An Intelligent System for Smart Tourism Simulation in a Dynamic Environment

Mohannad Babli, Jesús Ibáñez, Laura Sebastiá, Antonio Garrido, Eva Onaindia [1,2]

Abstract. In this paper, we present a smart tourism system that plans a tourist agenda and keeps track of the plan execution. A Recommendation System returns the list of places that best fit the individual tastes of the tourist and a planner creates a personalized agenda or plan with indication of times and durations of visits. The key component of the system is the simulator in charge of the plan monitoring and execution. The simulator periodically updates its internal state with information from open data platforms and maintains a snapshot of the real-world scenario through live events that communicate sensible environmental changes. The simulator builds a new planning problem when an unexpected change affects the plan execution and the planner arranges the tourist agenda by calculating a new plan.

1 INTRODUCTION

The exponential growth of the Internet of Things (IoT) and the surge of open data platforms provided by city governments worldwide is providing a new foundation for travel-related mobile products and services. With technology being embedded on all organizations and entities, the application of the smartness concept to address travellers' needs before, during and after their trip, destinations could increase their competitiveness level [2].

Smart tourism differs from general e-tourism not only in the core technologies of which it takes advantage but also in the approaches to creating enhanced at-destination experiences [8]. In the work [14], authors identify the requirements of smart technology integration in personalized tourism experiences including information aggregation, ubiquitous mobile connectedness and real time synchronization.

Many tourism applications provide a personalized tourist agenda with the list of recommended activities to the user [12, 13, 5, 16, 15]. In many cases, these systems provide a dynamic interaction that allows the user to interact with such agenda by adding or removing activities or changing their order. Additionally, the use of GPS in mobile devices allows recommender systems to locate the future user's location and recommend the most interesting places to visit. However, most of these applications work with fixed and static information throughout the execution of the activities. In other words, they do not easily react before changes in the world; for instance, a museum that closes, a restaurant which is fully booked, a bus route that is now diverted, etc. This has critical implications on the way tourists regard their experiences. Activities, even pre-designed in advance, must be dynamically adapted and personalized in real time. One essential prerequisite for smart technology is real time synchronization, which implies that information is not limited to a-priori collection but can be collected and updated in real time [14].

Creating an agile and adaptable tourist agenda to the dynamic environment requires tracing the plan and checking that activities happen as expected. This involves plan monitoring and possibly finding a new tourist agenda organization in case some particular activity can not be realized. In this paper, we relate our experience with a context-aware smart tourism simulator.

From the monitoring and simulation perspective, there exist many frameworks for different programming languages that support discrete event-based simulators (e.g. `http://jamesii.informatik.uni-rostock.de/jamesii.org`, `http://desmoj.sourceforge.net/home.html`, `http://simpy.readthedocs.io/en/latest/`). Although they can be programmed for very particular scenarios, they fail to take a general domain description and simulate its behavior in highly dynamic environments. At this stage, planning technology can be very valuable. The Planning Domain Definition Language (PDDL) provides a simple way to define the physics of the domain (a tourism domain in our case, although it is valid for any other scenario) and the particular problem that instantiates such a domain. In PDDL we can define the activities to be executed similarly to rules, with their preconditions, effects and other interesting features like duration, cost, reward, etc. The result of using planning in a tourism domain is a plan, represented as the agenda of activities the user will follow. The plan needs to be validated, executed and adapted, if necessary, to new information. *VAL* is a plan validation tool [9] that can be used to validate and simulate a successful plan execution. However, *VAL* does not consider the dynamic changes of the world and, consequently, it cannot react to them.

In this work, we present a smart tourism system that attempts to overcome the previous limitations. Particularly, we use a PDDL tourism description that can be easily adapted to many different scenarios with new activities, preconditions, effects and points of interest. We run a planner to obtain a plan and we simulate the plan execution like in a real context, dynamically simulating changes in the environment. The simulator reacts to the changes by checking whether the real

[1] Universitat Politècnica de València, Valencia, Spain, Email: {mobab, jeibrui, lstarin, agarridot, onaindia}@dsic.upv.es

[2] This work has been partially supported by Spanish Government Project MINECO TIN2014-55637-C2-2-R

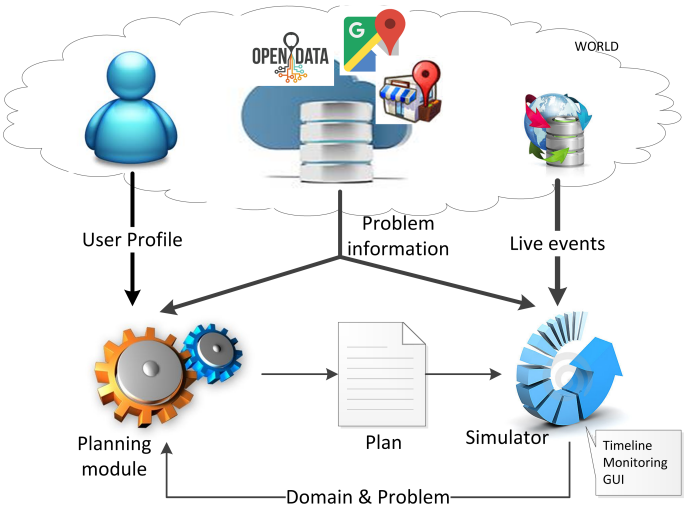

Figure 1. GLASS Architecture

world matches the expected one or not and reformulating the PDDL problem if a failure in the plan is detected, while trying to reuse as many of the original set of recommended activities as possible.

This paper is organized as follows. Next section outlines the architecture of the smart tourism system. Section 3 presents the planning description of the tourism domain and highlights the main components of a planning problem. Section 4 describes the simulator behaviour with a special emphasis on the reformulation of a planning problem. Section 5 presents a case of study of a tourist plan in the city of Valencia in Spain and last section concludes.

2 GLASS ARCHITECTURE

This work is part of the ongoing GLASS[3] (Goal-management for Long-term Autonomy in Smart citieS) project applied to a tourism domain. The idea here is to apply different strategies for dividing the set of goals (i.e. tourist recommendations based on previous plans executions by other tourists) for each user by using different utility recommendations systems. The GLASS architecture is shown in Figure 1. As can be seen, the architecture simply consists of a two-process loop: planning module and simulation+monitoring that share common information.

On the one hand, the input information is retrieved from different data sources. First, we need the user profile with the explicit interests of the user, the goals and preferences (such as points of interest he/she wants to visit), and temporal constraints. Second, we need to access a different set of databases that identify and categorize the points of interest (e.g. museums, restaurants, etc.), their timetable, and geographic sources to find out routes, distances and times between points. Currently, we use standard APIs, such as Google Places[4] and Directions[5] for this, but other open databases would be also valid, such as OpenStreetMap[6]. Third, there exists a snapshot of the environment or real-world scenario where the plan is executed. Since this world is highly dynamic and can change frequently (e.g. the opening hours of a museum has changed, or a restaurant is fully booked and the duration for having lunch is longer than expected), we get the new information as live events.

On the other hand, the planning module takes the user profile and the problem information to create a planning scenario in a PDDL format, as described in Section 3. We need to model the user preferences, constraints and the actions that the user can do, such as visit or move. The output of this is a plan, as a sequence of actions the user has to execute. As a proof of concept, in GLASS we actually simulate that execution rather than having a real execution that would require a true group of tourists equipped with sensoring information to their current geographic positions, pending and already satisfied goals, etc. This simulation process, more detailed in Section 4, takes the plan and creates a timeline structure to run a timed events based execution. It simulates and monitors the resulting states of the world, according to the changes in the plan, that is the effects that actions provoke and, probably, being also modified by the live events. This simulation is shown in a specially designed Graphical User Interface that shows what is happening at any time. If the expected state is different to the real state, i.e. a discrepancy has been discovered, because some live events prevent the remaining actions in the plan from being executed, a (re)planning module becomes necessary. The idea is to reuse the same planning module, thus closing the loop, with a new PDDL domain+problem specification to adapt the plan to the new emerging scenario.

3 PLANNING MODULE

The main goal of our system is to provide a personalized plan to a given tourist. This resulting plan has to reflect the preferences of the tourist according to his/her profile (demographic classification, the places visited by the user in former trips and the current visit preferences). Moreover, in order to build this plan, the duration of the activities to perform, the opening hours of the places to visit and the geographical distances between places (time to move from one place to another) needs also to be considered. Thus, solving this problem requires the use of a planning system capable of dealing with durative actions to represent the duration of visits; temporal constraints to express the opening hours of places and soft goals for the user preferences. Soft goals will be used to denote the preferable visits of the user, the non-mandatory goals that we wish to satisfy in order to generate a good plan that satisfies the user but that do not have to be achieved in order for the plan to be correct. In our case, the goal of visiting a recommended place according to the user profile (the list of potential places that the user can visit is returned by a Recommender System), is defined as a soft goal (more details in section 3.2). Among the few automated planners capable of handling temporal planning problems with preferences, we opted for OPTIC [1] because it handles the version 3.0 of the popular Planning Domain Definition Language (PDDL) [7], including soft goals.

[3] More info at `http://www.plg.inf.uc3m.es/~glass`
[4] More info at `https://developers.google.com/places/web-service`
[5] More info at `https://developers.google.com/maps/documentation/directions`
[6] More info at `http://wiki.openstreetmap.org/wiki/API`

All the information required by OPTIC to build the plan is compiled into a planning problem encoded in PDDL3.0 language, as described in the following sections.

3.1 Initial state

The initial state of a planning problem describes the state of the world when the plan starts its execution. The initial state must reflect the opening hours of the places to visit, the distances between them, the initial location of the user, etc. Some information is expressed with predicates and functions, while other information is represented by Timed Initial Literals (TILs). TILs, which were first introduced in PDDL2.2, are a very simple way of expressing a restricted form of exogenous events that become true or false at given time points [3].

The predicate `(be tourist ?l)` is used to represent the location of the user and the pair of TILs `(at 0 (active tourist))` and `(at` $t_{available}$ `(not (active tourist)))` determine the available time of the user for the tour, where $t_{available}$ is the difference between the time when the tour starts and finishes. The time indicated in the TILs is relative to the starting time of the plan; that is, $t_{available} = 540$ refers to 7pm if the plan starts at 10am. Another pair of TILs is used to define the time window in which the tourist prefers to have lunch. For example, if the preference is between 2pm and 4pm, the TILs are `(at 240 (time_for_eat tourist))` and `(at 360 (not (time_for_eat tourist)))`.

The duration of a particular visit `?v` for a tourist `?t` is defined through the numeric function `(visit_time ?v ?t)`. Assigning a value to a numeric function gives rise to a numeric-valued *fluent*; for example, `(= (visit_time Lonja tourist) 80)` (details about calculating the duration of the visit are shown in the following section). The list of available restaurants is given through the predicate `(free_table ?r)`; for example, `(free_table ricard_camarena)`. For each restaurant, we define the time slot in which it serve meals, which may depend on the type of restaurant, closing time of the kitchen or other factors. Both, places to visit and restaurants, have an opening hour and a closing hour that are specified by a TIL: `(at` t_{open} `(open` a`))` and `(at` t_{close} `(not (open` a`)))`, to indicate when the place/restaurant is not longer available. For example, `(at 0 (open Lonja)), (at 540 (not (open Lonja)))`.

The distance between two locations `?a` and `?b` is defined by the function `(moving_time ?a ?b)`, which returns the time in minutes needed to travel from `?a` to `?b` by using the travel mode preferred by the user. The time to move between two places is represented through a numeric fluent (e.g., `(= (moving_time caro_hotel Lonja) 9)`), where the value 9 is taken from Google Maps.

3.2 Goals and preferences

We handle two types of goals: *hard goals*, that represent the realization of an activity that the user has specified as mandatory (e.g., the final destination at which the user wants to finish up the tour `(be tourist caro_hotel)`); and *soft goals or preferences*, that represent the realization of a desirable but non-compulsory activity (e.g., visiting the Lonja `(preference v3 (visited tourist Lonja))`). Preferences are expressed in PDDL3.0 so we need to define how the satisfaction, or violation, of these preferences will affect the quality of a plan. The penalties for violation of preferences (costs) will be handled by the planner in the plan metric to optimize at the time of selecting the best tourist plan; i.e., the plan that satisfies the majority of the tourist preferences and thereby minimizes the penalties for violation.

The objective is to find a plan that achieves all the hard goals while minimizing a plan metric to maximize the preference satisfaction; that is, when a preference is not fulfilled, a penalty is added to the metric. Specifically, we define penalties for non-visited POIs and for travelling times.

The *penalty for non-visited places* is aimed to help the planner select the activities (tourist visits) with a higher priority for the user. Given a plan Π, this penalty is calculated as the ratio between the priority of the activities not included in Π and the priority of the whole set of activities recommended to the user (RA):

$$P_{non\_visited} = \frac{\sum_{a \in RA - \Pi} Pr^a}{\sum_{a \in RA} Pr^a}$$

For example, if the priority for visiting the Lonja is 290, and the sum of the priorities of all the visits is 2530, the penalty for not visiting the Lonja would be expressed in PDDL as: `( / (* 290 (is-violated v3)) 2530)`. The priority of the activities (Pr^a) is calculated by a hybrid Recommender System (RS) which returns a value between 0 and 300 according to the estimated degree of interest of the user in activity a. The value of Pr^a is also used by the RS to return a time interval that encompasses the minimum and maximum recommendable visit duration following a normal distribution $N\left(\mu_a, \sigma_a^2\right)$, where μ_a represents the average visit duration for a typical tourist [10]. Thus, the higher the value of Pr^a, the longer the visit duration.

The *penalty for movements* enforce a reduction in the time spent in travelling from one location to another, so that closer activities are visited consecutively. This penalty is calculated as the duration of all *move* actions of Π (Π_m):

$$P_{move} = \frac{\sum_{a \in \Pi_m} dur(a)}{dur(\Pi)}$$

The function `(total_moving_time tourist)` accumulates the time spent in transportation actions, so this penalty would be defined in PDDL as: `( / (total_moving_time tourist) 540)`. The plan metric to be minimized by the planner is expressed as the sum of both penalties: $P_{total} = P_{non\_visited} + P_{move}$.

3.3 Actions

We define three actions in the tourism domain. The action to **move** from one location to another is defined in Figure 2. It takes as parameters the user `?per`, the initial location `?from` and the destination `?to`. The duration of the action is set to the estimated/actual time to go from `?from` to `?to`, which is stored in the database. The preconditions for this action to be applicable are: (1) the user is at location `?from` and (2) the time window for the available time of the user is active during the whole execution of the action. The effects of the action assert that (1) the user is not longer at the initial location, (2) the user is at the new location at the end of the action and (3)

```
(:durative-action move
  :parameters (?per - person ?from - location
               ?to - location)
  :duration (= ?duration (moving_time ?to ?from) )
  :condition
    (and
      (at start (be ?per ?from))
      (over all (active ?per)))
  :effect
    (and
      (at start (not (be ?per ?from)))
      (at start (walking ?per))
      (at end (be ?per ?to))
      (at end (not (walking ?per)))
      (at end (increase (total_moving_time ?per)
                  (moving_time ?from ?to)))))
```

Figure 2. Action `move` of the tourism domain

the time spent in move actions is modified according to the movement duration. In order to indicate the position of the user during the execution of the action, a `walking` predicate is asserted at the start of the action and deleted at the end of the action. In this paper, we only consider *walking* as the move action; however, more transportation modes according to the user's preferences can be included; e.g., cycling, driving, and public transport, as in [10] .

The action to **visit** a place is defined in Figure 3, whose parameters are the place to visit `?mon` and the user `?per`. The duration of the action is defined by the function `(visit_time ?mon ?per)`. The conditions for this action to be applicable are: (1) the user is at `?mon` during the whole execution of the action; (2) `?mon` is open during the whole execution of the action and (3) the time window for the available time of the user is active. The effects of the action assert that the place is visited.

The action to perform the activity of **eat** is defined in Figure 4, whose parameters are the user `?pers` and the restaurant `?loc`. The duration of the action is defined by the function `(time_for_eat ?pers)` and specified by the user. To apply this action, the following conditions must hold: (1) the user is at `?loc` during the whole execution of the action; (2) `?loc` is open during the whole execution of the action; (3) the restaurant has a free table and (4) both the time window for the time to have lunch defined by the user and the available time are active. The effects of the action assert that the user has had lunch.

4 SIMULATOR

The objective of the simulator is to execute the plan and monitor that everything works as expected. To accomplish this, we first need to create the structures to perform the simulation. We use a timed event simulation, where events occur at particular times through a timeline, possibly provoking changes in the world state. During the plan monitoring, we check the predicates and functions and we visually show the plan trace in a specially designed GUI. In case a failure that prevents an action of the plan from being executed is found during the plan simulation, we activate a replanning mechanism that requires a knowledge-based reformulation of the new planning scenario. Next, we describe these tasks in more detail.

```
(:durative-action visit
  :parameters (?per - person ?mon - monument)
  :duration (= ?duration (visit_time ?mon ?per))
  :condition
    (and
      (at start (be ?per ?mon))
      (over all (be ?per ?mon))
      (over all (active ?per))
      (over all (open ?mon))
  :effect
    (and
      (at end (visited ?per ?mon))))
```

Figure 3. Action `visit` of the tourism domain

```
(:durative-action eat
  :parameters (?pers - person ?loc - restaurant)
  :duration  (= ?duration (eat_time ?pers ?loc))
  :condition
    (and
      (at start (free_table ?loc))
      (at start (be ?pers ?loc))
      (over all (be ?pers ?loc))
      (over all (active ?pers))
      (over all (open ?loc))
      (over all (time_for_eat ?pers)))
  :effect
    (and
      (at end (eaten ?pers))))
```

Figure 4. Action `eat` of the tourism domain

4.1 Timed event simulation: the timeline

A timeline is a simple structure that contains a collection of unique timed events in chronological order that represents a sequence of world states and that need to be monitored. The timeline is generated with the actions of the plan, the problem information and the live events, as depicted in Figure 1. A timed event is an event that happens at time t and contains the following information: (1) the *start*, *over all* or *end* conditions to be checked at t; (2) the *start* or *end* effects to be applied at t; (3) TILs that represent exogenous events but that are defined as part of the problem information, so they are known at the time of the plan simulation; and (4) *live events*, that dynamically appear during the executing/monitoring process and so they are unknown a priori. This way, a timeline encapsulates the information about the plan (irrespective of it is a sequential or parallel one), TILs and live events[7], and the corresponding world states. The time scale of the timeline will depend on the granularity of the plan and the periodic steps we want to use for monitoring the timed events. In our implementation, live events can be manually supplied or they can be retrieved from a datasource that keeps information about the real world.

Given a plan with two actions (`move` and `visit`), of duration 20 and 60, respectively, and a live event that indicates the

[7] The information about the plan, problem information and live events is modeled in PDDL format. We used PDDL4J (`https://github.com/pellierd/pddl4j`), an open source library that facilitates the development of JAVA tools for automated planning based on the PDDL language

```
0.00 move person1 hotel museum [20.00]
20.01 visit person1 museum [60.00]
90.00 live event (not (open museum1))
```

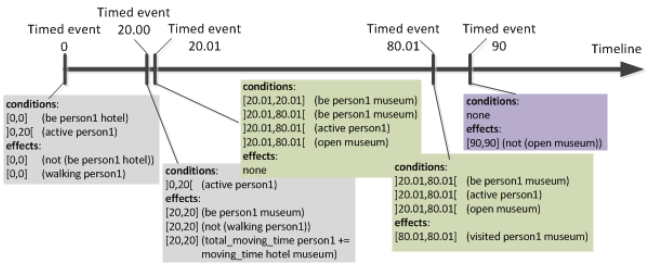

Figure 5. An example of a timeline with five timed events. Note the closed interval for the *at start/end* conditions and effects, and the open interval for the *over all* conditions

museum is closed (not open) at time 90, the resulting timeline is shown in Figure 5.

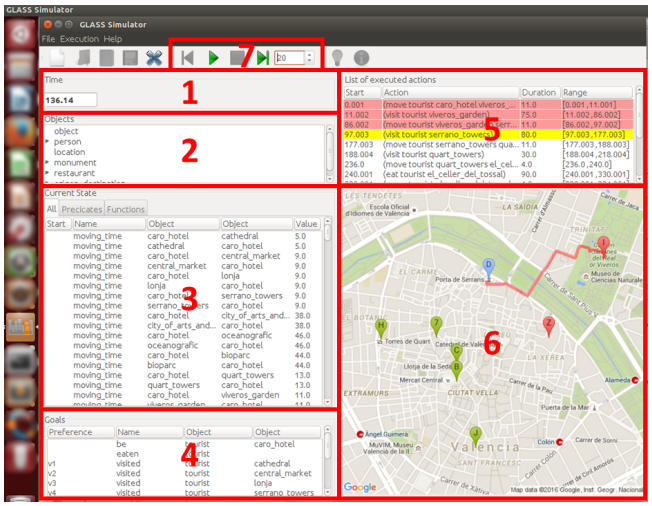

Figure 6. Simulator graphical user interface

4.2 Plan execution simulation

The simulation of the plan execution requires to set the size of the execution step that will be applied along the timeline explained in section 4.1. We can set the step size to the time granularity of the planner or choose a larger size. The smaller the size of the execution step, the more frequently access to external databases (Google APIs) to acquire new information and update the real-world state. Thus, the execution step size specified by the user determines the update frequency of the internal state of the simulator with respect to the real-world state. If changes frequently occur in the domain, a small execution step will result in a more reliable simulation with a proactive behavior. The simulation state is also updated at each timed event, checking the conditions of the actions in the state and applying the effects of the event. Additionally, the simulator also interacts with the real-world through live events, which may in turn modify or create new timed events in the timeline.

The simulation of the plan execution starts at time zero, with an initial state equal to the real-world state, and the simulator advances through the timeline in every execution step (see Figure 5). The simulator checks that conditions are satisfied, the current state matches the expected state, and then updates the current state accordingly — this whole process is visually shown in our GUI, described below. More specifically, every execution step involves two main tasks:

1. processing the live events for changes and update the respective timed events
2. for every unprocessed timed event within the current step: (1) update the simulation state with the TILs and effects of the live events; (2) check the conditions of the timed event to find differences between the current state and the expected state; and (3) update the state with the effects of the actions.

If a difference between the current state and the expected state is found and this difference leads to a situation where the plan is no longer executable, then a failure has been detected. In such a case, the GUI informs the user about the cause of the failure: the action that has failed and the conditions that have been violated. For instance, in the example of Figure 5, let us suppose there is a live event at time 60 that indicates the museum will no longer be open from 60 onwards. In this case, the *overall* condition `]20.01,80.01[ (open museum)` is violated, which means the `visit` action cannot be successfully executed. Then, we need to invoke the replanning module as described in Section 4.4

4.3 Graphical User Interface

The graphical user interface (GUI) has been designed to provide information about the internal state of the simulator during the whole plan execution simulation and provides mechanisms to control the next step of the simulation. The GUI is specifically designed to offer a smart-city orientation. It includes six distinguishable GUI parts:

1. Figure 6-section 1 shows the current simulation time
2. Problem objects (Figure 6-section 2): it displays the planning problem objects along with their types. This static information will not change over the simulation process.
3. Current state (Figure 6-section 3): This graphical section contains the PDDL description of the current state, which can change after an action starts or ends, when a live event arrives or when a user introduces a manual change (TILs). Propositions and numeric fluents of the current state can be separately consulted in two different tabs.
4. Figure 6-section 4 shows the problem goals. In later refinements, we intend to show the goals that are expectedly to be achieved with the plan under execution.
5. Figure 6-section 5 shows the dynamic list of plan actions, their start time, the objects involved in the action execution and the action duration. In addition, actions are shown with a representative colour: actions currently in execution are shadowed in yellow, past or already executed actions in red, and future actions in white.
6. Representative map (Figure 6-section 6): The map depicts with location icons the relevant places involved in the plan

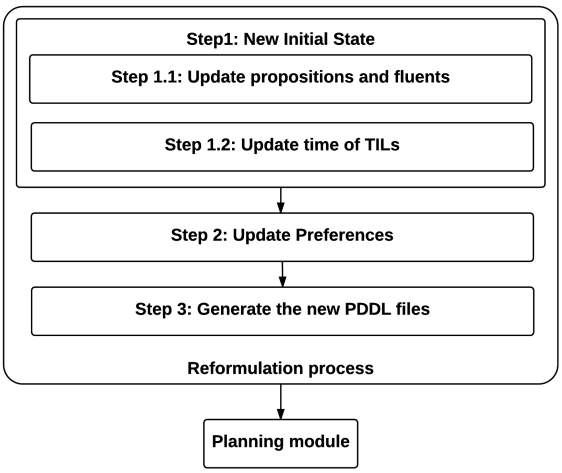

Figure 7. Reformulation steps

(places to visit, restaurant, hotel). These location icons change their colour when the corresponding action is executed. The map also displays distances between locations.

7. Simulation control buttons (Figure 6-section 7): In the middle of the top menu, the interface displays four buttons to run the simulator step by step (the step size is defined by the user), continue with the simulation, stop the simulation and reset the simulation.

4.4 Reformulating the planning problem

Figure 7 shows the steps of the reformulation procedure.

Step 1: Create the New Initial State. The initial state will comprise the information known by the simulator at the time of creating the new problem. This includes the information of the current simulation state plus the information about future TILs; that is currently known information about some future events. Thereby, the occurrence time of the future TILs must also be updated.

Step 1.1: Update propositions and fluents. This step refers to the update of the current state. The propositions and fluents after the failure are retrieved from the current world state. However, this might not be an accurate state since we do not have sensing actions that provide us with a precise picture of the real world. This may be particularly problematic when an *overall* or an *at end* condition of an action is violated and the action has *at end* effects. Let us take as an example the action `(move tourist caro_hotel viveros_garden)`, with an *at start* effect `(at start (not (be tourist caro_hotel)))`, an *at end* effect `(at end (be tourist viveros_garden))`, and an *overall* condition `(over all (active tourist))`. Due to a failure that resulted from a live event which violated the previously mentioned *overall* condition, the tourist is neither in `caro_hotel` because the *at start* effect were already executed, nor in `viveros_garden` because the *at end* effects were not yet executed due to the failure. For simplicity, and because of the lack of sensing actions in our current implementation, when a failure happens due to an *overall* or an *at end* condition violation, we will calculate the new initial state by simply rolling back the *at start* effects of the failing action (if any).

Step 1.2: Update the time of TILs. When the new problem is reformulated, we invoke the OPTIC planner, which resets the time of execution and generates a plan starting from time equal to zero. Consequently, we need to update the occurrence time of the TILs to the result of its original time minus the failure time. Let us assume that a failure occurred at time 100, and that we have the TIL planned at time 235 `(at 235 (not (open la_paella)))`, meaning that the restaurant *la paella* will close at 235. Therefore, in the new initial state formulation, its time will be 135 (235 minus 100); and it will be updated to `(at 135 (not (open la_paella)))`.

Step 2: Update preferences. When a failure occurs, we come across a situation where we can distinguish two types of preferences or soft goals:

1. Goals that have already been achieved at the time of the failure by the actions that have been successfully executed before the failure. These preference goals along with their penalties will not be included in the new reformulated problem.
2. Goals that have not been satisfied, and which can in turn be divided into two sets:
   (a) The problem goals that were not included in the original plan;
   (b) The problem goals included in the original plan that have not been satisfied yet due to the failure.

- For the set of goals in (a), the penalties are kept intact as they were originally defined in the problem file.
- For the set of goals in (b), we want to keep the plan stability metric [4] similar to the concept of minimal perturbation [11], which is why we increase the penalties of these goals in the new reformulated problem. Particularly, we opt for assigning a relatively high priority to these pending goals (twice as much as the maximum penalty among all goals), in order to potentially enforce these goals in the new plan. We have thus opted for applying a *stability strategy* that gives more priority to goals that were already included in the original plan than goals that were not. Other strategies such as keeping a higher level of stability with respect to the failed prior plan can also be adopted. In the case of a tourism domain, we think that maintaining the original agenda of the tourist as far as possible is more advisable.

  For example, let us consider that the preference `(preference v2 (sometime (visited tourist central_market)))` is one of the soft goals in the set (b); this preference indicates that sometime during the execution of the plan the tourist wishes to visit the `central_market`. Assuming that the penalty of this preference in the original problem file was 270, and that the highest among all preferences was 300, the new penalty for preference `v2` will be 600.

Finally, two points are worth mentioning. First, we learn the soft goals that the planner decided to pursue in the original plan by simply executing the plan without any live events. Second, the pending hard goals of the original problem are kept as hard goals in the new problem file.

Step 3: Generate the new PDDL files. The last step of the reformulation process consists in generating the new PDDL files. In principle, the domain file remains unchanged, unless we wish to necessarily include some particular action in the new plan. In this case, we would need to encode a *dummy effect* that triggers the corresponding action. Otherwise, only the problem file is generated taking into consideration the modifications discussed in step 1 and step 2.

In future implementations of the simulator, we will test a Constraint Programming approach [6] for reformulating the planning problem, as in [15], and compare the performance when relying on a scheduler rather than a planner.

5 CASES OF STUDY

The aim of this section is to show the behaviour of our simulation system with a representative example. We have a tourist who wishes to make a one-day tour in the city of Valencia. Initially, the system retrieves a set of recommended places according to the user profile (table 1, column 1) and a set of restaurants. The list of recommended places is calculated by a Recommender System through the user profile. This list of places comes along with a recommendation value (Table 1, column RV) according to the interest degree of the user in the particular place. This value will be used by the planning module to obtain a plan that fits the user's likes.

The tour (plan) for the user calculated by the planner is shown in the left snapshot of Figure 8. The visits included in the plan are marked with a red location icon in the snapshot. The tour starts from the origin location of the tourist, i.e., the hotel in which the user is staying at (green location icon), and includes six visits to monuments (red icons) and one stop at a restaurant (orange icon).

The simulator starts the plan execution simulation with the information provided above. Let us assume that at time 1:55 pm, a live event is received, `(at 235 (not (free_table el_celler_del_tossal)))`, indicating that the restaurant chosen by the planner, *el celler del tossal*, is completely full and has no available table. At the time the live event arrives, the tourist has already visited the first three monuments (1. Viveros garden; 2. Serrano towers; 3. Quart towers), and he is currently at the location of the restaurant *el celler del tossal*. When the user learns the restaurant is fully booked, the simulator detects a failure because the action `(eat tourist el_celler_del_tossal)` is not executable. Then, the simulator reformulates a new planning problem in order to obtain a plan that solves the failure:

1. *Initial state*: the current location of the tourist is the point at which the previous plan failed; i.e., the restaurant *el celler del tossal*. Since the new initial state is initialized to time zero ($t_{ini} = 0$), the simulator updates the time of the TILs in the current state, namely, the opening and closing time of places, the time slot for having lunch and the TIL `(at` $t_{available}$ `(not (active tourist)))`, where $t_{available}$ is set to the new time the tourist must get back to the hotel from $t_{ini} = 0$. Additionally, the fluent `(total_moving_time tourist)` is updated with the total time the tourist has spent in moves around the city.
2. *Goals*: the places that have already been visited (the first three monuments) are removed from the goal list. The new set of goals includes two lists: (a) the *pending goals* of the failed plan; that is, have lunch (4. have lunch) and the three remaining monuments that have not been yet visited by the user (5. Lonja, 6. Central market, 7. Town Hall); plus (b) the goals of the original problem goals that were not included in the first plan.

Regarding penalties of the goals, the list of goals in (b) are included in the new planning problem with their original recommended values (see the non-bold values RV' in column 2 of Table 1). As for the pending goals of (a), the penalty of these goals is increased with respect to their penalty in the first plan (see the bold values RV' in column 2 of Table 1 for the three pending monuments) accordingly to the stability concept explained in section 4.4.

The simulator invokes OPTIC and obtains a new plan, displayed in the middle snapshot of Figure 8. A few things must be noted in this new plan:

1. OPTIC suggests a new restaurant (orange icon labeled with number 4) which is rather far away from the prior restaurant. The reason is that we have only included in the planning problem the 10-top restaurants in Valencia suggested by *Trip Advisor*, and the closest one to the prior restaurant is the one shown in the second map.
2. The places included in the new plan are marked with green location icons as well as the paths between places. We can observe that the new plan maintains the visit to Town Hall (now represented with the green icon numbered as 5) and to the Lonja (now indicated with the green icon with number 6). However, the visit to Central Market has been discarded in this new plan. This is likely due to the longer distance to the new restaurant.

The simulation continues. Let us assume that when the tourist is visiting the Town Hall, a new live event announcing the building closes before the scheduled closing time (`(at 140 (not (open town_hall)))`) is received, the current time being equal to 140 . A new failure is detected in the middle of the execution of `(visit tourist town_hall)` due to a violation of an *overall* condition. As we explained in section 4.4, the simulator applies a rollback process to obtain the current state before the last visit action but preserving the simulation time. Then, in the new reformulated problem, the user is located at the Town Hall, he has not visited the Town Hall and the live event causes the proposition `(open town_hall)` to be removed from the initial state. Note that the goal `(visited tourist town_hall)` will be included as a goal in the new problem to maintain the plan stability but since the Town Hall is no longer open for visits, the planner will not include this goal in the new plan. The penalties of the goals for this third problem are shown in column 3 of Table 1.

The third plan (right snapshot of Figure 8), shown in light blue colour, retrieves the visit to the Central market that was eliminated from the second plan. The new plan suggests visiting La Lonja (5. la Lonja) and then the Central market (6. Central market). Finally, the user returns to the hotel.

6 CONCLUSIONS AND FURTHER WORK

We have presented a context-aware smart tourism simulator that keeps track of the execution of a tourist agenda. The sim-

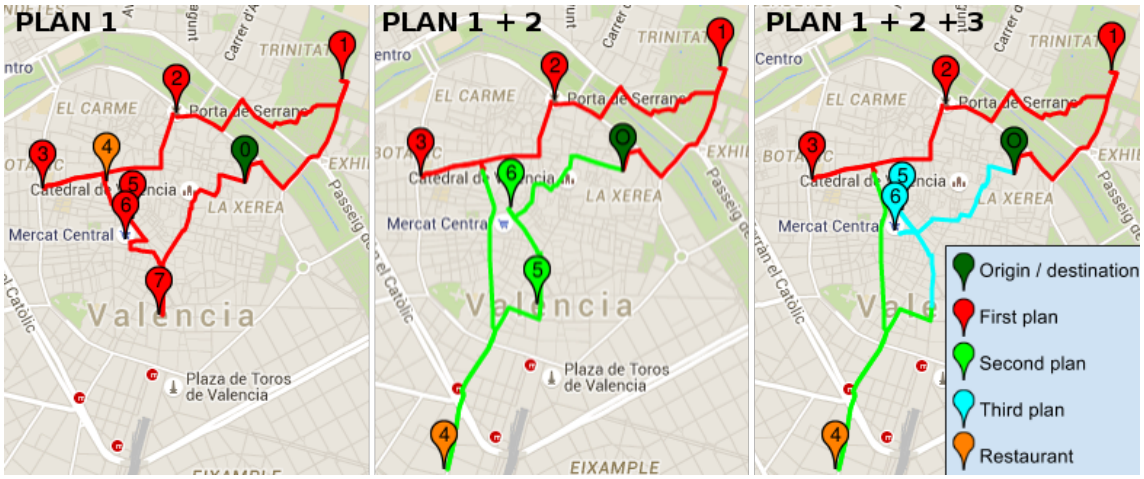

Figure 8. The three simulated plans. Icons in PLAN1: (0) Caro hotel, (1)Viveros Garden, (2) Serrano towers, (3) Quart towers, (4) El Celler del Tossal (RESTAURANT), (5)Lonja, (6) Central market, (7) Town hall. Icons in PLAN2: 1,2,3 are the same as PLAN1, (4) the Pederniz (RESTAURANT), (5) Town hall, (6)Lonja. Icons in PLAN3: 1,2,3,4 are the same as PLAN2, (5) Lonja, (6) Central market

PLACES	RV	RV'	RV"
Cathedral	280	280	280
Central market	270	**600**	**600**
Lonja	290	**600**	**600**
Serrano towers	250	—	—
City of arts and sciences	280	280	280
Oceanografic	300	300	300
Bioparc	210	210	210
Quart towers	200	—	—
Viveros garden	250	—	—
Town hall	200	**600**	**600**

Table 1. Recommended places

ulator periodically updates its internal state with real-world information and receives sensible environmental changes in the form of live events. Events are processed in the context of the plan and in case of failure, a new planning problem is formulated. This involves creating the new initial state and updating the time of timed events. In the case of study, we have shown how the user can track the plan execution through a GUI that automatically displays the plan under execution.

As for future work, we intend to endow the system with a pro-active behaviour, analyzing the incoming of live events that entail a future failure in the plan.

REFERENCES

[1] J. Benton, A. J. Coles, and A.I. Coles, 'Temporal planning with preferences and time-dependent continuous costs', in *ICAPS*, (2012).

[2] D. Buhalis and A. Amaranggana, 'Smart tourism destinations enhancing tourism experience through personalisation of services', in *Proc. Int. Confernece on Information and Communication Technologies in Tourism*, pp. 553–564, (2013).

[3] S. Edelkamp and J. Hoffmann, 'PDDL2.2: The language for the classical part of the 4th International Planning Competition', *IPC 04*, (2004).

[4] M. Fox, A. Gerevini, D. Long, and I. Serina, 'Plan stability: Replanning versus plan repair', in *ICAPS*, volume 6, pp. 212–221, (2006).

[5] I. Garcia, L. Sebastia, E. Onaindia, and C. Guzman, '*E-Tourism*: a tourist recommendation and planning application', *International Journal on Artificial Intelligence Tools*, **18**(5), 717–738, (2009).

[6] A. Garrido, M. Arangu, and Eva. Onaindia, 'A constraint programming formulation for planning: from plan scheduling to plan generation', *Journal of Scheduling*, **12**(3), 227–256, (2009).

[7] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, 'Deterministic planning in the 5th International Planning Competition: PDDL3 and experimental evaluation of the planners', *Artificial Intelligence*, **173**(5-6), 619–668, (2009).

[8] U. Gretzel, M. Sigala, Z. Xiang, and C. Koo, 'Smart tourism: foundations and developments', *Electronic Markets*, **25**(3), 179–188, (2015).

[9] R. Howey, D. Long, and M. Fox, 'VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL', in *16th IEEE ICTAI*, pp. 294–301, (2004).

[10] J. Ibañez, L. Sebastia, and E. Onaindia, 'Planning tourist agendas for different travel styles', in *ECAI*, (2016).

[11] S. Kambhampati, 'Mapping and retrieval during plan reuse: A validation structure based approach.', in *AAAI*, pp. 170–175, (1990).

[12] F. Martínez-Santiago, F. J. Ariza-López, A. Montejo-Ráez, and A. U. López, 'Geoasis: A knowledge-based geo-referenced tourist assistant', *Expert Syst. Appl.*, **39**(14), 11737–11745, (2012).

[13] I. Mínguez, D. Berrueta, and L. Polo, 'Cruzar: An application of semantic matchmaking to e-tourism', *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications. Information Science Reference*, (2009).

[14] B. Neuhofer, D. Buhalis, and A. Ladkin, 'Smart technologies for personalized experiences: a case study in the hospitality domain', *Electronic Markets*, **25**(3), 243–254, (2015).

[15] I. Refanidis, C. Emmanouilidis, I. Sakellariou, A. Alexiadis, R.-A. Koutsiamanis, K. Agnantis, A. Tasidou, F. Kokkoras, and P. S. Efraimidis, 'myVisitPlanner ^{gr}: Personalized Itinerary Planning System for Tourism', in *SETN*, pp. 615–629. Springer, (2014).

[16] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, 'The city trip planner: An expert system for tourists', *Expert Syst. Appl.*, **38**(6), 6540–6546, (2011).