A System for Green Personal Integrated Mobility: Compensation Engine

Francesco Rizzi
Dept. of Industrial and Information Engineering, University of Pavia
Via Ferrata 5, 27100 Pavia, Italy

Gianmario Motta
Dept. of Industrial and Information Engineering, University of Pavia
Via Ferrata 5, 27100 Pavia, Italy

Daniele Sacco
Dept. of Industrial and Information Engineering, University of Pavia
Via Ferrata 5, 27100 Pavia, Italy

ABSTRACT
This paper presents a component of the ongoing research project Integrated Real-time Mobility Assistant (IRMA). The component's name is Compensation Engine. IRMA is a software system that targets the personal mobility in a near future scenario, based on green, shared and public transports. IRMA handles end-to-end itineraries that may involve multiple transport systems, and supports the users in schedule and re-schedule their itineraries. This paper focuses on the description of the Compensation Engine component, which monitors the progress of the journey and spots possible transportation issues. The component alerts the user when the journey can not be completed and allows the rescheduling of the route. The Compensation Engine has been implemented and proved on test cases.

General Terms
Software engineering, Applications of Computer Science in Modeling, Data and Information Systems

Keywords
Transport systems; software engineering; applications of computer science in modeling data and information systems; smart cities; urban mobility; human mobility; mobility integrator; service oriented architecture.

1. INTRODUCTION: THE IRMA PROJECT
The Integrated Real-time Mobility Assistant (IRMA) targets individuals in whole lifecycle of their mobility. Work started in Department of Information and Industrial Engineering of Pavia University to develop a mobile application that could assist travelers to meet their schedule even with transport disruptions [1].

IRMA architecture includes various elements, namely mobility analysis, mobility forecasting, mobility assistant, terminals, communication services, sources which is shown in **Fig. 1**. The personal mobility assistant shall assist the end user to plan, configure, monitor, alarm, and reschedule mobility across multiple mobility options. It manages and supports the mobility itinerary by two phases in the mobility life cycle with different modules/services [2].

Before the trip (Planning), the Request Handler processes the mobility request that may concern an individual trip or a calendar. The Handler helps the user to define the optimal mobility plan by accessing mobility forecast and mobility timetables through the Information Retrieval sub service. The user will choose and confirm the ideal option as an individual itinerary.

During the trip, the user receives information about disruptions (Event Notifier) and use the assistance to choose a viable alternative (Compensation Engine). Event Notifier is a set of instances that are activated when the user confirms and actually starts the trip. It concerns all connections of the individual itinerary and relevant process disruption information provided by communication services. Compensation Engine processes mobility alternatives in front of a disruption or a change, by browsing on mobility analyzer the closest option (as an alternative the request handler could fetch a plan B in advance). A more comprehensive description of the IRMA project can be found in the paper at the reference [2].

The second section of this paper presents an overview about the real-time information in the mobility field. This kind of information is important to enhance the user experience. The third section introduces the reader to the main concept of the Compensation Engine and to its terminology. The design and implementation of the component are described in the fourth and fifth section. The UML diagrams are provided for a better explaination. The sixth section is about the description of the functionality test. In the end, some conclusions are outlined with some proposal for the future evolution of the Compensation Engine component.

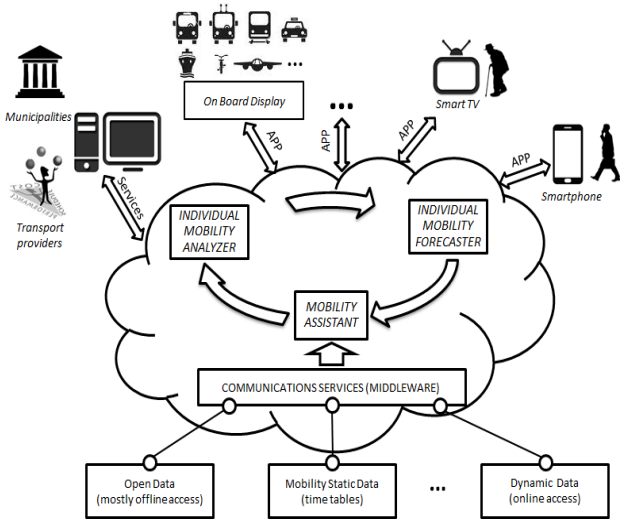

Fig. 1: IRMA Architecture

2. RELATED WORK

Mobility assistance systems are evolving by taking into account several research fields. Some systems focus on tourism [3], others on disabled or not self-sufficient people [4]. Many solutions support the travelers to search, filter and choose the best offer [5], [6], [7]. Many web oriented solutions have been developed to better fit the needs of the travelers [8, 9, 10]. A good incentive has been the widespread use of smartphone devices.

Current mobility can be defined as mono-modal because it is based on a single type of transportation mean. This system addresses multimodal mobility. This type of mobility is still at a beginning phase and represents a potential advance in the field of mobility. The travel experience of the user can be enhanced by implementing a multimodal mobility system. This is a good return for the transportation companies. Today public transportation travelers need two kind of information [11]: 1) static information: as the departure time and arrival time, 2) dynamic information: as delays or cancellations of carriers on the route.

The usage of dynamic information is increasing into the transportation terminals like rail stations or bus stops. A notable amount of money has been spent every year by transportation companies for real-time information systems. Screens are used more and more to provide real-time information about delays, cancellations, incoming departures or platform changes [12]. It has been verified that the usage of real-time information allows transportation companies to offer a better service level to the customers, because they enhance the reliability and the quality of the whole amount of provided information [13]. A multimodal trip planner system should be aware of the updated travel time and delay problems, in this way it can provide reliable information about the feasibility of the trip [14]. Another key point is the simplification of the trip planning task [15], because the user should be able to reschedule the trip plan in every moment. A typical example of rescheduling happens when the traveler changes a vector for a cheaper one. In this case, the user should be able to compare alternative vectors on the same route and reschedule according to his or her needs. In several sources the rescheduling is presented as a value for the traveler and a winning feature for the system [16], [17], [18], [19], [20].

3. COMPENSATION ENGINE CONCEPT

Compensation Engine is a component of IRMA. Its aim is to be aware of the issue on each registered journey in the system.

A specific terminology is adopted:

1. Journey: It is every user's trip. It is featured by a starting and ending point/time. It is composed by one or more different steps called Connections. Not all of the Connections are peformed by the same type of vector or by the same public transportation provider.
2. Connection: It is a single step into the user's Journey. Every Connection is featured by a starting and ending point location and time. A Connection is also featured by a vector, that is supposed perform the transportation.
3. Transportation carrier: any mean that is supposed to carry the user from a location to another location. This system focuses on public transportation means.

The Compensation Engine manages the journeys thanks to two lists, inbound and outbound journey list. These two journey lists are sorted by time. Every journey has got a states and state transictions as it is shown in the state diagram in Fig. 2. The states are three: scheduled, running, completed.

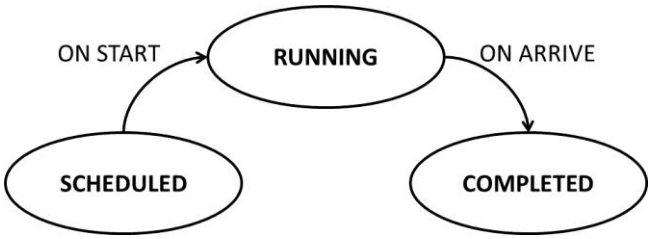

Fig. 2: State Diagram

A journey is moved through the two lists according to its state. The scheduled journeys are inserted in the inbound list. The running journeys are inserted in the outbound list. The completed journeys are removed from outbound list. The mechanism of the two lists is showed in Fig. 3.

Every running journey gets its own monitor object that is responsible of the successful completion of the journey. The monitor object uses a set of further objects that constantly monitor the transportation carriers. This last operation is accomplished by retrieving third part real-time information from transportation companies. In case the journey can not be completed, the Compensation Engine allows the user to reschedule the journey.

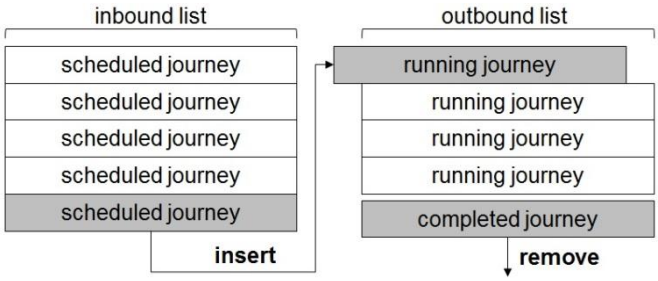

Fig. 3: Inbound and outbound lists

4. ANALYSIS OF NEEDS

4.1. Usage Scenario

A situation is proposed to highlight the problems that the Compensation Engine is supposed to solve:

Mister Rossi books a trip by train from Pavia to Rome. The trip involves a change in Milan. He arrives to Pavia rail station on time, but his train has got a delay. Mister Rossi won't be able to arrive on time in Milan in order to take the train to Rome. Luckily, he notices the problem and he goes to the ticket office to reschedule his trip plan. Unfortunately, the line in front of the ticket office is too long.

This scenario is uncomfortable for the traveler. He or she needs to accomplish several steps and be aware of different variables to solve the problem. The aim of the Compensation Engine is to automatize as much as possible these steps and so reduce the worries of the traveler.

4.2. Process Modeling

The analysis starts from this example. First, the logic of the process is outlined. The assembly line in Fig. 4 shows the sequence of macro-activities of the process. The assembly line diagram is a variant of a Business Process Diagram (BPD).

The assembly line diagram is successfully used for process modeling [21]. The diagram shows the interactions (read/write) between the process and the information objects during the execution. In this case the information objects are identified by the database.

4.3. Requirement Design

A set of references in an assembly line diagram typically becomes a use case that the information system has to provide. This is very important, because it maps the business process to use cases. The use cases describe the functional requirements of an information system and its actors [21]. The "Choose an alternative" use case was identified, its description is in Table 1.

Table 1. Use case description table: Choose an Alternative

Choose an alternative	
actors	User
description	the system alerts the user and allows the user to reschedule the journey.
preconditions	a journey has been already saved and it can not be completed because of an issue on a transportation carrier.
normal flow	1. the system recognize an issue on a carrier (delay or cancellation) 2. the system checks the promise of the journey. 3. the promise is not respected. The system notifies the user. 4. the user sees the notification 5. the system allows the user to reschedule the journey 6. the user modifies the journey
alternative flow	3. the promise is respected

5. COMPONENT DESIGN

5.1. Deployment Diagram

The deployment diagram is presented in Fig. 5. The Compensation Engine is located to the server-side. In this way, the workload of the client is lightened.

Third part servers are owned by transportation companies. They are used to retrieve real-time information about transport carriers.

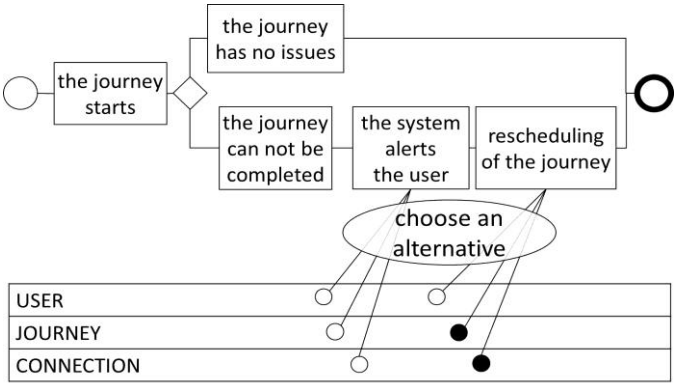

Fig. 4: Assemly line. Full circle: information writting. Empty circle: information reading

Compensation Engine communicates with the Persistence component and the EventsNotifier component. The Persistence component manages the access to the IRMA server-side database. The EventsNotifier component manages the notification delivery to the client.

5.2. Class Diagram

This section describes the classes involved into the Compensation Engine. Much importance has been attached to

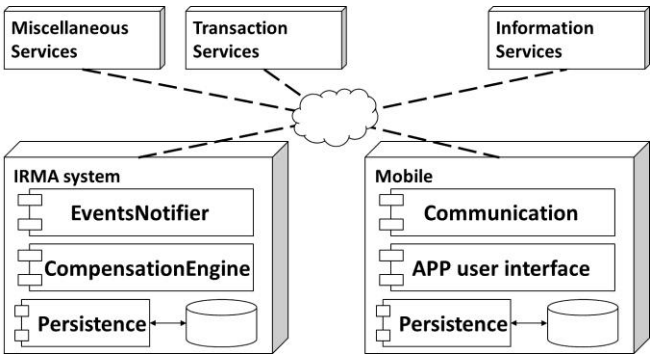

Fig. 5: Deployment Diagram of IRMA.

the modularity and maintenance of the system. The classes of the component are seven:

1. JourneyGuardian
2. PromiseGuardian
3. ConnectionListener
4. CLManager
5. AlternativeManager
6. JourneyRecord
7. ConnectionRecord

The Fig. 6 shows the class diagram. The Compensation Engine has got a multilevel approach on the journey. The JourneyGuardian class is at the higher level and it treats the journey as a single entity. The JourneyGuardian class manages the journey state transitions.

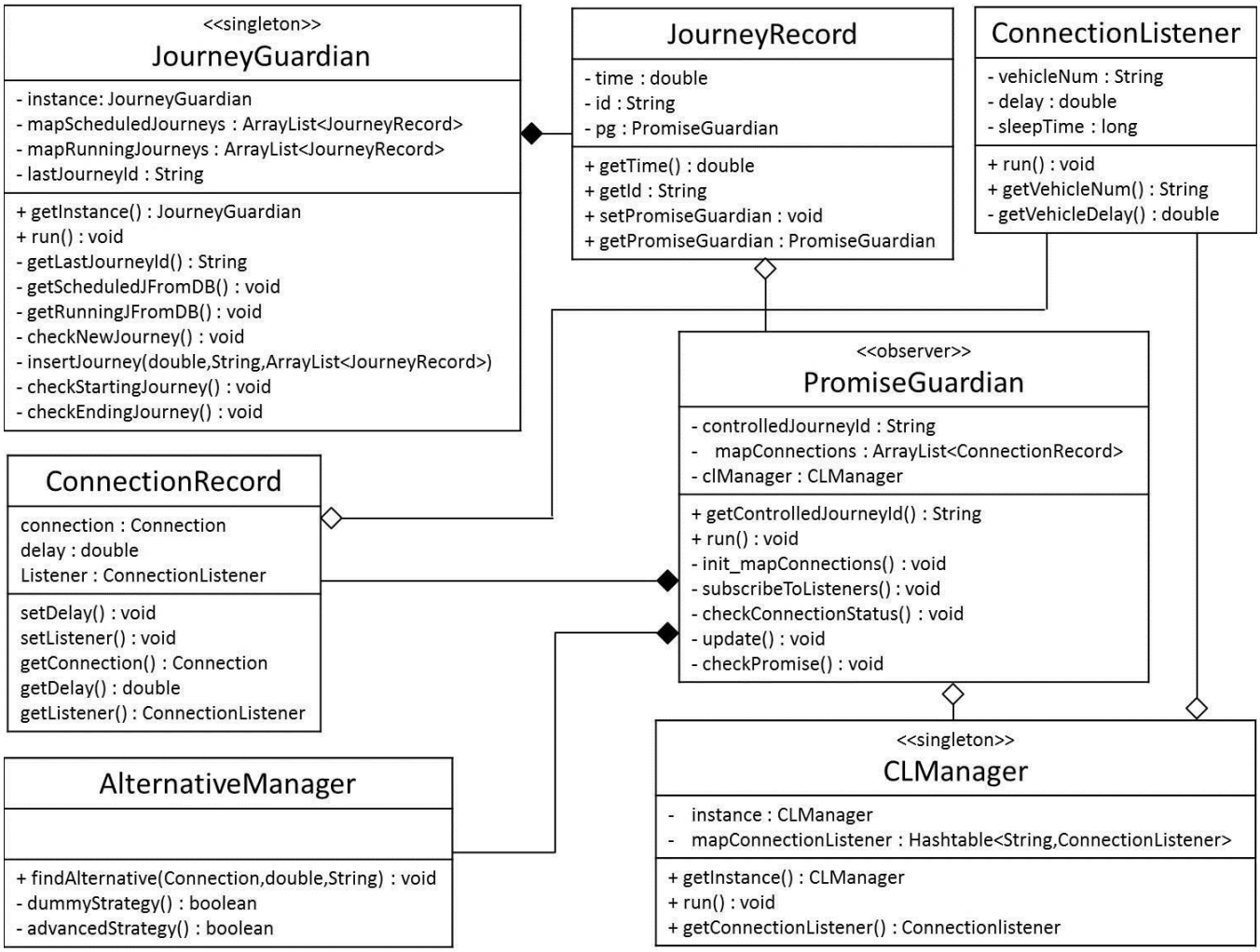

Fig. 6: Class diagram

Every journey gets its own PromiseGuardian instance. The PromiseGuardian sees the journey at a lower level, so as a sequence of connections.

The PromiseGuardian needs to be constantly aware about issues on transportation carriers and use ConnectionListener class that retrives connection information from third part sources. Since more than one PromiseGuardian could be interested in the same connection, the CLManager class manages the ConnectionListener objects and their usage. Aim of this class is to associate a PromiseGuardian with the connections of its journey.

JourneyRecord and ConnectionRecord classes are data type of the records of the lists that are used into the Compensation Engine. A more detailed explanation of the other classes is provided below.

5.2.1. JourneyGuardian

In the JourneyGuardian the state transiction of the journey is managed thanks to an inbound and an outbound list. The scheduled journeys are into the inbound list and the running journeys are into the outbound list. The scheduled journeys are sorted according to the departure time in descending order. The running journeys are sorted according to the arrival time in descending order. The descending order is guaranteed by the method JourneyGuardian.insertJourney(), it inserts a journey into a list without compromising its order. The descending order makes sure that: 1) the last element of the scheduled journeys is the next starting journey, 2) the last element of the running journeys is the next ending journey.

The JourneyGuardian constantly checks the last element of the lists. First, it checks which journeys have been already started and which journeys are starting in thirty minutes. A PromiseGuardian is created for these journeys. They are moved to the outbound list and become running journeys. In this way a journey is monitorized by its PromiseGuardian thirty minutes before the departure.

The class checks also the running journeys. When a journey is finished, its state becomes completed and it is removed from the outbound list.

The class implements the Singleton Design Pattern [22] to ensure only one instance of the class and to avoid managing conflicts.

5.2.2. PromiseGuardian

The PromiseGuardian class manages one single journey and its related connections. A ConnectionListener is created for every Connection of the Journey. The Observer Design Pattern [22] is implemented. The PromiseGuardian takes the role of the observer and the ConnectionListener takes the role of the subject.

The method PromiseGuardian.checkPromise() implements an algorithm, the so called check-promise algorithm (it is described further in this paper). The algorithm's aim is to check that the time interval between every Connection doesn't last less then a fixed threshold. The algorithm ensures enough time to the user for switching from a carrier to another one. The time interval can last more because of delays on carriers, in this case the PromiseGuardian calls the AlternativeManager.

5.2.3. ConnectionListener

The ConnectionListener periodically checks the carrier. It retrieves information like actual departure time, arrival time, delay, cancellation, actual departure and arrival platform.

The ConnectionListener implements the Observer Design Pattern [22] as the role of the subject.

In the proof of concept implementation, it takes information by parsing every 5 minutes an Italian website for railways www.viaggiatreno.it.

5.2.4. CLManager

The CLManager class is meant to support the Observer Design Pattern [22] between PromiseGuardian and ConnectionListener. Its aim is to manage the subscription of the PromiseGuardians to the ConnectionListeners and to eventually create the latters.

When two or more PromiseGuardians are interested to the same carrier, the CLManager creates only one ConnectionListener that will be unique for that carrier.

CLManager keeps a list of existing ConnectionListeners and implements the Singleton Pattern to keep that list unique.

5.2.5. AlternativeManager

The **AlternativeManager** class is instantiated by the PromiseGuardian when the promise is not respected. Its aim is to make the user able to reschedule the Journey.

The rescheduling logic is simple, the AlternativeManager.dummyStrategy() just notifies the user's device in order to allow the user to modify the Journey. AlternativeManager.advancedStrategy() is an empty and not used method, it was inserted for an improvement of this logic in a future work.

5.3. Sequence Diagram

During the design phase, the sequence diagram was used to design the interactions between classes and to outline the entire execution flow.

The sequence diagram in Fig. 8 focuses on the JourneyGuardian class. This class executes in loop the following tasks. At first, the JourneyGuardian checks for new journeys in the database. If necessary, the inbound list will be updated. Then, the next starting journey and the next ending journey are checked. If a journey is beginning, the relative PromiseGuardian will be instantiated. If a journey is completed, the relative record will be removed from the outbound list.

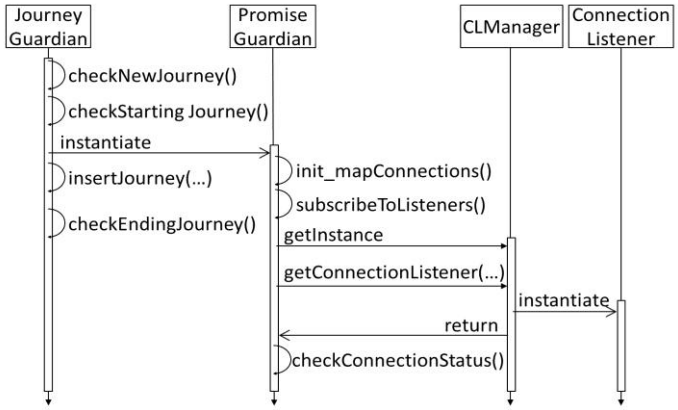

Fig. 8: Sequence diagram: focus on JourneyGuardian.

A PromiseGuardian, during its instantiation, will subscribe to one or more ConnectionListener thank to the CLManager. Then, it will wait for an update from the listeners.

The interaction between the PromiseGuardian and the ConnectionListeners is presented in the Fig. 7. A ConnectionListener periodically retrieves information about its transportation carrier. When an information changes, the ConnectionListener updates the database and calls the method PromiseGuardian.update(). The PromiseGuardian executes the check-promise algorithm on its connections. If the journey can not be completed, the method AlternativeManager.findAlternative() is called. AlternativeManager instantiates SendMSG from the Communication component and sends a message to the user.

5.4. Check-Promise Algorithm

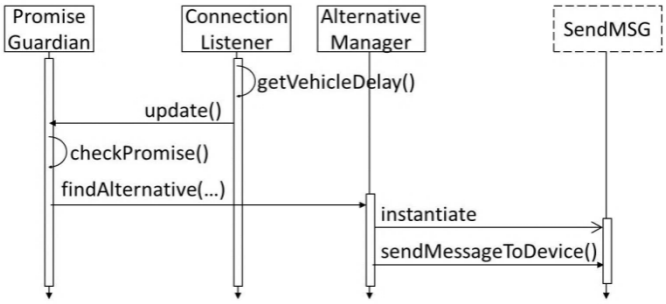

Fig. 7: Sequence diagram: focus on the interaction between PromiseGuardian and ConnectionListener.

The flowchart in Fig. 9 shows the check-promise algorithm at the base of the Compensation Engine. Two concepts need to be defined: 1) gap-time: the time interval between two consecutive connections. 2) safety-time: a minimal time interval that must be ensured to switch from a carrier to another.

The check-promise algorithm checks that the gap-time is greater or equal to the safety-time, taking into account the accumulated delays on the connections.

The algorithm's goal is then to check that the time interval between two connections is long enough to make the user able to catch the next carrier.

6. TEST

A demonstration version of the Compensation Engine was implemented and integrated with a prototype version of IRMA system. The goal was to validate the use case and the functionalities to provide a proof of the concept. The performances of the component were not considered as a crucial point for the prototype, however, they are meant to be tested and improved in further developments of the IRMA project.

The implementation of the Compensation Engine includes unit tests and logs. The unit tests ensured the correctness of every implemented class. The logs track the component's execution and catch every potential exception at run-time.

A set of scenarios has been individualized and the relative test cases have been outlined. Some of the test cases are presented in Table 2.

The component has successfully passed the test cases performed by the people of the developing team.

7. CONCLUSION

IRMA (Integrated Real-time Mobility Assistant) was introduced and one of its components was described: the Compensation Engine. IRMA is part of a research in progress whose aim is to support integrated mobility into smart cities. The Compensation Engine retrieves third part real-time transportation information to reduce the incidence of an issue on a transportation carrier. An overview on the real-time information in transportation has been provided. Requirement analysis has been performed and afterwards design, implementation and test of the component have been outlined.

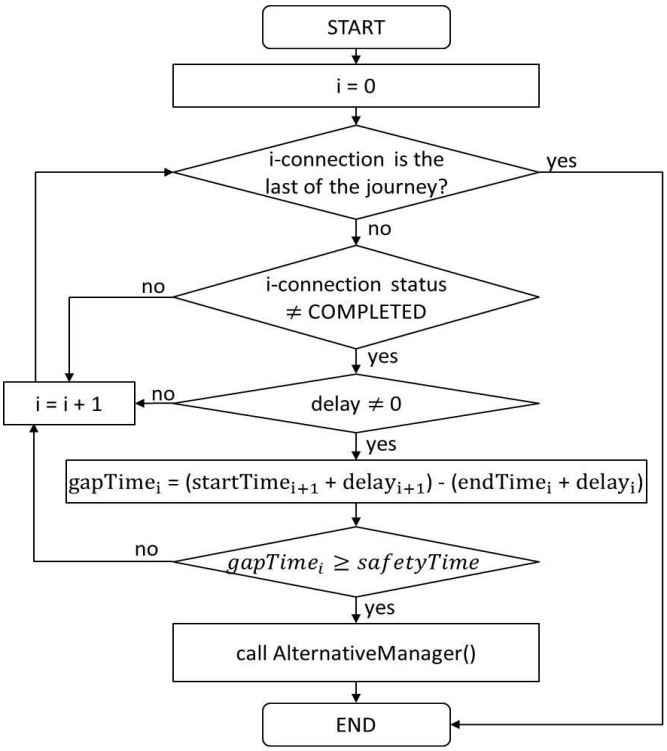

Fig. 9: Flowchart of the check-promise algorithm.

Future works should consider a further improvement of the Compensation Engine. A more advanced logic, for instance, could be implemented into the AlternativeManager to suggest the alternative journeys to the user. Another important point would be the improvement of the collaboration with transportation providers. This would increase the amount of real-time information and it would diversify the offer that is proposed to the end-user.

Morover, the rescheduling of a running journey involves ticket, money and booking issues. This kind of problems can be solved by an agreement with transportation companies. A good compromise, for example, would be to generate a single

Table 2. Test case table.

TEST	DESCRIPTION	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT
Scenario 1	No issues on the journey. The user is not alerted.	A journey is created. N° connections: 3 **Departure**: Bari (Italy) **Destination**: Pavia (Italy)	The system keeps monitoring the journey without alert the user.	It meets the expected output.
Scenario 2	Delay on a connection. The gap-time is smaller then the safety-time, so the system alerts the user. The user decides to not reschedule the journey.	A journey is created. N° connections: 2 **Departure**: Trapani (Italy) **Destination**: Messina (Italy)	The system spots the issue and alerts the user. The system keeps monitoring the journey before, during and after sending the alert.	It meets the expected output.
Scenario 3	Delay on a connection. The gap-time is smaller then the safety-time, so the system alerts the user. The user reschedules the journey.	A journey is created. N° connections: 2 **Departure**: Torino (Italy) **Destination**: Pavia (Italy)	The user successfully reschedules the journey. The system monitors the rescheduled journey.	It meets the expected output.

virtual ticket at the end of the journey. The virtual ticket would be charged to the user account and it would exclude the costs of the carriers not actually used by the user because of the rescheduling of the journey.

8. REFERENCES

[1] Barroero T., Telese F. Motta G., "Design of performance aware service systems," in The International Joint Conference on Service Sciences (IJCSS), Taiwan, 2011.

[2] Sacco D., Belloni A., You L. Motta G., "A system for green personal integrated mobility: a research in progress," in Service Operations and Logistics, and Informatics (SOLI), 2013 IEEE International Conference on, Dongguan, 2013.

[3] W. Souffriau, P. Vansteenwegen, "Tourist trip planning functionalities: state-of-the-art and future," Katholieke Universiteit Leuven, Belgio, 2010.

[4] S. Barbeau, P. Winters, N. Georggi, "Travel assistant device (tad) to aid transit riders with special needs," National Center for Transit Research at the Center for Urban Transportation Research, University of South Florida, 2008.

[5] C. Buhler, H. Heck, C. Radek, R. Wallbruch, J. Becker, C. Bohner-Degrell, "User feed-back in the development of an information system for public transport," Forschungsinstitut Technologie und Behinderung, Rhein-Main-Verkehrsverbund GmbH (RMV), Rhein-Main-Verkehrsverbund Servicegesellschaft mbH, Germania, 2010.

[6] S. Qi, W. Hai-yang, "Meta service in intelligent platform of virtual travel agency," School of Computer Science and Technology, Shandong University, China,.

[7] P. Moraitis, E. Petraki, N. I. Spanoudakis, "Providing advanced, personalised infomobility services using agent technology," Department of Computer Science, University of Cyprus, Cipro, 2003.

[8] C. Canali, M. Colajanni, R. Lancellotti, "Performance evolution of mobile web-based services," University of Modena, University of Reggio Emilia, 2009.

[9] L. Jun, D. Junping, W. Su, "Study on travel route intelligent navigation system based on webgis," School of Computer Science, Beijing Key of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunications, China, 2009.

[10] B. Pressl, C. Mader, M. Wieser, "User-specific web-based route planning," Institute of Navigation and Satellite Geodesy, Austria, 2010.

[11] H. Westerheim, B. Haugset, M. Natvig, "Developing a unified set of information covering," in 13th World Congress on Intelligent Transport Systems and Services, Norvegia, 2007.

[12] K. Dziekan, K. Kottenhoff, "Dynamic at-stop real-time information displays for public transport: effects on customers," Transportation & Logistics, Royal Institute of Technology (KTH), Svezia, 2006.

[13] C. Zhou, Y. Liu, Y. Tan, L. Liao, "Dynamic vehicle routing and scheduling with variable travel times in intelligent transportation system," College of Information System and Management, National University of Defense Technology, China, 2006.

[14] A. T. Baptista, E. Bouillet, F. Calabrese, O. Verscheure, "Towards building an uncertainty-aware personal journey planner," in 14th International IEEE Conference on Intelligent Transportation Systems, Washington, DC, USA, 2011.

[15] M.K. Natvig, H. Westerheim, "National multimodal travel information – a strategy based on stakeholder involvement and intelligent transportation system architecture," in 13th World Congress on Intelligent Transport Systems and Services, Norvegia, 2007.

[16] J. P. Dillenburg, O. Wolfson, P. C. Nelson, "The intelligent travel assistant," in The 5th International Conference on Intelligent Transportation Systems, Singapore, 2002.

[17] M. D. Hickman, N. H. M. Wilson, "Passenger travel time and path choice," University of California-Berkeley, Massachusetts Institute of Technology, USA,.

[18] J. Jariyasunant, D. B. Work, B. Kerkez, R. Sengupta, S. Glaser, A. Bayen, "Mobile transit trip planning with real–time data," Department of Civil Engineering, University of California, USA, 2009.

[19] S. J. Barbeau, N. L. Georggi, P. Winters, "Dynamic travel information personalized and delivered to your cell phone," National Center for Transit Research Center for Urban Transportation Research, University of South Florida, USA, 2011.

[20] S. Barbeau, N. L. Georggi, P. Winters, "Travel assistance device (tad) – deployment to transit agencies," National Center for Transit Research, Center for Urban Transportation Research, University of South Florida, USA, 2010.

[21] Magnus Penker Hans-Erik Eriksson, Business modeling with uml: business patterns at work.: John Wiley & Sons, 2000.

[22] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software., 1995.