Adopting a Middleware for Self-Adaptation in the Development of a Smart Travel System

L. Sabatucci, A. Cavaleri, M. Cossentino

ICAR-CNR, Palermo, Italy
{sabatucci, a.cavaleri, cossentino}@pa.icar.cnr.it

Abstract. A smart travel system is a complex distributed system acting as a tour operator for organizing holiday packages and supporting travelers on-the-run. A couple of key characteristics of such a system are the ability of self-configuring a set of heterogeneous services and self-adapting to unexpected circumstances. This paper reports an experience of developing a smart travel system by adopting MUSA, *a Middleware for User-driven Service Adaptation*. The prototype supports users in organizing their time by the specification of goals: this triggers the automatic composition and dynamic orchestration of touristic services. The chosen middleware has played a fundamental role by simplifying the development process thus to speed up the time-to-complete.

1 Introduction

Traditionally, the orchestration of services [4] uses approaches based on static models that provide little support for allowing self-configuration and adaptation of activities at run-time. Among these approaches, BPEL [14] – the main standard for implementing the orchestration of services– does not support advanced features for facing mutable and dynamic operative environment. Some of the well-known weakness are: 1) the flow of activities can not be changed when the execution context changes; 2) incorporating dynamic user preferences complicates the modeling activity; 3) revising the whole workflow is necessary every time a new services is introduced in the model; 4) even if it is possible to include service failures, the process is not robust enough to react to unexpected situations.

This work reports the development of a smart travel system as a customization of MUSA [11, 13, 2] (a Middleware for User-driven Service Adaptation). The aim is to aggregate heterogeneous services on-demand and to orchestrate touristic services in a dynamic, open and geographically distributed environment. Creating new travel experiences grounds on putting traveler at the center of the process. Firstly users may express travel preferences supported by a flexible language and a specific interface to convey her goals about: places to visit, activities to do and –in general– the kind of vacation. It is worth noting that this language deals with undefined (or better not completely defined) requirements, thus leaving users the choice to not specify something, and delegating

the system to propose alternative ways to complete travel itineraries. The second point is to allow the system to act as a local guide for traveler –running on personal device– by providing contextual information, monitoring the state of reserved resources and proposing viable alternatives on the run. Indeed, when traveler deviates from the planned route (willingly or not) the smart travel system will re-arrange services and resources to meet new emerging needs. So far, the developed prototype simulates this latter point.

The paper is organized as follows. Section 2 presents the main features of the smart travel system and motivates the choice of MUSA for the development. Section 3 focuses on two of the main characteristics provided by the middleware: self-configuration and self-adaptation. Therefore, the steps for implementing the system are briefly described in Section 4. Concluding remarks are drawn in Section 5.

2 The Smart Travel System

The smart travel system is a complex software designed to act as a tour operator, combining travel components on-demand to create a holiday package. The user may serve of the smart travel system to organize a vacation by specifying set of preferences about the kind of desired vacation including the geographic area of interest, places of interest, activities to perform, budget and so on. The system arranges alternative solutions to these specifications by composing a set of travel services: flights, transfers, hotels and tickets for museums, for the opera and for other local events. Moreover, when a user selects and pay for the package, the system works as a local assistant, providing contextual information, warning about train delays, checking flight cancellations, and re-organizing the vacation on the need. Such a software may be classified as a socio-technical system in which tourists and touristic services must be orchestrated in order to satisfy the former (improving the whole experience) and to maximize the use of the latter (increasing incomes for providers).

It is worth to underline the nature of the involved services. From the one hand services are *real* [5], i.e. each of them is composed of an electronic interface (e.g. the web protocol for booking a flight) and of an actual service the user will directly consume (e.g. flying). On the other hand, these services are heterogeneous –providing different benefits– and are geographically distributed in the territory.

Other significant features of the system are: i) user preferences: these must be flexible enough to allow configuring many aspects of the expected travel but also supporting user indecision, by allowing to specify only partial information; ii) self-configuration: the system must be able to check available touristic services and to arrange one o more solutions (travel packages) that address all the user preferences; iii) monitoring: as a local assistant, the system has to check that travel proceeds correctly; iv) self-adaptation: during the travel, the system must assure valid on-the-run alternatives, when something changes. It is possible that traveler changes his desires, or that a booked service is no more available. The

system must be able to re-configure the vacation, respecting new contextual constraints.

2.1 A Smart Travel Scenario

In this section we illustate a scenario for the smart travel system.
Herbert from Munich wants to travel to Sicily for a week with his family. By the smart_travel_system he plans a vacation of 7 days and set the following preferences: to stay 2-3 days in Palermo to look around the old city; at least 1 day at the beach to please his son; to watch a performance in the Greek theater in Syracuse; and finally at least 1 day in Catania city. He decides to leave the system free to suggest something for the remaining 2 or 3 days.
As response, the smart_travel_system suggests to flight to Palermo, where to stay 2 days, than 1 relaxing day at the Cefalù beach, 1 day in Catania, 2 days in Syracuse, (tickets for assist at the Greek tragedy are available only on day 5th), then 1 day at Agrigento and finally back to Palermo for the return flight.
Herbert confirms the travel plan and the smart_travel_system books the two flights, buys the ticket for the Greek tragedy, reserves hotels and buy tickets for transfers.

The whole package is shown on the left side of Figure 1. The second part of this scenario illustrates the ability of the system to adapt.
Herbert and his family are enjoying their vacation. They have been visiting Palermo for two days and decide to extend staying in the city by 1 day (variation to the plan).
Therefore the smart_travel_system proposes to variate the vacation, trying to maintain the tickets for the theater that are not reimbursable. The new plan comprises to stay another day in Palermo (addressing the new desire), and to skip the day at Agrigento and delaying the beach day (Cefalù) at the end. The new package is shown on the right side of Figure 1.
Herbert confirms the new travel plan, so the smart_travel_system cancels trains and hotels reservations that are no more necessary according the new plan.

We conducted a traditional study of the domain, followed by a requirement engineering analysis. A subset of the requirements are detailed in the next section.

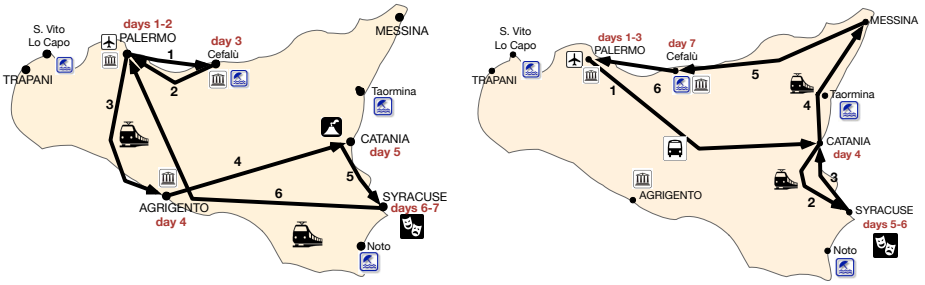

Fig. 1. On the left the original travel plan. On the right the new travel plan, adapted on the run, after a user preference to visit Palermo one day more.

3 Self-Composition and Adaptation

This section analyzes some of the Smart Travel System's requirements, in order to drive strategic choices for the implementation phase. In this context we present MUSA [2], a Middleware for User-driven Service Adaptation [1].

A summary of the main characteristics of the system to implement are listed and described hereafter.

- Heterogeneity. The same result may be addressed by properly composing different categories of services and resources. The design phase requires new design abstractions for describing services and the corresponding resources.
- Proactivity. The smart travel system holds a degree of freedom in taking decisions about how and when achieving user's goals. This requires a flexible description language to convey user's preferences that drive the system decision making.
- Dynamism. When the travel package is formed, the system must deal with a situation in which the operative context may change: services could fail, resources may be unavailable or even user's goals may change.
- Mobility. The system shall run on personal mobile devices with the aim of monitoring traveler's position and service failures. The system control loop must be flexible enough to support distributed data.
- Human Interaction. During the travel, users may either change preferences or react to warning about failures/changes. The system must support an active role of the user in the control loop. Moreover, the system must deal with user's changes of preferences.
- Awareness. The system shall constantly acquire knowledge about the state of services thus to raise the adaptation when necessary. Monitoring is a fundamental but costly activity. It must be optimized for the specific dynamic context.

The MUSA middleware offers a suitable infrastructure for implementing many of the high level features of the smart travel system. In the following we describe the main characteristics of the middleware, highlighting between brackets the impact on a category of requirements. The MUSA backbone is the decoupling of the dimensions of what and how. The GoalSPEC language [13]allows run-time specification of users requirements (*human interaction*) whereas the system adopts a descriptive logic for supporting the high-level reasoning on service semantic (*heterogeneity*).

Specific abstractions [12] are provided for representing what the system knows being able to do (*awareness*). The main function of MUSA is the ability of automatically associating system functions to user's goals thus to address the desired requirements (*proactivity*). This approach allows for implementing an architecture for self-configuration and self-healing (*dynamism*).

In MUSA the working key is configuring a solution as a response to the users request. It is implemented as a multi-agent system where entities are autonomous

[1] Website: http://aose.pa.icar.cnr.it/MUSA/.

and driven by a proactive goal-directed behavior. Agents guarantee knowledge acquisition, distributed coordination and robustness. When a solution becomes operative, the system executes special monitoring activities that capture the deviations between expected results and runtime performance, thus to adjust its behavior accordingly.

3.1 Main Concepts of the System

We informally introduce some important concepts used in the middleware. For the sake of clarity formal definitions and the reasoning framework are described with more details in [11, 12].

A ***Capability*** is a semantic wrapper for services that allows the developer to specify i) how to invoke the specified functionality (which data must be passed and which data will be returned) and ii) which effect is expected by executing the encapsulated application or service. The capability also has the advantage of being composable in order to address a complex result.

A ***User-Goal*** is "a desired *change* in a state of affair the user wants to achieve". The concept of goal is often used in the context of business process for representing enterprise strategic interests that motivate the execution of business processes [15].

A ***Configuration*** is a set of capabilities that address a set of user-goals. The main advantage of MUSA is the ability to self-configure, i.e. to automatically discover and aggregate capabilities to address dynamic user-goals.

A ***User-Norm*** is the description of a constraint the system must hold when addressing user-goals. It is described as a function in the domain of configurations that returns a boolean indicating admissible/non-admissible aggregation of capabilities.

A ***User-Metric*** is a user defined quality, associated to how the system will address user-goals. It is described as a function in the domain of configurations that returns a real number. This number may be used to compare two different aggregation of capabilities.

3.2 The Three Layered Architecture

The MUSA's core architecture for self-configuration and self-adaptation is composed of three communicating levels.

The uppermost layer of this architecture is the ***Goal Layer*** responsible for arranging system evolution. The user may specify the expected behavior of the system in terms of high level goals, norms and metrics. Whereas goals are dynamic entities specified through a language, norms and metrics are, so far, hard coded at design time. At run time, the goal injection phase allows free specification of user-goals; conversely norms and metrics are selected from preset lists. Injecting goals triggers a change in the system behavior. The user may be involved also as supervisor of the process, in order to take decisions about alternative cases of adaptation.

The second layer is the ***Capability Layer***, responsible of managing the strategic deliberation. The objective is selecting, aggregating and configuring available capabilities [11] as the response to i) self-evolution events (generated from the upper level) and ii) self-adaptive events (generated from the lower level). This activity may be very costly when the number of services (and their dependencies) grows. To reduce the complexity, the implemented algorithm reasons about services and environment through abstract data rather than concrete data [7, 8], so to be more affordable and scalable. The consequent output is one or more configurations of abstract services.

The third layer is the ***Service Layer***, responsible of translating from abstract services to concrete ones by adding the coordination logic necessary for enacting the corresponding business process. This layer provides atomic blocks of computation for acquiring and analyzing real data from the environment and producing the desired result. The proposed implementation adopts a distributed MAPE-K model [9, 1] that comprises: i) a *Monitor component* that acquires information from the environment, and updates the system knowledge accordingly; ii) an *Analyze component* that uses the knowledge to determine the need for adaptation with respect to expected goals and capabilities failure; iii) a *Plan component* that synchronizes the available capabilities according the goals to address and, finally, iv) an *Execute component* that modifies the environment by activating the appropriate capability.

4 Implementing the Smart Travel System

The middleware we have decided to adopt for implementing the advanced features of the smart travel system is a general purpose one: it has been recently used for implementing a document management system, a cloud mashup application process and a emergency management scenario. The strength is the easiness in customizing self-configuration and self-adaptation features for the specific domain of interest. The purpose of this section is to illustrate the steps for this customization.

4.1 Implementing Self-Configuration

In the smart travel system, self-configuration is mainly intended for automatically generating a number of alternative travel plans that are suitable for the user's goals. The challenge relies on the fact that travel services are heterogeneous, and they are not designed to be composed. Aggregating them means evaluating a great number of combinations. Figure 2 illustrates, in practice, the business process behind this activity.

1. The user will set his own preferences via either a web interface or a personal device.
2. The smart travel system discovers available services, and filters those may be useful for addressing the user request.

Fig. 2. Flow of activities for generating a travel plan.

3. The self-configuration algorithm arranges travel services, considering user's goals, uncertainty and domain constraints.
4. The result is a set of possible configurations (i.e. travel plans) to present via a user interface.
5. Finally, the user selects the travel plan he prefers.

The MUSA system allows to easily generate such behavior by associating user's preferences to user-goals/norms/metrics and by implementing travel services as capabilities.

The first step is a preliminary study of the domain for building the reference ontology. Ontologies provide a shared understanding of a domain of interest to support communication among human and computer agents. An ontology will be the base for matching design-time elements (norms, metrics and capabilities) and run-time elements (goals). An ontology for the travel domain has to cover i) geographical places (ex: Sicily, Palermo, Syracuse), ii) activities (visiting, swimming, watching opera), iii) transportation (by train, by car), and iv) immaterial qualities (cost, hotel rating). A plethora of web ontologies already exist, ready to be reused. In our system we adopted a subset of the Knublauch's OWL ontology [3] for a Semantic Web of tourism [2].

4.2 Capabilites for Touristic Services

A Capability is described as a self-contained autonomic entity. Whereas web-services are typically passive entities that act when receive the control [6], Capabilities are based on software agents able of self-managing, sensing the environment and making decisions on their own [11]. Despite their intrinsic complexity, developing a capability is not such a hard work, because MUSA provides basic facilities for self-awareness, self-adaptation, social interactions and self-configuration. The capability-developer has to specify two additional aspects.
The abstract description: a specification of why, when and how the related web-service may be used. This specification exploits a description-logic language based on the reference ontology. Typical fields are: pre/post conditions, information for simulating the aggregation, data input and data output.
The concrete implementation: contains the entry points for invoking the specific web-service and a reference to the protocol to use (examples are HTTP, SOAP or REST).

[2] Available via the Protegé website: protege.cim3.net/file/pub/ontologies/travel/travel.owl

In the context of the smart travel system we have defined 6 capabilities for touristic services: flight, hotel, train, bus, theater and museum. Each of them is able of gathering information and making/canceling reservation. An additional capability – visit_city – has been conceived for arranging time for freely visiting a place. During the visit, it has the special purpose of providing useful information via personal device and to monitor the traveler's position.

4.3 Norms and Metrics for Traveling Preferences

Besides the capabilities, norms and metrics are fundamental mechanism to increase user's flexibility in expressing their desires. Despite the intrinsic simplicity of the presented norms and metrics –more complex one could be added without changing the model– they represent a powerful instruments to dynamically refine the set of requirements.

A norm is a rule that describes the configurations of capabilities that produce undesired situations. For the smart travel system we have defined four norms the user may attach to goals:

- budget(MAX): for ensuring total expenses will not exceed a given budget.
- km(MAX): for ensuring the total number of kilometers will not exceed a max number.
- noTrain, noBus, a couple of norms for preventing respectively the use of either trains or buses for the transportation.

On the other side, metrics are high level elements that measure the quality associated to a configuration, thus to allow the system to compare and sort them. They are generally used in MUSA for enriching user's goals for specifying non-functional preferences. For the smart travel system we have defined the following metrics:

- artCityTour, that rewards spending time in famous cities of art.
- beachTour, that rewards spending time in seaside places.
- trainLovingTour, that rewards trips with continuous movements by train.
- wineFoodTour, that rewards visiting farmhouses and wineries.

4.4 A Web Application for Facilitating the Definition of Goals

MUSA provides a high-level language, GoalSPEC [13], to specify requirements as set of goals. For increasing flexibility and user-friendliness, it has been conceived as a controlled natural language. However such a language can not be directly used by smart travel users because it requires specific skills [10].

To this aim a web-based interface has been designed to mediate between users and their goals. It is based on a set of templates for specifying goals. Each template is a fixed structure in which some elements must be replaced by data coming from the forms.

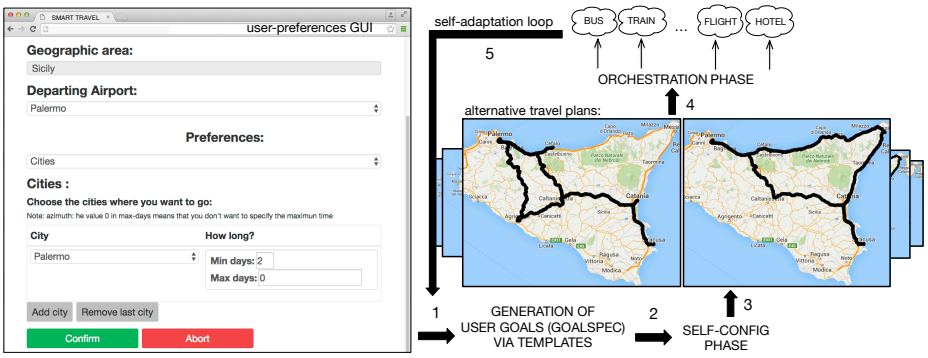

Fig. 3. An example of the MUSA approach to the case study of the smart travel

This approach reduces the flexibility of the language, but it allows an unskilled user to produce a valid set of goals without supervision. He also may select metrics and norms from pre-filled combo-boxes.

When the user confirms the whole set of preferences, then the web-portal generates the corresponding set of goals (step 1 of Figure 3) and injects them in the platform thus to activate the self-configuration phase (step 2 of Figure 3). The self-configuration phase consists in aggregating available services according to user preferences (goals, norms, metrics) and contextual availability of resources. The result is presented in geotagged maps where the travel plan is detailed step-by-step.

So far, we have not implemented the payment module necessary to buy each service of the travel package, but we have rather built a simulator engine that replaces the orchestration phase (step 2 of Figure 3). It simulates the progression of the stages of the vacation, by updating the current state as if the user is going to benefit of the acquired services. The evolution of the trip is represented directly in a geotagged map via an animation. During this simulation the user interface also allows to interactively produce events for adaptation. For instance, it is possible to mark a resource (e.g.: a train route) as unavailable. This triggers a new cycle of self-adaptation, as shown in step 5 of Figure 3, for proposing an alternative plan for the remaining part of the trip.

5 Conclusions

We presented a practical experience of customizing the general-purpose MUSA middleware in the context of the smart travel. This represents a novel approach for the development of smart and complex system where most of the complexity is inherited by reusing the underlying platform. After a traditional requirement engineering phase, the adoption of MUSA has facilitated the overall effort by limiting the development to three fundamental steps: 1) building (or reusing)

an ontology of the domain, 2) developing and deploying a repository of autonomous functions that incorporate the available services (capabilities) and finally 3) designing and developing the interface for supporting user participation in the loop. By following this approach, fundamental characteristics such as self-configuration, self-evolution and self-adaptation are automatically integrated and do not impact the time-to-complete.

References

1. Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. Springer, 2009.
2. M. Cossentino, C. Lodato, S. Lopes, and L. Sabatucci. MUSA: a middleware for user-driven service adaptation. In *Proceedings of the 16th Workshop "From Objects to Agents", Naples, Italy, June 17-19, 2015.*, pages 1–10, 2015.
3. H. Knublauch. Editing owl ontologies with protégé, 2004.
4. M. Laukkanen and H. Helin. Composing workflows of semantic web services. In *Extending Web Services Technologies*, pages 209–228. Springer, 2004.
5. A. Marchetto, C. D. Nguyen, C. Di Francescomarino, N. A. Qureshi, A. Perini, and P. Tonella. A design methodology for real services. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems*, pages 15–21. ACM, 2010.
6. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.
7. R. C. Moore. *Reasoning about knowledge and action.* PhD thesis, Massachusetts Institute of Technology, 1979.
8. A. Newell. The knowledge level. *Artificial intelligence*, 18(1):87–127, 1982.
9. T. Patikirikorala, A. Colman, J. Han, and L. Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 33–42, 2012.
10. K. Ryan, N. Maiden, and M. Glinz. If You Want Innovative RE, Never Ask the Users; A Formal Debate. In *18th IEEE RE*, pages 388–388. IEEE, 2010.
11. L. Sabatucci and M. Cossentino. From Means-End Analysis to Proactive Means-End Reasoning. In *Proceedings of 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Florence, Italy*, May 18-19 2015.
12. L. Sabatucci, C. Lodato, S. Lopes, and M. Cossentino. Highly customizable service composition and orchestration. In *Service Oriented and Cloud Computing*, volume 9306 of *LNCS*, pages 156–170. Springer, 2015.
13. L. Sabatucci, P. Ribino, C. Lodato, S. Lopes, and M. Cossentino. Goalspec: A goal specification language supporting adaptivity and evolution. In *Engineering Multi-Agent Systems*, pages 235–254. Springer, 2013.
14. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more.* Prentice Hall PTR, 2005.
15. E. Yu and J. Mylopoulos. Why goal-oriented requirements engineering. *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*, 15, 1998.