermodal travel plan in a one-shot process but it must connect some intermediate IMRP agents that interface to the local travel services. The respective society structure as shown in Figure 5 results in a control flow that operates at several hierarchy levels. First of all, the user queries the local IMRP agent that is responsible for the area the user is currently located. Next, this IMRP agent forwards part of the user request to other local IMRP agents that compute a partial intermodal route for their area and return the solution to the first IMRP agent that assembles a full intermodal route for the user request and returns the solution to the user.

In this section, we have modeled the society structure of the IMRP application and outlined some extension to the structure as it will be implemented in a the first version of the system. In the next section, we will see how the various features are transformed into a coherent system architecture.

Figure 5 Hierarchical IMRP Organization



6 Architecture View

6.1 System Architecture

In this section, I will characterize the required system architecture according to the MASSIVEcharacterization scheme.

Entities The only entities that occur in the target system are agents for uni-modal route planning and intermodal route planning, respectively.

Control flow Because of the fact that the only entities are agents, the control flow within the system is determined by the Interaction view. The same applies for the information flow.

Agent management Since the problem domain was chosen as an evaluation domain for FIPA standards, the agent management will be handled according to the requirements specified there.

Communication model The system architecture should allow for a geographical distribution of the system components and therefore provide some network based message passing mechanisms. In order to be able to debug the application locally, it is desirable to have local message passing that can be traced within a single debugger.

Agent architecture The agent architecture will be discussed in the next section.

Database design Since the individual agents within the system do not share data explicitly, it is not necessary to provide a system wide database.

External components/devices The external components that must be integrated in the target system are the existing travel services that are run by external providers. These services must be encapsulated in agents in order to achieve the required degree of inter-operability.

Thus, the overall system architecture is determined by the concepts specified in the FIPA standard that is used as the general basis of the application. According to the above requirements, the MECCA agent application framework (STEINER, 1992), (GERBER ET AL., 1999) was chosen as the implementation basis for the actual system because MECCA is a FIPA compliant framework that provides libraries of pre-defined agents according to the FIPA standard and includes a generic agent architecture as well as a protocol specification language for the agents.

The target system is implemented in a fully parallel system environment where all agents are running on different host computers. However, the difficult task of debugging a fully distributed system is eased as the MECCA framework offers a transparent scheme for running the agents locally without any changes to the code.

6.2 Agent Architecture

According to the characterization MASSIVE scheme, the agent requirements are as follows.

Reasoning capabilities According to their actual role, the agents need some planning facility that enables them to find an unimodal or an intermodal route using the given travel service.

Resources limitations The resource requirements of the agents is low because the planning process mentioned above is not highly complex. However, due to the necessary encapsulation of external services, the agents will be equipped with the computational resources of a complete workstation.

Control flow The control flow within the agent is determined by the algorithms described in Section 4 and is basically a sequential one. It is therefore not necessary for the agent architecture to provide a parallel execution model.

Knowledge Representation The major knowledge structures used by the agents are the unimodal and intermodal plans. These plans must be explicitly represented in a knowledge base as they are likely to change in the course of the route planning process and because they must be transfered to a human-readable form.

Autonomy The agents in this limited part of the full scenario do not exhibit any pro-active behavior as the only react to a user request or to a request from another agent to provide some well-defined service. In the full scenario, however, the agent are supposed to become active whenever the external state as observed by the agent contains some information that may be useful for their user. Still, this is not subject of this part of the system.

User interaction The degree of user interaction is not very high for this part of the system and is limited to the route specification in form of start and destination nodes, travel times, preferred travel modalities and probably a few other things. Therefore, the agents user interface will not need any particular attention.

Temporal context The lifetime of an agent in the target system can be quite long as the unimodal route planners are supposed to encapsulate real-world services that are available 24 hours a day. However, the agents do not need any persistence mechanisms because these are already present in the underlying services.

Because of the above requirements, the MECCA agent architecture was the first choice as agent architecture. MECCA agents are specifically designed to serve as wrapper for proprietary software systems that are to be used in a multiagent systems and are thus well-suited for the encapsulating of the individual travel services. A MECCA agent consists of three parts as shown in Figure 6.

The agent *body* encapsulates application specific functionalities e.g. existing SW systems, databases etc., the *head*: contains the goals and plans of the agent and controls the body via the application interface. Furthermore, it also contains the protocol execution unit that can manage several contexts at the same time. The *communicator*, finally, is the physical communication interface that implements low-level communication protocols such as TCP/IP, GSM or IIOP.

Figure 6 MECCA Agent Architecture

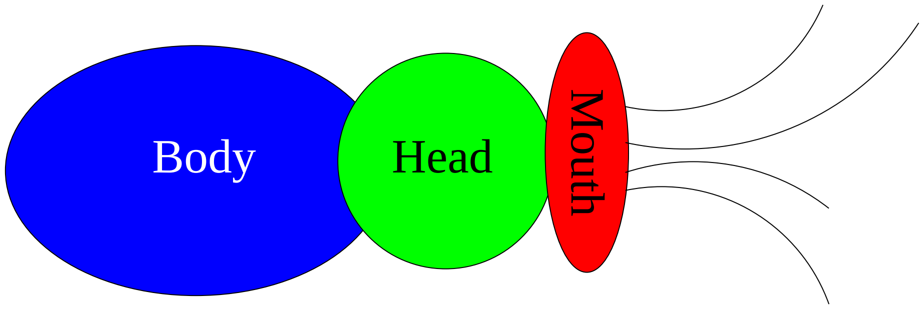
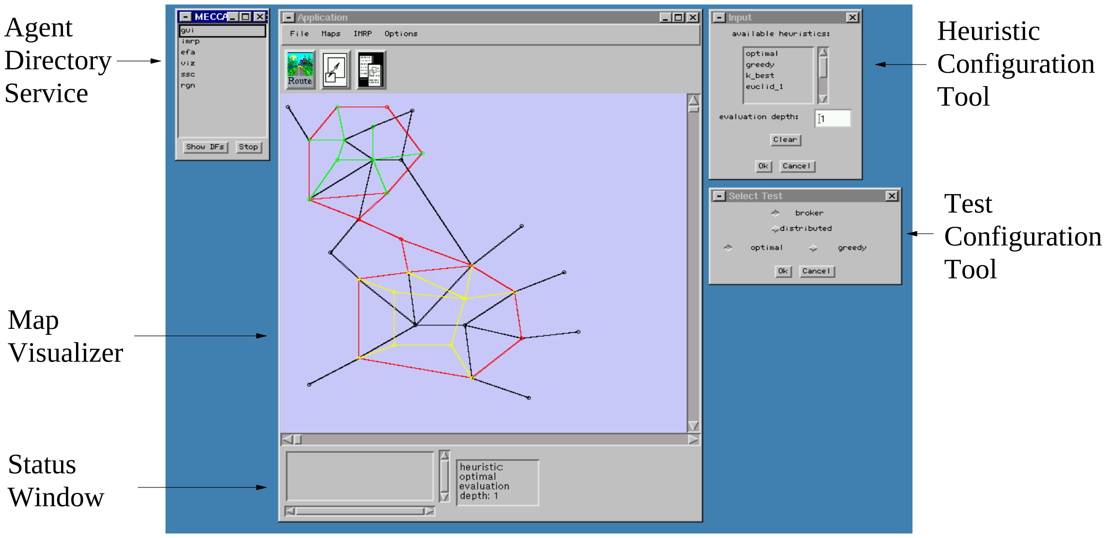


Figure 7 IMRP Graphical User Interface



7 System View

7.1 Graphical User Interface

The User interface of the IMRP system as shown in Figure 7 consists of five major elements that are briefly described below.

Map Visualizer The Map visualizer window contains the graphical representation of the unimodal maps that are managed by the unimodal route planners. The user selects the start and the destination node of the journey and is then queried for the temporal constraints. The system will then calculate an intermodal route according to the provides specification.

Test Configuration Tool In order to evaluate the different approaches presented earlier, the user can select between the broker and the distributed variant and between the optimal computation of the intermodal route and the use of a heuristic as is described in Section 7.2.

Heuristic Configuration Tool This window allows the user to select the heuristic and the parameters of the heuristic. The heuristics that can be used as well as their parameter settings are described in Section 7.2.

Status Window The Status window informs the user about the result of the computation (e.g. the length or the duration of the intermodal route) and about exceptional situations such as constraint inconsistencies that occur e.g. when the time window is to tight for particular start and destination nodes.

Agent Directory Service This window, finally, is provided by the FIPA compliant MECCA agent platform and contains the names of the agents currently known to the system. The user can click on the agent name to obtain additional information about the respective agent.

Since the intermodal route planning facility is only a service that is provided in a larger context, the user interface is only very fragmentary. It is, however, functional enough to enable the user to experiment with the different variants of the the system. Experiments that were conducted with early versions of the system showed, that the system performance might become unacceptable if the number of unimodal maps and the size of these maps increases beyond a particular point. Therefore, several attempts were made to obtain rather good – though not optimal – solutions with a reduced effort. The results of these attempts are discussed in the following section.

7.2 Performance Engineering

Suboptimal Solutions Until now, the IMRP agent has computed optimal routes for each user request which implied the exploration of the entire search space. In this section, we present some search heuristics that aim at finding sufficiently good solutions with less computational effort. The main idea of the search heuristics discussed below is a flexible steering of the search process by narrowing the set of junction points between two adjacent maps in a routing scenario.

Therefore, we define a selection function for the junction points that should be expanded during the search process as follows.

DEFINITION 7.1 (SELECTION FUNCTION) *Define* $select : (V,V)^n \rightarrow (V,V)^m, n \leq m$ *as* $select(\{v_i, v_j | (v_i, v_j) \in J(M_k, M_{k+1})\}) = \{v_o, v_p | (v_o, v_p) \in J(M_k, M_{k+1})\}$ *i.e.* $select$ *chooses an arbitrary subset of the junction points for any two maps.*

Now, by using the $select$ function from definition 7.1, we extend the search algorithm such that it expands only the selected nodes. In the following, this extension is implemented using the broker architecture discussed earlier, the changes apply in an analogous way to the distributed architecture. The modified broker algorithm is given below, the implementation is shown in Algorithm 5.

$$imrp\_sc(v_s, v_z, rs) = \begin{cases} rp_{Map(v_s)}(J_{M_{v_s} M_{v_z}}(v_s), v_z) & \begin{array}{c} Map(v_s) = Map(v_z) \\ oder \quad |rs| = 1 \end{array} \\ \min\limits_{\substack{(j_1, j_2) \in \\ select(J(M_1, M_2))}} \begin{array}{l} append(rp_{M_1}(v_s, j_1), \\ imrp\_sc(j_2, v_z, tail(rs))) \end{array} & otherwise \end{cases}$$

Algorithm 5 $imrp\_sc(v_s, v_z, rs)$ with selection function (Broker Architecture)

Require: $rs \in RS(v_s, v_z)$
if $Map(v_s) == Map(v_d) || (|rs| == 1)$ **then**
  $minroute := rp_{Map(v_s)}(J_{M_{v_s} M_{v_d}}(v_s), v_d)$
else
  $minroute := \emptyset$
  for all $(j_1, j_2) \in select(J(M_1, M_2))$ **do**
    $route := rp_{M_1}(v_s, j_1) + imrp\_sc(j_2, v_d, tail(rs))$
    if $weight(route) < weight(minroute)$ **then**
      $minroute := route$
    end if
  end for
end if
return $minroute$

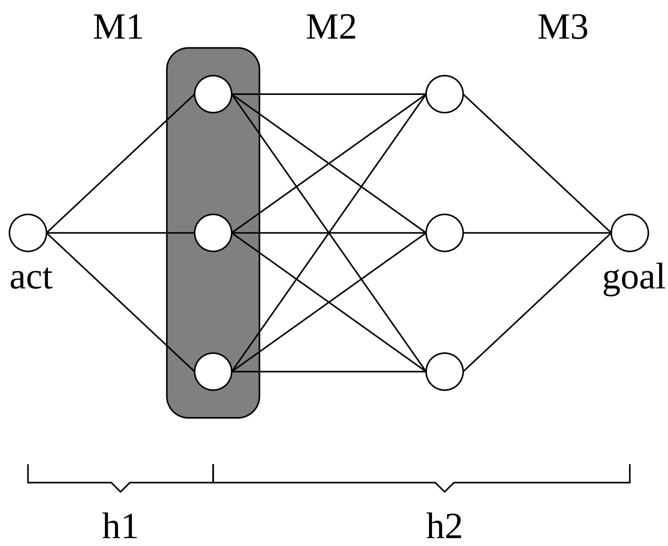
As we have seen, the integration of the selection function is quite straightforward but it is yet unspecified how a particular selection might work to produce satisfactory results. Therefore, we first of all define a *generic selection function* and then instantiate this generic function to obtain specific selection functions.

The generic $select$ function operates in three steps:

(1) Generate a partial ordering on the set of possible junction points.

(2) Sort the junction points according to the partial ordering.

Figure 8 Generic Selection Function



(3) Select the best candidates.

Figure 8 illustrates the process. In the first step, we assign a weight to each junction point x with respect to the current search node act and the destination node $gaol$ according to

$$g(x) = h_1(act, x) + h_2(x, goal)$$

where h_1 estimates the weight of the route from the current node to the junction point and h_2 estimates the weight of the route between the junction point and the goal. Now we can sort the list according to the combined weights and select the best nodes according to a given evaluation depth.

In the following description, we discuss some instantiations of the generic $select$ function by specifying the two estimation functions h_1 and h_2 and the evaluation depth of the final selection step.

Optimal Selection This instantiation realizes the standard case without narrowing the search space. All possible junction points are selected which guarantees, that the optimal solution is found. This instantiation is mainly included for the sake of completeness.

$$h_1(act, x) = weight(rp(act, x))$$
$$h_2(x, goal) = weight(rp(x, goal))$$

Obviously, all junction points must be selected in order to guarantee optimality in the search process.

Greedy Search This variant of the $select$ function considers only the locally best alternatives. Therefore, we first define the following concepts.

DEFINITION 7.2 (K-BEST SEARCH) **k-best search** *expands only the first* k *junction points that have the shortest distance to the source node of the route.*

DEFINITION 7.3 (K-BEST JUNCTION POINTS) $topJPs : UMAPS \times UMAPS \times V \times \mathbb{N} \mapsto (V,V)^n$ *is defined as*

$$\begin{aligned}
&topJps(M_1, M_2, v, k) = ((j_{11}, j_{12}), \ldots, (j_{k1}, j_{k2})) \textit{ with}\\
&(\forall i \in \{1, \ldots, k\} : (j_{i1}, j_{i2} \in J(M_1, M_2) \wedge\\
&(\exists l \in L_M, i \in \{1, \ldots, k\} : \alpha(l) = v \wedge \in \omega(l) = j_{i1}) \wedge\\
&(\forall j \notin j_{il} \forall i \in \{1, \ldots, k\} : \gamma(v, j) > \gamma(v, j_{i1})))
\end{aligned}$$

We obtain a greedy algorithm by setting $select(J(M_1, M_2)) = topJPs(M_1, M_2, v_s, 1)$. This corresponds to the following setting

$$\begin{aligned}
h_1(act, x) &= weight(rp(act, x))\\
h_2(x, goal) &= 0
\end{aligned}$$

In the second step of the selection process, only the first junction point in the ordered set is expanded, an obvious extension of the algorithm is to select not only the best junction point, but the k best junction points by setting $select(J(M_1, M_2)) = topJPs(M_1, M_2, v_s, k)$ for some fixed value of k.

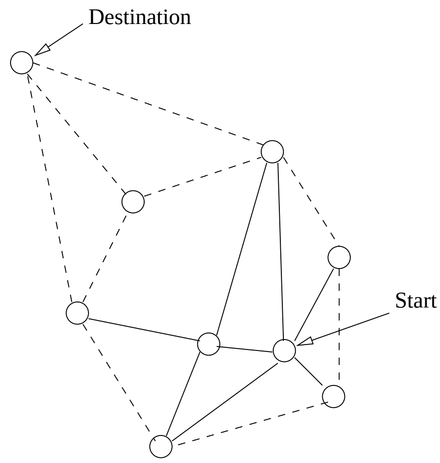
Euclidean Distance Measure The Euclidean distance between two points $p = (x_1, y_1)$ and $q = (x_2, y_2)$ is defined as

$$d(p, q) = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

From this definition, we can derive three different heuristics:

$$\begin{aligned}
h_1(act, x) &= weight(rp(act, x))\\
h_2(x, goal) &= d(x, goal)
\end{aligned}$$

Figure 1 Example 1



$$h_1(act, x) = d(act, x)$$
$$h_2(x, goal) = weight(rp(x, goal))$$

$$h_1(act, x) = d(act, x)$$
$$h_2(x, goal) = d(x, goal)$$

The k best junction points are selected from the ordered set of junction points according to the given evaluation depth.

Precedence Ordering on Maps This heuristic uses a partial ordering defined on the set of unimodal maps. We define a weight function $w : UMAPS \rightarrow \mathbb{R}$ that assigns a weight to each unimodal map. We set

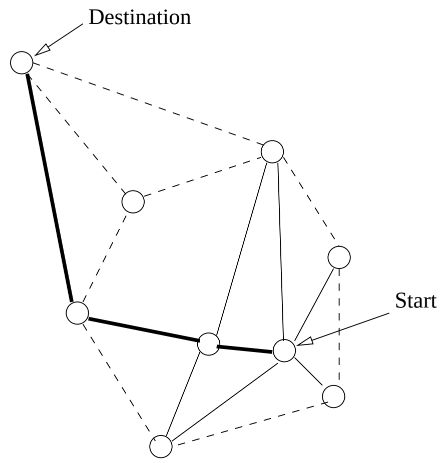
$$h_1(act, x) = weight(rp(act, x)) * w(Map(act))$$
$$h_2(x, goal) = weight(rp(x, goal)) * w(Map(x))$$

to obtain the map precedence heuristic.

Figure 1 shows an example with two maps indicated by the solid and the dashed line, respectively. The solid line map represents the public transport service of a city, whereas the dashed map represents the interregional individual traffic.

Using this heuristic, a route query from "source" to "destination" will yield a solution as it is shown in Figure 2. This solution reflects the shortest path from the source to the destination node, but it neglects the fact that

Figure 2 Example 2



the public transport is usually much more time consuming then individual traffic. Generally speaking, it is often more efficient to leave the city as fast as possible and to use high speed individual traffic routes around the city.

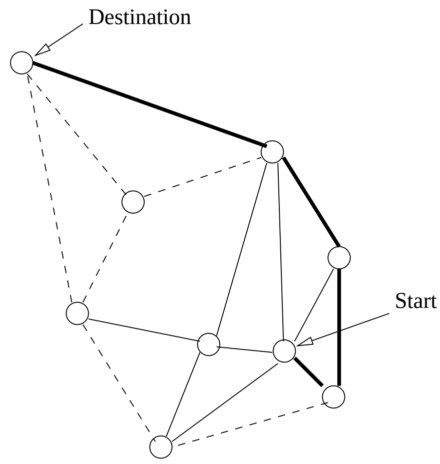
Using a precedence ordering on maps (and implicitly on means of transport), the higher preference for individual transport towards public transport yields the result presented in Figure 3.

Caching The maximum number of unimodal routes that must be computed for an intermodal route $(U_1, \ldots, U_n)$ is a function of the number of junction points between the unimodal maps for a particular routing scenario. For a given routing scenario $rs = (M_1, \ldots, M_m)$, the number of possible unimodal routes $(\#_{rs})$ computes as follows:

$$\begin{aligned}
\#_{rs}(v_s, v_z, ()) &= 0 \\
\#_{rs}(v_s, v_z, (M)) &= 1 \\
\#_{rs}(v_s, v_z, (M_1, \ldots, M_m)) &= |J(M_1, M_2)| + \\
&\quad \sum_{(j_1, j_2) \in J(M_1, M_2)} \#_{rs}(j_2, v_z, (M_2, \ldots, M_m))
\end{aligned} \tag{2}$$

The recursive formula (2) is not optimal for practical purposes and therefore we also give the number of possible unimodal routes in a non-recursive form. Note that we define $|J(M_m, M_{m+1})| = |J(M_{m+1}, M_{m+2})| = 1$ in Formula 3.

Figure 3 Example 3



$$\#_{rs}(v_s, v_z, rs) = \sum_{i=1}^{m+1} \prod_{j=1}^{j \leq i} |J(M_j, M_{j+1})| \tag{3}$$

Comparing this to the number of *unique* unimodal routes which computes to

$$\#_{unique}(v_s, v_z, rs) = \sum_{i=1}^{m-1} |J(M_i, M_{i+1})| * |J(M_{i+1}, M_{i+2})| \tag{4}$$

reveals a high potential for performance gains if the re-computation of unimodal routes can be avoided. For example, given three maps with 10 junctions points between each of them, we obtain a total of 111110 unimodal routes that must be combined according to equation 3 out of which only 300 are unique according to equation 4. To take profit from this discrepancy, we define a generic cache as follows.

DEFINITION 7.4 (GENERIC CACHE) *A* **cache** *is a data structure that stores* $(key, value)$ *pairs where* $key \in KEY$ *and* $value \in VALUE$. *We define functions* $store, retrieve$ *and* $flush$ *that operate on these* $(key, value)$ *pairs.*

$store : KEY \times VALUE \rightarrow \{success, failure\}$

$$store(k, v) = success, C' = (C \backslash \{(k, v') | v' \neq v\}) \cup \{(k, v)\} \tag{5}$$

$retrieve : KEY \rightarrow VALUE$

$$retrieve(k) = \begin{cases} v & falls(k, v) \in C \\ \emptyset & sonst \end{cases} \tag{6}$$

$$flush :\to \{success, failure\}$$

$$flush() = success, C' = \emptyset \tag{7}$$

Now, whenever the IMRP needs a particular unimodal route, it first checks the cache whether it already contains the route or not. In the first case, the route is used directly whereas in the second case, the IMRP agent issues a request to the UMRP in charge and then stores the result in the cache for potential reuse.

8 Summary

This section demonstrated how the MASSIVE method was used in the travel planning domain. In the next section, I will present another case study where MASSIVE was used to build a generic software library.

References

BAYRISCHE LANDESREGIERUNG (1996). Bayerninfo. http://www.bayerninfo.de.

FIPA (1997). Fipa '97 specification parts 1–7, version 1.0. The Foundation for Intelligent Physical Agents.

FIPA (1998). Fipa '98 specification parts 1–13, version 1.0. The Foundation for Intelligent Physical Agents.

GERBER, C., STEINER, D., AND BAUER, B. (1999). Resource adaptation for a scalable agent society in the MECCA domain. In *Intelligent Software Agents for Communication Networks*. Springer.

STEINER, D. (1992). MEKKA: Eine Entwicklungsumgebung zur Konstruktion kooperativer Anwendungen. In MÜLLER, J. AND STEINER, D., editors, *Kooperierende Agenten*, number D-92-24 in DFKi Document Series, pages 17–21. DFKI, Saarbrücken.

SUN MICROSYSTEMS (1999). The Java Programming System. http://java.sun.com.