ITC 4/46
Journal of Information Technology and Control
Vol. 46 / No. 4 / 2017
pp. 484-498
DOI 10.5755/j01.itc.46.4.17331

Cloud Services for On-Demand Vehicles Management

Received 2016/12/24 | Accepted after revision 2017/10/16

http://dx.doi.org/10.5755/j01.itc.46.4.17331

Cloud Services for On-Demand Vehicles Management

Andrea Fornaia, Christian Napoli, Emiliano Tramontana

Department of Mathematics and Informatics, University of Catania, Viale Andrea Doria 6, 95125, Catania - Italy
e-mail: {fornaia, napoli, tramontana}@dmi.unict.it

Corresponding author: tramontana@dmi.unict.it

Smart cities providing connectivity to users and other advanced services can be leveraged to improve public transport services. This paper proposes a solution that lets citizens request a public vehicle to perform additional stops off the main route, hence achieving a customisation of the transport operator services to better assist users. A cloud infrastructure and a proper distributed architecture have been designed to assess whether user requests can be accepted. The proposed software solution considers viable the requests that can fit to available secondary routes, while also satisfying other user demands that have been previously accepted. Then, drivers will be alerted in advance in order to adapt their route.

KEYWORDS: Cloud computing, workflow, monitoring, intelligent assistive services, smart cities.

1. Introduction

Smart Cities and services providing citizens with means to better organise daily life, and Smart Mobility are hot topics for future city developments [2]. An important field where sensors and the internet-of-things will be used is that of monitoring the physical world, especially critical infrastructures such as bridges, airports, ports, motorways, etc. Additionally, sensors will let cars and their environment exchange data [1,5].

Several novel services are under test, including autonomous vehicles, infrastructures that analyse car flows and seek to minimise queues at intersections, etc. Moreover, users could be provided with an on-demand transport service for them or their goods. This paper analyses the scenario of public transport and provides a solution to organise the flow of publicly accessed vehicles according to user requests, i.e. *on-demand*. Thanks to timely processed user requests, the transport operators can improve their services by using more vehicles on highly requested routes, and avoid over-provisioning and wastes. The ubiquitous communication infrastructure would be used to

connect users, drivers, and transport operators; then gathered data allow each of them to properly react to some changes, e.g. newly programmed bus stops, fully booked vehicles, as well as traffic jams. Of course, the devised solution can be used in combination with the future autonomous unmanned vehicles. While for the time being alerts for adapting the routes would be passed on to drivers, later they would be sent to the vehicles themselves.

In the given scenario, we have organised as a *workflow* the interactions between services and users requesting a vehicle to stop on some location and for a selected destination. A *workflow* is a standard flow of execution for the activities related to the offered service [9]. The proposed workflow is executed on the server-side and comprises several activities such as: assessing whether the currently available running vehicles can stop on requested location, adapting the route for a vehicle, and alerting the requesting users, driver and passengers for the adaptation. The server-side will have to ensure that route changes will take place only when disruption is minimal, i.e. the other planned stops are satisfied as well as the timing constraints agreed with the users. In a smart environment, such a workflow is one among many that is executed on a cloud computing resource and a processing engine [6].

Cloud-computing provides transparent access to services, hardware and data. Thanks to the possibility to add computing resources on-demand a high-level of *availability* and performances are ensured. However, a cloud-based infrastructure can bring about delays for the virtual machine (VM) to be started, services to be allocated, etc. Primary goals for enterprises handling the said transport scenario include: (i) having the minimum amount of disruption to paths of running vehicles, (ii) providing scalability to handle a variable number of requests with high availability, and (iii) minimise the potential delays of the processing infrastructure.

This paper is structured as follows. Section 2 details the transport scenario that we have tackled. Section 3 describes the proposed software architecture for workflow management. Section 4 details our proposed predictor for transport requests. Section 5 discusses the related work, and Section 6 draws our conclusions.

2. Smart City Transport Scenarios

The proposed enhanced bus service lets users reserve seats and then possibly adapts the a priori established course. Whether the user requested route changes would be accepted depends on the agreements that have been taken between users and the transport operator. Users request an additional stop and provide: the desired pick-up and destination location, and the useful timeframe for pick-up. The services on a cloud computing system will then assess and possibly acknowledge the request, indicating the best rendezvous point for the user to catch the bus.

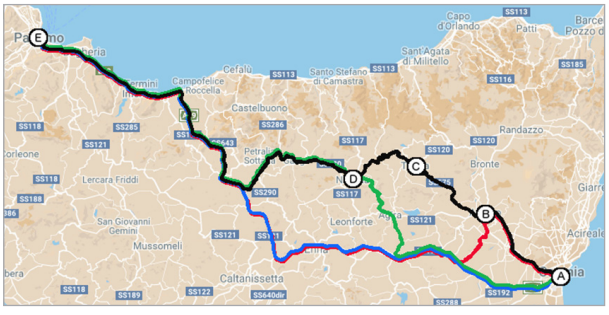
In our model, a *rendezvous point* is a possible *bus stop*, one among many known by the transport operator, that is activated on-demand, hence avoiding the need to serve bus stops when not requested. As a result, vehicles travel paths can be optimised, and energy consumption and CO2 emissions reduced.

2.1. The First Scenario

Figure 1 shows four possible routes connecting the Catania and Palermo cities of the Italian region of Sicily. By gathering user requests, the transport operator can determine in advance which of the five potential rendezvous points depicted will be served, hence planning the route to follow. User requests are gathered before the bus starts going and during its journey,

Figure 1

Different routes connecting Catania (A) to Palermo (E) in Sicily. According to user requests, the fastest route (in blue) could be bypassed to serve secondary on-demand rendezvous points, such as Adrano (B), Troina (C) and Nicosia (D). Suppose that only B has been requested, then the red route will be chosen; in case D has been requested, the green one instead; if all B, C and D have been added, the black route will be chosen, considerably diverging from the original blue one



then a server-side component notifies the bus drivers when it would be possible for the bus to change the route in order to serve requested rendezvous points. In the depicted example, the transport operator provides as a daily basis a connection between two of the main Sicilian cities, located on the opposite coasts of the island. Along the way there are possible intermediate stops allowing the bus to serve different minor locations just following different routes. Since the number of requests for the three depicted inland locations (Adrano(B), Troina(C) and Nicosia(D)) could be not enough to justify a separate daily served route for each, by knowing in advance which of the secondary stops have been booked, then the main bus route from Catania to Palermo (depicted in blue) can be adapted.

To timely sense actual customer presence, the transport operator provides an app that allows users to request either a scheduled bus stop (such as Catania) or asking for the activation of an on-demand bus stop among the ones available for the specific adaptable route (such as Adrano, Troina and Nicosia). In the first case, choosing a scheduled stop will not cause any changes in the ongoing travel schedule; in the second one, before activating an on-demand stop the system assesses whether updating the route schedule is both worthy and possible for the transportation company.

Figure 2 shows two screen-shots of an app suited to access the transport service. After the selection of the Catania – Palermo route the server side will provide a complete list of the possible stops. In this example, the bus has already left Catania (the state for the scheduled stop in Catania is set to *served*), and its progress is such that the Adrano stop is *no more available*, that is, the bus driver cannot change the travel route (from the blue to the red one in Figure 1) to serve that stop without violating the service agreement (e.g. time schedules). The Nicosia stop is still *available*, since we are still in time to modify the route from the scheduled one (in blue) to the one passing from it (in green). The system has already evaluated the possible adaptation providing the user with a ticket that could be charged with a fee due to the on-demand activation of a non-scheduled stop.

Once the user has requested the Nicosia *on-demand* stop, the cloud infrastructure will send the *route schedule update* to the bus driver on a devised app (see Figure 3) asking them whether to *accept* or *refuse* the change, i.e. diverging from the blue route to the green one.

Figure 2

Using a smartphone app, the transportation company can gather user requests, and by determining in advance which of the optional stops need to be served, vehicles travel cycles can be adjusted and optimised

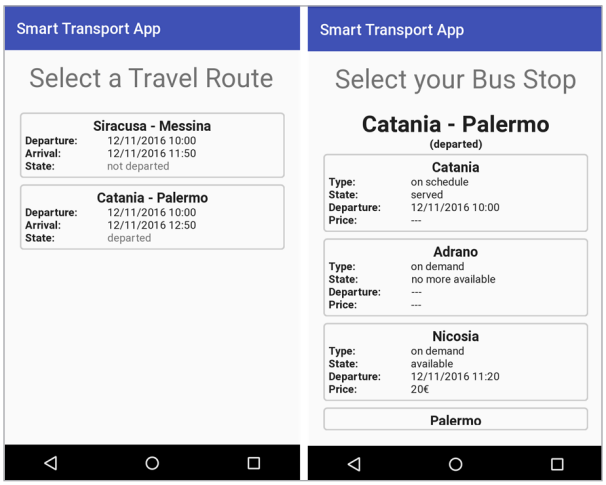
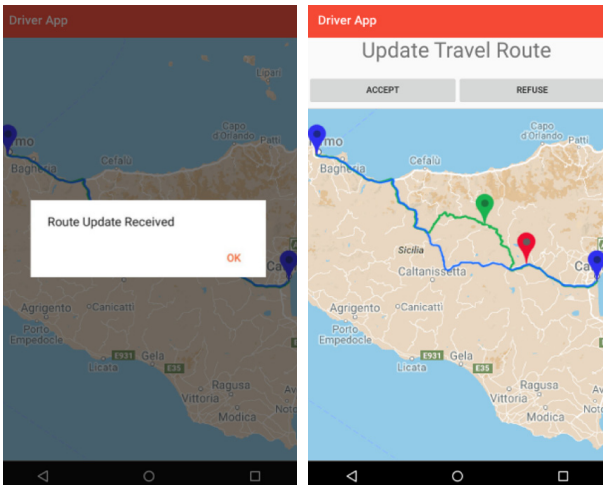


Figure 3

For the transport company to notify drivers on route changes, they are given an app. After receiving a route update request from the cloud infrastructure (left) the app will show the driver the route change (right) from the blue to the green one. Red marker shows the bus position on the map



2.2. The Second Scenario

In the previous example, the described enhanced bus service was used to serve cities across the Sicily island, thus considering long distances to be covered by a scheduled route. However, the on-demand bus ser-

vice is also suitable to address transportations needs over a smaller scale.

Urban areas are usually well connected and frequently served by different interconnected transport services. However, this may not be the case in *non-urban* areas, where longer distances and typically unpredictable request rates over a widespread number of possible bus stops need to be considered. In these cases, the proposed flexible transportation service can be used to optimise the number of vehicles to be employed.

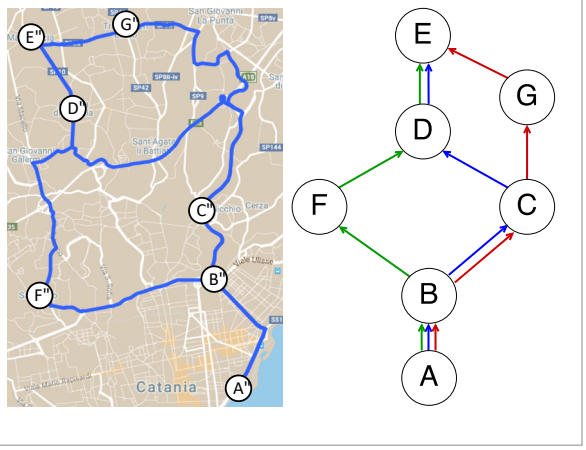
Figure 4 shows three possible routes connecting Catania (A) to the Mascalucia (E) village in its non-urban area. The fastest route to reach the destination is depicted in blue (see right side of Figure 4), so serving the Gioeni (B), Canalicchio (C), and Gravina (D) stops along the way. In this scenario, serving the B stop is considered mandatory for the transportation company, thus any possible route variation needs to serve it.

Additionally, two on-demand stops can be requested, such as Nullo (F) and Tremestieri (G). The cloud infrastructure will show these two stops only when actually available, i.e. when the route schedule can still be changed: F will be shown only if B has not been served yet, thus it is still possible to leave the blue path moving to the green one; G will be available only if C has not been served yet, thus it is still possible to move to the red path.

In both cases, the route change can cause some stops to be skipped, because belonging to different alternative routes, i.e. F excludes C and G excludes D. The cloud infrastructure determines whether to show possible on-demand stops by evaluating both the possibility and advantage for the transportation company to move from one route to the other.

Figure 4

Left side shows different routes connecting Catania (A) to Mascalucia (E), a village in the Catania province (non-urban area). Right side shows the corresponding graph representation: depending on user requests, the fastest route (in blue), which already serves the Gioeni (B), Canalicchio (C), and Gravina (D) stops, could be adapted to serve secondary on-demand stops, such as Nullo (F) or Tremestieri (G). In the former case, the fastest blue route will be changed to follow the secondary green one. In the latter case, the red route will be followed



2.3. Route Planning Workflow

Choosing the right rendezvous points to satisfy user requests is an important concern: the server-side planning component has to properly manage the trade-off between providing the solution nearest to the requesting user and modifying the established bus schedule (even on-the-fly). In Section 4 we will see how this schedule management can be assisted by an intelligent system able to predict future transport service requests in a given location.

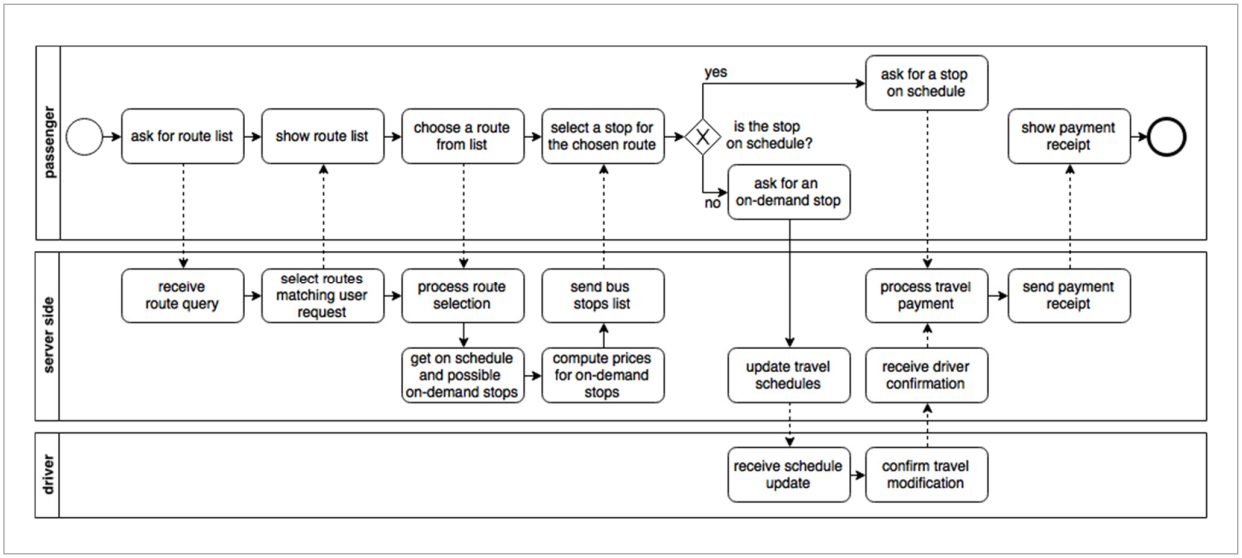
The proposed solution goes beyond the urban transport use case; i.e. other smart facilities can take advantage of the said data. The number of citizens wishing to use a transport service would be known for many locations over time. Then, a possible strategy, which can benefit mobile service providers, such as telco operators, is to take advantage of data representing user locations to estimate the mobile cell occupancy and operate needed counteractions.

Figure 5 shows a simplified version of the workflow used to manage bus stop requests and the consequent travel schedule updating described above, dubbed *on-demand adapted route*. Of course, this is not the only workflow that a transport operator can leverage to provide an innovative service. For example, in a previous contribution [8] a different workflow description was shown, considering a scenario where instead of selecting a bus stop from a list either scheduled or on-demand, citizens were supposed to actively request vehicles to pass by a suggested place, in a given time-frame, as needed, asking the system to find the best suited solution to serve user request.

In both cases, on the server-side, once a user request has been received, a workflow is available to execute and regulate several services (activities), each corre-

Figure 5

An example of workflow called *on-demand adapted route* describing the activities and the interactions between the passengers reserving a bus stop, the drivers that need to update their travel schedules and the server side. While the passenger and the driver side will be held by two different smartphone apps, the more complex server side will take the advantages of using a distributed cloud infrastructure for the execution of the related workflow activities



sponding to a step that has to be performed. In our reference model for the software system assisting such steps we will have one or more client applications enabling the user to submit a request, and wait for a reply. Hence, e.g. the *ask for route list* step is performed using a dedicated smartphone app that lets the user choose the desired schedule or on-demand stop for a specific travel route.

Services on the server side are processes running, or started, according to the indication given by the workflow description, hence e.g. *receive route query* is the first step of an ad-hoc workflow, and is a process listening for incoming requests, residing in-side a persistent web service, and *select routes matching user request* is a process started as the second step of the workflow once the previous step has been performed, etc. Since each service needed for a workflow completion in general can have its own preconditions, i.e. data and processing requirements, then each service should be handled ad-hoc to provide the proper quality of service. Let us suppose that *compute prices for on-demand stops* is a CPU-bound process whose execution time has to be guaranteed, because, e.g. it could involve several scheduling decisions and transport combinations, starting from the temporal and location request given by users, aiming to provide the best solution for the users while reducing schedule changes. Instead, another service could simply provide immutable stored documents. Then, handling requests that trigger service execution requires the provisioning of ad-hoc computing resources to ensure the quality of service.

In a wider smart city scenario, we can consider the transport concern as one of the possible services that a smart city environment can offer, ranging from a consumption-aware city lighting management to the coordination of rescue interventions in case of e.g. adverse natural events. Therefore, to properly support the execution and integration of different smart city processes a proper cloud-based infrastructure is needed.

3. Workflow Management System

A smart city process can be formalised by means of a workflow description and given to a workflow engine that timely starts the composing services in the desired order over a cloud infrastructure [6]. Deploying and executing workflows on a cloud calls for a software architecture providing support for deploying,

executing, and monitoring services, while minimising resource waste and standard delays affecting operations typical of this infrastructure (i.e. VM instantiations).

In the scenario depicted in Figure 5, each workflow activity needs a dedicated description to guide proper resource allocations inside the cloud without worsening the level of service. In our solution, new workflow instances are started by a request coming from a *Web Service* (see the following section) and then a *Workflow Scheduler* finds the related workflow description and prepares resources for the execution of the first workflow activity. Then the proper cloud resources are used according to the activity description, e.g. *compute prices for on-demand stops* is a CPU intensive task, then a dedicated VM with large hardware flavour configuration is requested, where a flavour is a set of hardware characteristics for a VM [11].

An *activity description* is a tuple *(service_name, start_status, end_status, hw_flavour, sharing, estimated_duration)*: the service name identifies a specific basic cloud image previously created inside the cloud infrastructure and containing all of the binaries end executable necessary to run the activity. This will constitute the *software requirements* of the specific activity. Each activity must also be characterised by a start and an end status, together with a description of the cloud resource type and an indication about the needs for a dedicated or a shared resource. The start status of a service environment (VM), such as: already active, waiting for new process submissions (*on*); to be created and then turned on (*off*); *standby*, reducing resource consumption and response time by avoiding the creation of a VM. If we specify that *dedicated* resources are needed, we will assign a separate VM for task execution, while for *shared* resources (useful e.g. to handle simple tasks like querying or updating a database or retrieving a document from the object storage) we run a process inside a shared VM.

The *flavour* (e.g. could be set to small, medium and large) is used to characterize the *hardware requirements* for the VM that will be used to handle the workflow activity, such as allocated disk space, number of (virtual) CPUs and RAM memory. Finally, the *estimated_duration* is useful for schedule decisions, and will be constantly refined while more actual duration data are gathered during the next workflow executions. Not all the activities need a cloud resource to run: for example, a web service waiting for user requests (such as *receive route query* in Figure 5), and then triggering an actual workflow instantiation, need not be executed on a cloud, since it is only an interface for the more complex instantiation of a workflow over the cloud resources.

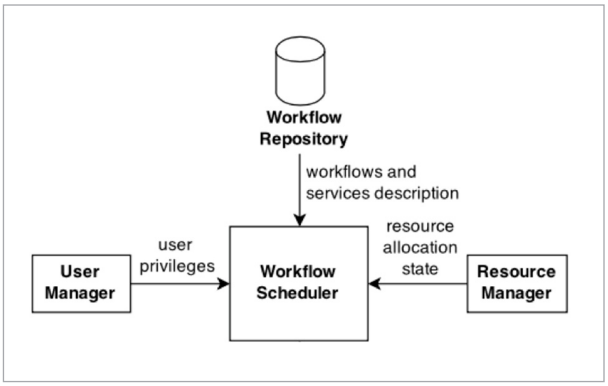
Together with the activity data described above, we have to provide additional data related to the whole workflow, such as e.g. the priority, the allowed deadline and the number of instances that can be concurrently executed. Moreover, for each service (regardless of the involving workflow), the number of its instances that can be concurrently active is also given. This is an important information to properly manage the constraints that must be applied for the concurrent execution of different instances of the same service inside the cloud. Even if we can think of a cloud as a limitless resource provider, we are forced to trade with the *pay-per-use* model to access resources. Therefore, the considered resource constraints are needed to limit the costs related to resource consumption.

3.1. Software Infrastructure

In order to support the features outlined above, we propose appropriate software components that enhance services and provide: (i) monitoring of workflow services, (ii) resource management, (iii) high availability.

Figure 6 shows how our components cooperate. Workflow Scheduler collects the user requests, determining whether to promptly accept them or not, ac-

Figure 6
Workflow Scheduler needs data from several other components to take its scheduling decisions



cording to the infrastructure load state. As any other scheduler, it manages a list of pending requests, using priority policies to determine their order, and interacts with other three components: User Manager, Resource Manager and Workflow Repository.

User Manager is responsible for the AAA service (Authentication, Authorisation, Accounting) related to user requests and will be asked to check user privileges and roles. Together with the number of requests previously accepted and completed for this user, it determines whether to lower the request priority. *Resource Manager* provides a high-level representation of cloud resource states, holds the scheduling outcomes, and let us know whether a given resource is available in a certain time frame. *Workflow Manager* handles the descriptions of the workflow recorded inside the *Workflow Repository*, and is responsible for workflow instantiation and the monitoring of the completion of each inner activity for the requested workflow. It interacts with the *Cloud Manager* to ask for the execution on the cloud of the services implementing each activity.

Cloud Manager is a Façade for cloud resources, hence providing the means for executing services inside the cloud infrastructure, which is actually managed by a cloud middleware such as e.g *OpenStack*. Each workflow activity will require the support of a service running on the cloud, either dedicated (in a separated VM) or on shared resources (in a separated process of a shared VM).

A *Monitoring Service* gathers data on the completion time of each activity for running workflows. Basing on such data, the accounting data associated with the requesting user will be updated, and a comparison is performed between the actual activity completion times and estimated times. Then such a comparison will be associated with the description of each activity.

3.2. The Flow of Operations and Use Case

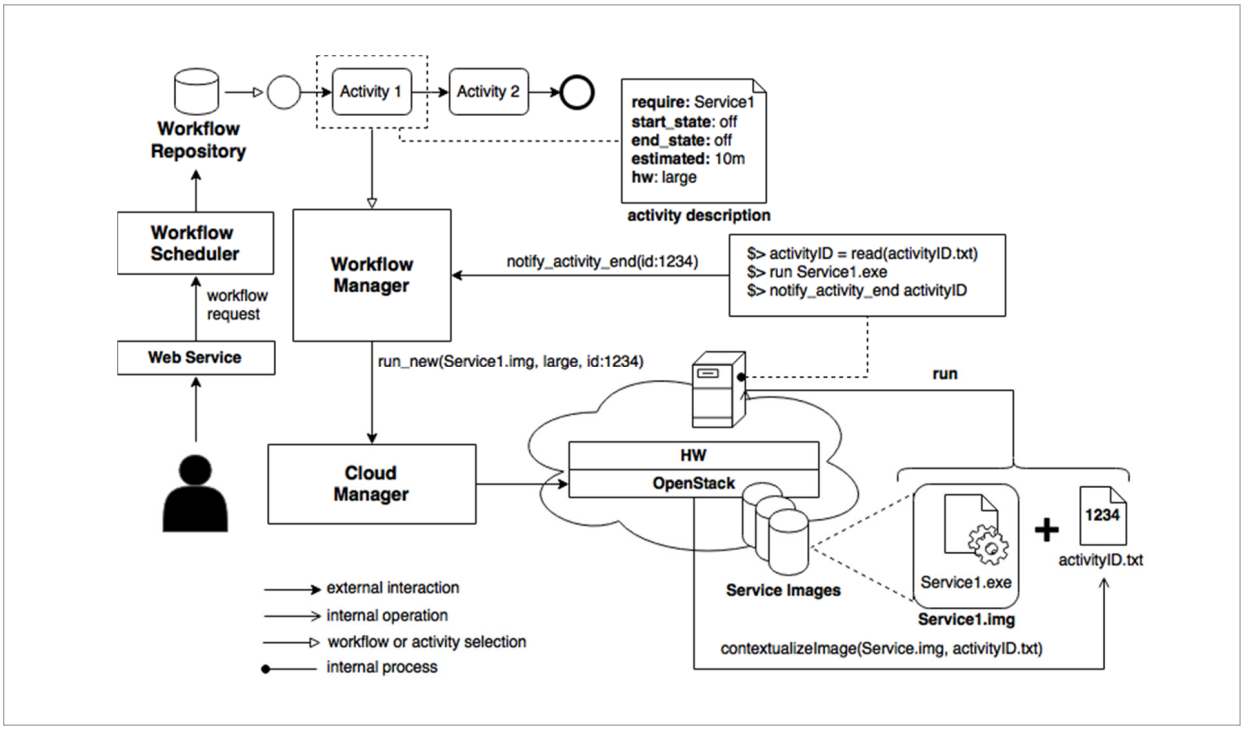
Figure 7 shows the interactions between the components of the workflow management system for the completion of a workflow execution request. Using, for example, a *web service*, the user can submit (either explicitly or not) an instantiation request of a specified workflow registered inside the management system.

This request arrives to Workflow Scheduler, that using the information coming from User Manager and Resource Manager determines whether to accept or reject the workflow execution request. It will then ask the Workflow Repository for the description of the requested workflow, providing the required services, resources and the estimated completion time for each of the inner activity. Workflow Manager will actually receive the instantiation request of a specified workflow, and starting from the first activity (*Activity 1* in Figure 7) it will be responsible for monitoring the execution state of each activity within the workflow. Workflow Manager interacts with the Cloud Manager to request the instantiation of the cloud services needed by the current activity, according to the workflow description.

In the depicted case (see Figure 7), we want to execute the first activity described in the selected workflow. The activity description shows that *Service 1* runs in a dedicated VM (because start state=off) with a "large" hardware flavor. The instantiation request is then sent from *Workflow Manager* to *Cloud Manager* that will create a new VM starting from the image that contains the desired service.

The *Cloud Manager* handles the execution of the single activity and need not be aware of the overall workflow composition. Its responsibility is to gather the requested resources from the cloud infrastructure, in our example managed using OpenStack. The interaction between the *Cloud Manager* and *OpenStack* is performed by using the *OpenStack API* (since our framework was built using Java, we used the *OpenStack4J* API) managing the communication with the different cloud services needed to drive the activity execution over the right cloud resources. The first step is to authenticate the *Cloud Manager* with the cloud interacting with the *Keystone (identity) service*; then the second step is to ask the *Glance (image) service* for the base image of the requested service (*Service1.img* in the use case). This image will contain all the executable needed to complete the specific activity. The next step is to ask the *Nova (compute) service* to create a dedicated *instance* (VM) of the selected image, that will be a *base* for our execution environment that need to be *contextualised* with the information and input data specific for the current activity instance. When the new instance is up and running, the *Cloud Manager* uses a direct *ssh* connection to proceed with the

Figure 7
Scheme of interactions between the components of the workflow management system for the completion of a workflow execution request



instance contextualisation, loading or giving a reference to the input files to be downloaded into the VM, together with a configuration file containing the activity information which the VM is related to. The most important configuration required is the *activityID*, that the VM uses to notify the activity completion to Workflow Manager. In this way, Workflow Manager is informed on the termination of a specific activity of the workflow, and can proceed with the destruction of the dedicated VM (because end state=off), asking also Cloud Manager to delete it. The Workflow Manager can then pass to the next activity described inside the workflow, and so on, until the workflow ends.

4. The WRNN Transport Request Predictor

The presented system constantly monitors the execution of each activity in order to keep track of the status of each workflow. The global status monitoring of each workflow provides inputs to a management system. The latter is responsible to provide constraints and priorities to the smart urban transport service in relation to the services requested by each passenger, as well as the distribution of the rendezvous points.

Then, passenger requests and the overall workflow status are known up to a recent past moment (the last data collection time), however in order to prepare operating conditions, it would be better to know the upcoming amount of passenger requests in the future. Such information is paramount to avoid service overload as well as over provision. This goal can be reached only by means of a finely tuned management of transport resources and human passengers. Without a prediction system, resources are usually managed by considering average load values computed over wide-ranging sets, hence are sub-optimal solutions at best.

Generally, the benefit of such strategies decreases while the load increases. Moreover, when the number of passengers overcomes available vehicles, passengers remain not served or suffer delays. However,

since the amount of required resources (e.g. drivers, vehicles, rendezvous points, etc.) is unknown in advance, this often causes over-provisioning, bringing negative effects on the related cost.

For this reason, we devised a predictive algorithm based on a Wavelet Recurrent Neural Network (WRNN) in order to predict the future *number of passenger requests in a certain location*, and therefore obtaining a suitable forecast of the future service load. The component responsible to perform and provide such predictions has been called *Request Predictor*, the related algorithm will be presented in this section.

The Request Predictor component takes as input the historical time series of requests, which have been collected by the management system during each workflow execution.

As known from the literature, a WRNN constitutes an optimal solution for the modelling and prediction of time evolving non-linear systems, therefore we pointed toward this solution for the development of the proposed predictor.

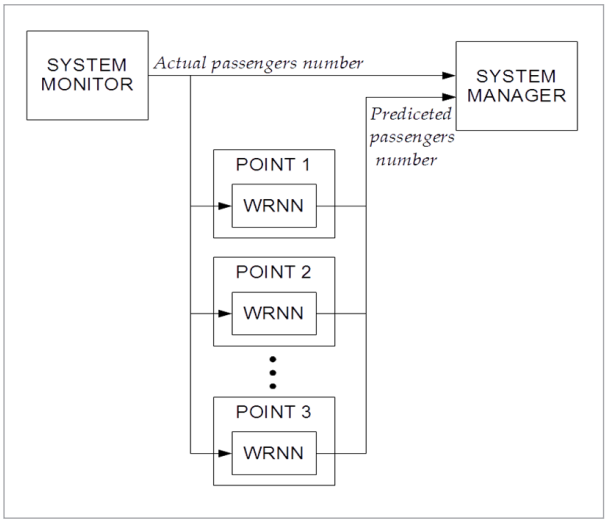
The WRNN architecture combines wavelet decomposition [15] with a peculiar neural network topology called nonlinear autoregressive networks with exogenous inputs (NARX) [21]: the first provide us with a transform capable to express the input data in a more suitable manner to train a recurrent neural network, while the second has the capability to model such non-linear systems. In this case, the latter feature strictly depends on the input transform into the wavelet domain, in fact it is possible to remove redundancies and other irregularities that could tamper with the network training.

As shown in the literature such an architecture is capable to devise a correct forecast for the future time evolution of a data series while both transforming data into the wavelet domain and then counter transform them into their original domain. In this manner, the overall architecture, while operating strictly on the wavelet domain, is capable to take inputs and return outputs on the original data domain, with no further intervention on the data series.

This twofold data transform procedure presents strong similarities with another approach to wavelet filters called Second Generation Wavelet Transform. The proposed model has been proven useful for the prediction of variable requests over time [17], and used as a basis for the predictor component to estimate the number of incoming passengers, hence the transport service load. The model gives such predictions as input for a dedicated management system allocating resources, i.e. vehicles and drivers. In order to comply with the required resource allocation in advance, the predictor models the number of incoming requests and passengers in each rendezvous point. Finally, the system determines the availability to service the upcoming requests. As shown in Figure 8, predictions are highly specialized for each one of the rendezvous point, hence a dedicated neural network returns a prediction set. Altogether the neural networks provide an overall status.

Figure 8

The adopted WRNN predictor models and predicts the future trends regarding the number of passenger waiting on a rendezvous point. A predictor is specialized for one rendezvous point, hence a set of predictors is used



This overall status prediction constitutes an early alert system that gives advices beforehand on the urgency of several transport routes as well as their predicted occupancy over time.

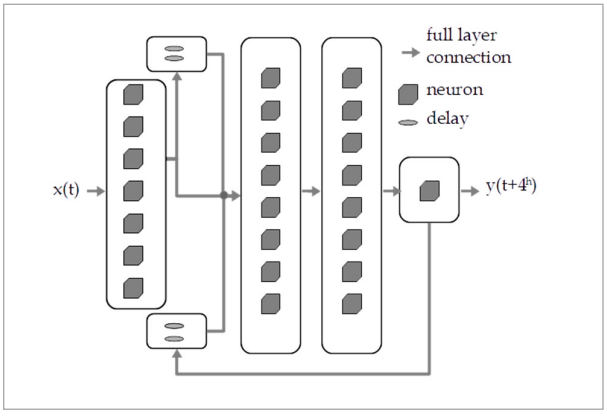
In this manner, it is possible to reschedule the service accordingly in order to better suit both passenger needs, in terms of newly allocated vehicles, and the companies needs in terms of unnecessary resources freed over time. The WRNN predictor operates with the time series indicating the passenger number at each rendezvous point over time. Firstly, the WRNN

predictor transforms the time series in the wavelet domain. The wavelet transform permits us to reduce data redundancies and obtain a representation that can express the intrinsic structure far more precisely than traditional analysis methods (e.g. Fourier transform). While the wavelet analysis exposes the time-frequency signature of the time series on different scales, the WRNN topology is the perfect complement to model the complexity of non-linear data correlation and perform data prediction on different scales. Thereby, a relatively accurate forecast of the passenger number can be achieved even when load peaks arise. The estimated result is fundamental for a management service that performs human and mechanical resources pre-allocation. The precision of our estimates allows just the right amount of resources to be used avoiding over provisioning. More details on WRNN architecture can be found in [3,17].

In this paper, we have adopted the Biorthogonal wavelet decomposition (this wavelet family is described in [15]), and for it symmetrical decomposition and exact reconstruction are possible with finite impulse response (FIR) filters [20]. The transformed data are fed to the WRNN [16]. The proposed WRNN consists of an input layer of 7 neurons, two hidden layers of 8 neurons with a radial basis activation function, a linear output layer with one linear neuron, and two delayed input units as well as two delayed feedback units from the output (see Figure 9). The neural network is fed with the data constituted by time steps of a time series in the wavelet domain representing the number of passengers requesting a transport services in a given location.

Figure 9

Devised neural network. Delays and feedback are obtained by using the relative delay lines and operators (D)



Using a discrete time index τ we can call $q_\mu(\tau)$ the number of passengers at a time τ for a certain rendezvous point μ. By applying the wavelet transform to the time series $q_\mu(\tau)$ we obtain the related representation in the wavelet space. Since we have used a 5-level transform, we have defined $q_\mu(\tau)$ as shown in equation 1, where the arrow represents the transform operation, $\widehat{W}$ represents the biorthogonal wavelet decomposition and the resulting vectors have the component values on the different decomposition scales (from scale 1 to scale 6, where the letter d indicates the wavelet coefficients and $a6$ the residuals on the most gross scale).

$$q^\mu(\tau) \xrightarrow{\widehat{W}} \left[q^\mu_{a_6}(\tau), q^\mu_{d_6}(\tau), q^\mu_{d_5}(\tau), q^\mu_{d_4}(\tau), q^\mu_{d_3}(\tau), q^\mu_{d_2}(\tau), q^\mu_{d_1}(\tau)\right] \quad (1)$$

For reasons related to the noise signature we were not interested in the last component of the transformed series, therefore we had an input vector $x_\mu(\tau)$ in the form shown in equation 2.

$$x^\mu(\tau) = \left[q^\mu_{a_6}(\tau), q^\mu_{d_6}(\tau), q^\mu_{d_5}(\tau), q^\mu_{d_4}(\tau), q^\mu_{d_3}(\tau), q^\mu_{d_2}(\tau), q^\mu_{d_1}(\tau)\right] \quad (2)$$

The overall input set, considering N time steps, can be then represented as a N x 7 matrix where the i-th row represents the i-th time step. Each row of this dataset is given as input value to the 7 input neurons of the proposed WRNN.

The properties of this network make it possible, starting from an input at a time step τ_n, to predict the number of requests and throughput at a time step $\tau_n + \sigma$.

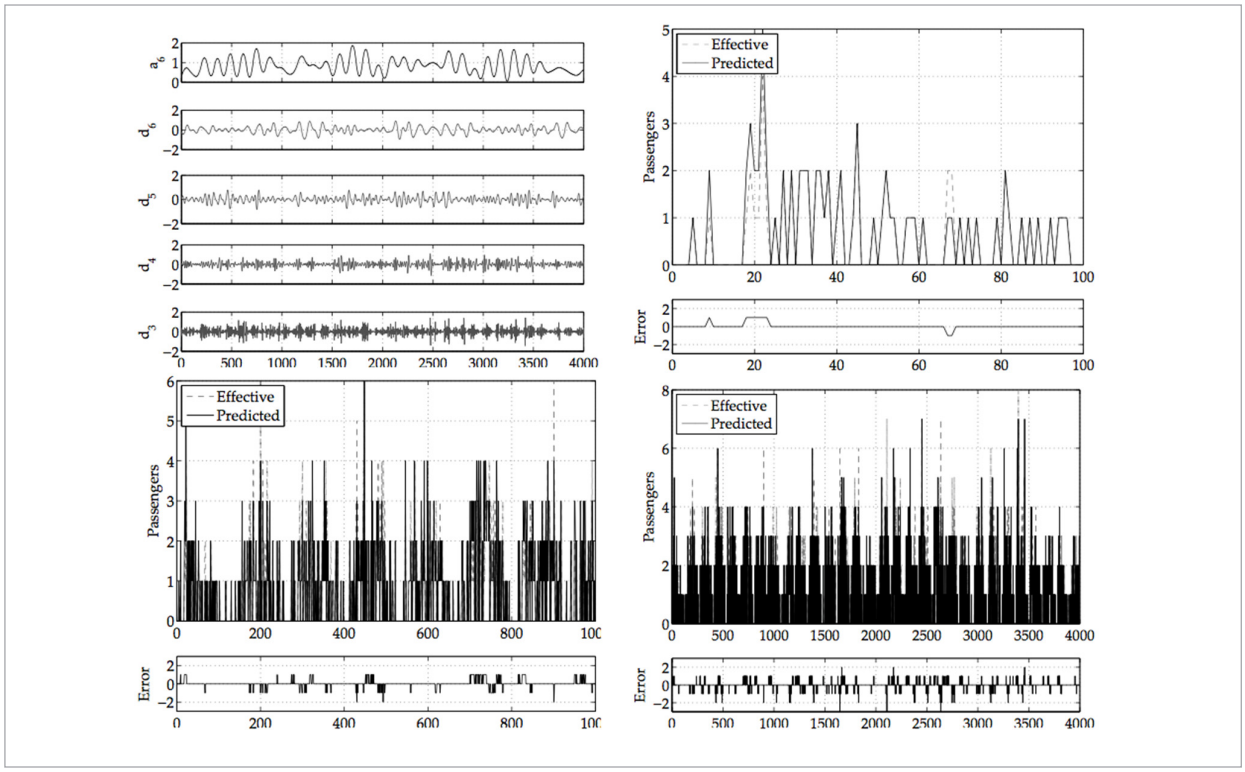
$$\tilde{x}^\mu(\tau_{n+\sigma}) = \widehat{N}_\mu[x^\mu(\tau_n)] \quad (3)$$

In this way, the WRNN acts like a function $\widehat{N}_\mu$ as defined by equation 3, where σ is the number of time steps of forecast in the future, and the tilde on the symbol $\tilde{x}$ indicates that it is a prediction instead of a measurement.

The number σ is not specified in the equation since it depends on the sampling frequency of the input and output time series. In our work, the data had a sampling period of 10 minutes, while we predicted the number of passengers 4 hours ahead, therefore with σ = 24. To model the time series and then to predict their future evolution the neural network is firstly trained with the historical time series, several train-

Figure 10

From left to right and top to bottom: the most gross scales of the wavelet decomposition of the historical time series of passengers for a selected rendezvous point; the predicted and measured throughput, the errors on the throughput predictions, the predicted and measured number of requests, the errors on the requests predictions. The grey dashed lines represent the effective measurements, the overlapping black lines represent the predictions made by the WRNN predictors



ing epochs are interleaved with the related supervised pruning procedure. When the process in concluded, a network starts to provide forecasts related to a specific service for which it was trained. By collecting each service forecast we obtain a complete map $\tilde{S}(\tau_{n+24})$, predicted beforehand, as defined by equation 4.

$$\tilde{S}(\tau_{n+24}) = \{\tilde{x}^{\mu}(\tau_{n+24}) \ \forall \mu\}. \quad (4)$$

The results of the WRNN predictor are shown in Figure 10. The predictor was able to propose an early estimate of the future number of passengers requesting a transport services in a certain rendezvous point. The maximum error occurring in the prediction is two passengers with respect to the effectively measured number. In our experiments the WRNN predictor has been used in order to schedule transportation means and manage an optimized planning of the related human and mechanical resources, and such operations are made possible thanks to predicted number of passengers, due to their availability in advance. Moreover, since each rendezvous point has been associated to a WRNN predictor, the proposed solution is general for any number and kind of transport services as well as for integrated transportation services. Finally, due to the cloud technology, the system could be expanded and scaled on demand.

5. Booking Management and Priorities

In the previous Section, we have seen how the WRNN can be used to predict the number of passenger requests in a certain location, so forecasting the future

service load to proper manage resource allocation. Another important problem for a transportation service provider is to properly manage seats booking in order to better serve the customer requests, trying to sell all available seats while avoiding overbooking.

To address this problem, when the number of available seats for a transportation vehicle goes beyond a certain threshold (properly chosen by the service provider according to a typical amount of seat requests for that particular route), the system will handle further booking requests for the remaining seats in a way that users which are *more likely to buy* a ticket will have *more time* for such an operation, i.e. more chances to reserve the remaining seats.

The rational for this is that on the one hand we want to give an advantage to *habitual customers*, since they probably will book one of the remaining seats, so we can give them more time to complete the booking; on the other hand, by giving less time to *non-habitual customers* we can both have a faster feedback on their actual decision and also encourage them to complete their purchase, such as by means of "buy now" alerts.

Our system records user activities to build a personal profile for each user id. Recorded data for a user comprise the number of tickets that have been bought (N_B) and the number of times the user has asked to view the details for a ticket (N_V). Then the ratio N_B/N_V is computed, which represents the *likelihood* of the user to buy a ticket (T_u).

Then, a personal *expiring time threshold* is given to each customer to complete their booking for the remaining seats. This threshold depends on T_u for the user. Let us suppose that the transport service provider sets a maximum time to buy a ticket as 10 minutes before the departure (this is at time $t = 0$), we will call it the BOARDING phase. Then, only users having $T_u \approx 1$ will be allowed to buy a ticket until the last moment. Other users, having lower values of T_u, will be required to conclude at an earlier time the procedure. The remaining time will possibly allow another (more habitual) customer to see the seat as available and, more likely, buy it.

Proper messages will be given to customers viewing seats availability over time, such as a "limited-time available" and "buy now", before the status "closed".

The remaining time is determined by means of a status equation given by the following

$$S(T_u, t_0, t_{max}; t) = \frac{logT_u(t-t_0)^2}{log(t_{max}-t_0)^2} \quad \forall\, t_0 < t < t_{max} \tag{5}$$

where T_u is the likelihood to buy a ticket, t_0 is the BOARDING time, t_{max} is the booking OPENING time, and t is the wall clock time. With such parameters, the system state will be set so that

$$\begin{cases} 1 > & S(T_u, t_0, t_{max}; t) > 0.8 & \Rightarrow & \text{the system is OPEN} \\ 0.8 > & S(T_u, t_0, t_{max}; t) > 0.6 & \Rightarrow & \text{the system gives a WARNING} \\ 0.6 > & S(T_u, t_0, t_{max}; t) > 0.4 & \Rightarrow & \text{the system is in BUY NOW phase} \\ 0.4 > & S(T_u, t_0, t_{max}; t) > 0 & \Rightarrow & \text{the system is CLOSED} \end{cases}$$

$S(T_u, t_0, t_{max}; t)$ has been devised to be proportional to Tu, the quadratic form $(t\text{-}t_0)_2$ grants us to have the time as leading term for the equation. The denominator is a normalization factor; and the logarithmic form has been chosen for its robustness with respect to small variations of the parameter that would have led to instability.

Together with the status equation S(), we consider the actual seat availability on the vehicle. Therefore, we modify the status formula according to the percentage of free seats. Therefore, $S()$ is used to manage booking operations until the number of available seats k drops under a certain threshold k_{max} (e.g. 20%). Then, given $k<k_{max}$, we compute a coefficient

$$c(k) = \sqrt{\frac{k}{k_{max}}} \tag{6}$$

in order to obtain a modified status equation

$$S_k(T_u, t_0, t_{max}; t) = S(T_u, t_0, t_{max}; t)c(k). \tag{7}$$

Since it is $k<k_{max}$ it follows $c(k)<1$. Therefore, this $c(k)$ coefficient will have the effect to anticipate the timing computed by $S(T_u, t_0, t_{max}; t)$. In this manner, as a vehicle reaches full occupancy, after a certain occupancy threshold, we anticipate the WARNING and BUY NOW phase by reducing the time that is given to the users to buy a ticket. This effect is stronger for users having a low value of T_u (see Figure 11).

Therefore, a user having a high T_u will barely notice the difference until very few seats are available. Conversely, a user having a low T_u witnesses a dramatic decrement of the remaining time to buy the ticket when the number of seats availability drops under a certain threshold.

Note that the $c(k)$ coefficient does not scale linearly, on the contrary it depends by the square root of the normalized number of available seats (k/k_{max}). The

square root lets us dump the effect of $c(k)$ for as long as possible, while granting a fast reaction of the system when only few seats are still available.

6. Related Works

In our work, we leveraged smart phone sensed data, such as GPS position, to provide users with transport services. Future urban environments will likely gather data from several cheap sensors, using cross-correlation and analytical models to mine valuable and reliable information from a lot of noisy and unreliable heterogeneous sensors. Smart Cities scenarios offer new challenges in coping with the amount of heterogeneous sensed data that need to be collected and organised to give uniform and useful exposed information [19]. Additionally, the integration between urban sensor infrastructures and Cloud services presents research challenges concerning the security issues related to the use of sensed data can affect our privacy [7]. Other works in the literature have covered the optimisation of a city transportation service, e.g. using prediction models to estimate traffic condition [13] or leveraging more flexible electrical vehicle spread over the urban environment having the scheduling decision managed by a cloud infrastructure [24].

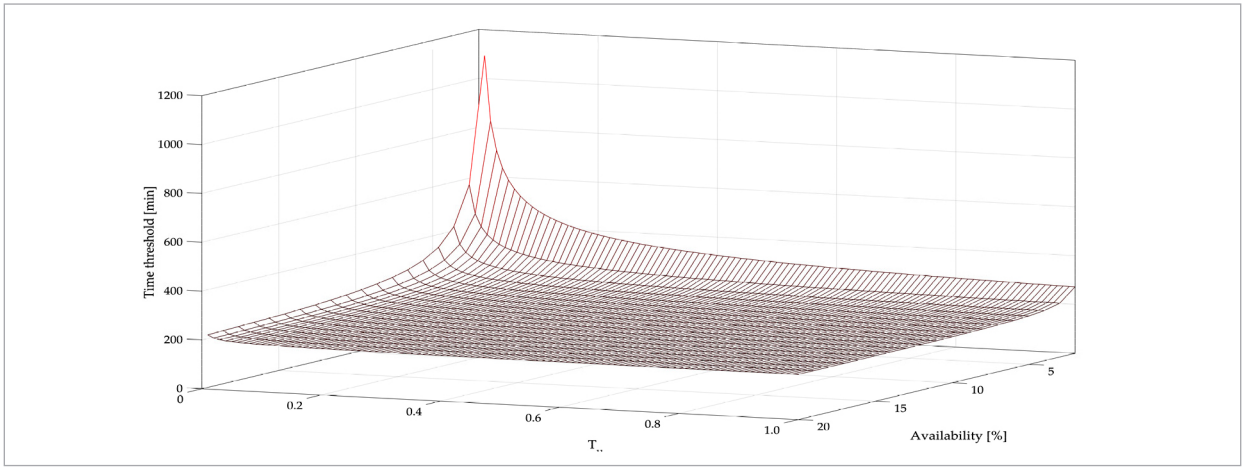
In [13] the authors combine actual data gathered by sensors displaced in the city environment, with future traffic prediction to provide individual trip planning for transport users. They used a spatial-temporal random field model and a Gaussian process regression to predict traffic condition and to estimate vehicles concentration in areas with low sensor coverage. This approach differs from ours both in the used predictive model and in the considered service point of view. While they highlight congested streets and city areas, hence accordingly organise user trips, we have rendezvous-points, i.e. bus stops, that the transport service vehicles have to reach to serve user requests. Therefore, in our approach it is the city environment, i.e. the devised city transportation service, that adapts itself to user requests, not vice versa.

In [24] the authors propose a public vehicle system that uses a cloud infrastructure to devise scheduling strategies and paths based on the demands of passengers. The proposed electric vehicles can transport more passengers than a taxi and do not have stops or routes like a bus service. The service they propose is different from ours, since they rely on a set of constraint based algorithms to compute the best solution satisfying user requests, instead we use a prediction model to assist scheduling decisions. We additionally provide predefined bus stops (rendezvous-points) which are activated on demand according to user requests, hence adapting the overall trip schedules.

In [10] an extensive survey of the literature on quality of service, availability and performance for distributed applications has been given. All the approaches in [10] are not intended to be used in combination with

Figure 11

The timeframe available to confirm the purchase of a ticket varies from customer to customer according to the likelihood of buying it (T_u), given to each customer by the system, and the availability of seats



cloud resources. Hence, further support is needed as shown in the sections above.

In [23] the authors use pre-processing stages in order to feed filtered data to neural networks to model time series with both seasonal and trend patterns. Hybrid models are widely used in the literature in order to model phenomena and obtain forecasting software systems for a wide range of purposes, such as e.g. hydro-geological time series and the related risk assessment [12,14]. Other kinds of neural network related approaches have been developed for traffic prediction, e.g. basing on a flexible neural tree and particle swarm optimisation algorithm [4].

Moreover, when wavelet transforms had been used in other contexts, they have been proved useful to properly characterise information in signals [18]. Other techniques have been used to model the distributed behaviour of complex systems [22].

Our approach provides a novel solution that encompasses new strategies and an overall advanced service. New means have been provided to interact with users to gather transport requests, predictive models have been employed to estimate future demands, and schedules for vehicles have been handled by a cloud infrastructure capable to execute workflows.

7. Conclusions

This paper has proposed an approach that on the client side provides a software solution giving users support to book transport vehicles, while on the server side support is given to deploy, execute, and monitor services on a cloud and according to workflows. The server-side components are independent of specific workflows and can execute services in a variety of ways to properly govern the life-time of services.

In our solution, a component has been specifically devised to plan the needed transport service ahead of time by modelling incoming requests and analysing them to remove noise, while characterising repetitive trends. This is performed by transforming data to the wavelet domain before giving them to a neural network component. Then, we are able to start operations, such as planning vehicles routes and driver shifts, avoiding over-provisioning.

Acknowledgments

This work has been partially supported by project PON CLARA SCN_00451 funded by the Italian Ministry of University.

References

1. Batty, M., Axhausen, K. W., Giannotti, F., Pozdnoukhov, A., Bazzani, A., Wachowicz, M., Ouzounis, G., Portugali, Y. Smart Cities of the Future. The European Physical Journal Special Topics, 2012, 214(1), 481-518. https://doi.org/10.1140/epjst/e2012-01703-3
2. Benevolo, C., Dameri, R. P., D'Auria, B. Smart Mobility in Smart City. In Empowering Organizations, Springer, 2016, 13-28. https://doi.org/10.1007/978-3-319-23784-8_2
3. Capizzi, G., Napoli, C., Paterno, L. An Innovative Hybrid Neuro-Wavelet Method for Reconstruction of Missing Data in Astronomical Photometric Surveys. In Artificial Intelligence and Soft Computing, Springer, 2012, 7267, 21-29. https://doi.org/10.1007/978-3-642-29347-4_3
4. Chen, Y., Yang, B., Meng, Q. Small-Time Scale Network Traffic Prediction Based on Flexible Neural Tree. Applied Soft Computing, 2012, 12(1), 274-279. https://doi.org/10.1016/j.asoc.2011.08.045
5. Chourabi, H., Nam, T., Walker, S., Gil-Garcia, J. R., Mellouli, S., Nahon, K., Pardo, T., Scholl, H. J. Understanding Smart Cities: An Integrative Framework. In IEEE Proceedings of 45th Hawaii International Conference on System Science (HICSS), 2012, 2289-2297.
6. Erl, T. SOA Design Patterns. Pearson Education, 2008.
7. Fazio, M., Paone, M., Puliafito, A., Villari, M. Heterogeneous Sensors Become Homogeneous Things in Smart Cities. In IEEE Proceedings of International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012, 775-780. https://doi.org/10.1109/IMIS.2012.136
8. Fornaia, A., Napoli, C., Pappalardo, G., Tramontana, E. Enhancing City Transportation Services Using Cloud Support. In International Conference on Information and Software Technologies (ICIST), Springer Communications in Computer and Information, 2016, Science, 695-708. https://doi.org/10.1007/978-3-319-46254-7_56
9. Fornaia, A., Napoli, C., Pappalardo, G., Tramontana, E. Using AOP Neural Networks to Infer User Behaviours and Interests. In Proceedings of the 16th Workshop from Objects to Agents (WOA), 2015, 1382, 46-52.
10. Guitart, J., Torres, J., Ayguadé, E. A Survey on Performance Management for Internet Applications. Concurrency and Computation: Practice and Experience, 2009, 22(1), 68-106. https://doi.org/10.1002/cpe.1470

11. Jackson, K. OpenStack Cloud Computing Cookbook. Packt Publishing Ltd., 2012.
12. Jain, A., Kumar, A. M. Hybrid Neural Network Models for Hydrologic Time Series Forecasting. Applied Soft Computing, 2007, 7(2), 585-592. https://doi.org/10.1016/j.asoc.2006.03.002
13. Liebig, T., Piatkowski, N., Bockermann, C., Morik, K. Dynamic Route Planning with Real-Time Traffic Predictions. Information Systems, 2016, 64(C), 258-265.
14. Lohani, A., Kumar, R., Singh, R. Hydrological Time Series Modeling: A Comparison Between Adaptive Neuro-Fuzzy, Neural Network and Autoregressive Techniques. Journal of Hydrology, 2012, 442-443, 23-35. https://doi.org/10.1016/j.jhydrol.2012.03.031
15. Mallat, S. A Wavelet Tour of Signal Processing: the Sparse Way. Academic Press, 2009.
16. Mandic, D. P., Chambers, J. Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. John Wiley & Sons, Inc., 2001. https://doi.org/10.1002/047084535X
17. Napoli, C., Pappalardo, G., Tramontana, E. A Mathematical Model for File Fragment Diffusion and a Neural Predictor to Manage Priority Queues over Bittorrent. International Journal of Applied Mathematics and Computer Science, 2016, 26(1), 147-160. https://doi.org/10.1515/amcs-2016-0010
18. Połap, D., Woźniak, M. The Use of Wavelet Transformation in Conjunction with a Heuristic Algorithm as a Tool for Feature Extraction from Signals. Information Technology and Control, 2017, 46(3), 372-381.
19. Puliafito, A., Celesti, A., Villari, M., Fazio, M. Towards the Integration Between IoT and Cloud Computing: An Approach for the Secure Self-Configuration of Embedded Devices. International Journal of Distributed Sensor Networks, 2015, 11(12), 1-9. https://doi.org/10.1155/2015/286860
20. Rabiner, L. R., Gold, B. Theory and Application of Digital Signal Processing. Prentice-Hall, 1975.
21. Williams, R. J. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation, 1989, 1(2), 270-280. https://doi.org/10.1162/neco.1989.1.2.270
22. Woźniak, M., Połap, D., Napoli, C., Tramontana, E. Application of Bio-Inspired Methods in Intelligent Gaming Systems. Information Technology and Control, 2017, 46(1), 150-164. https://doi.org/10.5755/j01.itc.46.1.13872
23. Zhang, G., Qi, M. Neural Network Forecasting for Seasonal and Trend Time Series. European Journal of Operational Research, 2005, 160(2), 501-514. https://doi.org/10.1016/j.ejor.2003.08.037
24. Zhu, M., Liu, X. Y., Qiu, M., Shen, R., Shu, W., Wu, M. Y. Transfer Problem in a Cloud-based Public Vehicle System with Sustainable Discomfort. Mobile Networks and Applications, 2016, 21(5), 890-900. https://doi.org/10.1007/s11036-016-0675-y

Summary / Santrauka

Smart cities providing connectivity to users and other advanced services can be leveraged to improve public transport services. This paper proposes a solution that lets citizens request a public vehicle to perform additional stops off the main route, hence achieving a customisation of the transport operator services to better assist users. A cloud infrastructure and a proper distributed architecture have been designed to assess whether user requests can be accepted. The proposed software solution considers viable the requests that can fit to available secondary routes, while also satisfying other user demands that have been previously accepted. Then, drivers will be alerted in advance in order to adapt their route.

Protingi miestai, aprūpinantys vartotojus susijungimo galimybėmis ir kitomis pažangiomis paslaugomis, gali būti pasitelkti ir visuomeninio transporto paslaugoms pagerinti. Straipsnyje siūlomas sprendimas, kuris gyventojams leidžia pateikus užklausą viešojo transporto priemonės paprašyti papildomų sustojimų, nesančių pagrindiniame transporto priemonės maršrute. Tokiu būdu užtikrinamas geresnis transporto operatoriaus paslaugų pritaikymas pagal vartotojų poreikius. Tam, kad būtų galima įvertinti, ar vartotojų užklausos gali būti priimamos, buvo suprojektuota debesies infrastruktūra ir tinkama paskirstytų sistemų architektūra. Siūlomas programinės įrangos sprendimas gyvybingomis laiko tokias užklausas, kurios gali tikti į pasiekiamus šalutinius maršrutus, tuo pat metu patenkinant ir ankstesnes iš kitų vartotojų priimtas užklausas. Tokiu būdu vairuotojai iš anksto gauna pranešimą ir gali atitinkamai pakeisti maršrutą.