An Automated User-Centered Planning Framework for Decision Support in Environmental Early Warnings

Armando Ordonez[1], Vidal Alcázar[2], Daniel Borrajo[2], Paolo Falcarin[3], and Juan Carlos Corrales[1]

[1] Universidad de Cauca
[2] Universidad Carlos III de Madrid
[3] University of East London
jaordonez@unicauca.edu.co, valcazar@inf.uc3m.es, dborrajo@ia.uc3m.es, jcorral@unicauca.edu.co, falcarin@uel.ac.uk

Abstract. This paper presents the integration of automated planning in AUTO, a framework able to design plans of composed services for environmental early warning management. AUTO is based on three components: a request processing module that transforms natural language and context information into a planning instance; the automated planning and execution module based on an architecture for planning and execution, PELEA; and the Service Execution Environment for Web and Telco Services. The integration of a planning component provides two basic functionalities: the possibility of customizing the composition of services based on the preferences of the user and a middleware level that interfaces the execution of services in the environment.

Keywords: Automated Planning, Service composition

1 Introduction

Service Composition can be defined as the process of creating a composite process by combining available component services. It is used in situations where the request of a client cannot be satisfied by any single available service [1]. However, services may change, become available or unavailable and their number may also grow to unmanageable sizes, which makes impossible to manually generate a composition plan [2]. Previous works from both academia [3, 4] and industry [4] have revolved around this topic. However, few academic approaches include implementations in production scenarios, and few works from the industry are open or easy to extrapolate to similar cases. This paper gathers experience from previous works and presents the integration of automated planning as a

[4] Some examples from the industry are Zypr http://www.zypr.net/, SIRI http://www.apple.com/iphone/features/siri.html and Vlingo http://www.vlingo.com/

key component of AUTO (an AUTOmated user-centric Telco and Web services composition framework) in the environmental early warning domain.

The goal of the environmental early warning field is to create contingency plans that help resolve potentially harmful or dangerous situations based on the information gathered from sensors and the input of relevant users. An example of such a situation would be evacuating all the villages close to a river after detecting that its level has risen beyond regular measurements or upon the request of an observer. In this case a contingency plan would include actions to monitor the river, determine affected areas, warn the villagers and coordinate the logistics of the evacuation.

Since the participation of a human is required in critical scenarios, we assume that all the requests are triggered by users. Besides, in rural environments the access to a device able to send a request in the format specified by the system may be limited. This means that the system should be able to process natural language so it can be initiated through simple communication means like a phone call or an SMS. Thanks to the rather restrictive process of composition in this domain, the main procedures and their associated services can be described using semantic annotations by experts, which allows a natural language recognition technique to be implemented without imposing many restrictions to the user. Moreover, this makes the integration of a planning process a much easier task. Although this depends on the domain, most of the principles discussed here are applicable to similar scenarios.

The overall functioning of the architecture is as follows: the user request is received in natural language from a given device and processed to determine the goals and preferences of the user. At the same time, information obtained from the sensors may be added to the request depending on the context. Next, the request is translated dynamically into a planning instance modeled using the Planning Domain Definition Language (PDDL) [5]. Then, the PDDL formatted request is sent to the High Level Replanner module of the Planning, Learning and Execution Architecture (PELEA) [6] in order to obtain and execute a plan that represents the composition of services. Finally, the composed plan is executed in a Jain SLEE [5] environment for convergent services.

In a previous work, we presented AUTO as a framework that dealt with the environmental early warning domain receiving input from natural language request [7]. However, the deliberative process was based on hierarchical planning, which has several limitations over domain-independent planners that use PDDL. First, it is a domain-dependent approach, so if there is a change in the domain an expert must encode the modifications. This encoding is usually much more difficult to define than modifications at the domain level, because it mixes domain knowledge with control knowledge. Second, most HTN planners like SHOP2 [8] find it difficult to use different metrics and cannot easily deal with user preferences. Third, HTN planners prune parts of the search space for performance reasons, which means that often low-quality solutions may be obtained in unexpected problems. Lastly, HTN planning cannot benefit from newer planners that

[5] JAIN SLEE v1.1 http://jcp.org/en/jsr/detail?id=240

may surpass the current state of the art as research in the area advances. In this context, the main contributions of this paper are: the automatic transformation of natural language queries into a general language like PDDL; the encoding of user preferences in the environmental early warning domain; and the inclusion of automated planning techniques in AUTO, aiming to fully integrate it with PELEA.

2 The Environmental Management Domain

A sketch of an environmental management system is presented in Figure 1. The role of the environmental manager is to make decisions about the environmental alarms and crop management of a region. For this purpose he/she can request information from sensor networks deployed at several spots. The environmental manager can also use telecommunication (Telco for short) and Web services to process basic data and send information to both farmers and sensors. Available services often change dynamically and the resources may be limited.

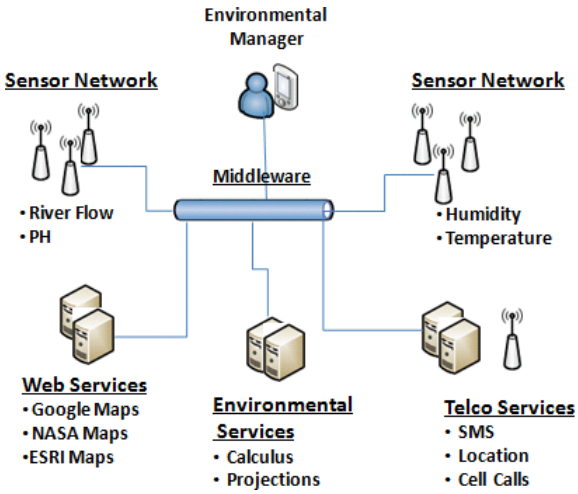

Fig. 1. Telecommunication and Web services interaction in Environmental Management Systems

As the environmental manager comes from fields like biology or agriculture, his/her technological background may be low. Furthermore, electronic devices in the area may be scarce. Thus, the preferred way to enter information to the system is by voice and in natural language. This way, users do not have to know how the system internally works and can make requests from regular mobile devices or landline phones. Commonly the user expresses his/her request informally; here are two examples:

- "I need to compute the hydrological balance of zone 1 and receive the resulting map to my cell phone"
- "the river flow of zone 2 is greater than 15% in average, emit an alarm to every farmer within a radius of 2 miles from the river and create an action map including emergency and rescue groups"

In the first case a composition of services could be: gathering information from the sensors in zone 1, using hydrological services from the Internet to process the sensed data, obtaining the map from Google maps, composing the final image and sending it to the user by MMS. A similar course of action would be done in the second case, making decisions about which maps are available or best suited to the case, the best way of warning the farmers (i.e. SMS, direct call, e-mail), etc. Several factors affect how the system must be designed:

- Usability for end users: simplicity of use and ubiquity forces the use of natural language recognition techniques. Also all the resulting processes must be readable so they can be supervised and modified by an expert due to the critical nature of the domain.
- Integration with the execution: Automated composition includes integration of Telco and Web functionalities, which requires specialized platforms.
- Reaction to change: planning systems deployed in dynamic environments must be able to react to potential changes during both the computation of the plans and the execution of the services.
- User preferences: multiple composition of services are often be possible, so the user may express his/her preferences in the request. Typical cases include minimizing the time of the execution of the composed services if it is an urgent request or minimizing the cost if there are no time constraints.

3 Automated Planning and Specification of the Domain

Automated planning is the task of finding a sequence of actions (plan) that leads to a state where some goals are achieved from a given initial state. Planning takes two inputs: a domain description and a problem description. Both are usually described using the standard language PDDL [5]. The domain file includes the types of the objects that may appear in the problems, a set of predicates/functions to be used to represent states and actions, and the actions set. Each problem file includes the initial state, the goals and, possibly, a metric.

If the domain and problem are specified in PDDL, almost all current domain-independent planners can be used to solve problems in our domain. Other paradigms related to domain-independent planning, like hierarchical planning, use representations on top of PDDL that tune the domain description towards some subset of solutions [8]. Usually, they are harder to generalize and reuse in other domains.

Here an overall description of the domain in PDDL will be given. First, the domain includes the definition of types of the objects that appear in the domain. As an example, these types might represent among others zones, users, communications means, etc.

```
(:types  zone user source map_step1 communication - object
         map_generation maptype - object)
```

Next, we have to define domain predicates and functions. As an example, we might define a predicate to express that we have already obtained some coordinates, or that we have already performed some sensing in a given zone. Predicates and functions have arguments that are typed.

```
(:predicates (coordinates_taken ?z - zone)
             (sensed  ?z - zone)
             (isolines ?z -zone) ...)

(:functions  (time) (cost)
             (messages ?m - communication)
             (access ?m - maptype)
             (source-cost ?m - source) ...)
```

Then, the domain actions model the different kinds of services. We describe now some of the actions on our domain. The *get_coordinates* action has as parameters: the user that performs the composition, the zone to be analyzed, the source of data (sensors, gps, database,...), and the type of communication to be used to gather coordinates (SMS, MMS, monitoring network,...). For this particular action there are no preconditions. Each source has associated different cost and time values, represented by the *source-cost* and *source-time* functions. We only show cost and time for simplicity.

```
(:action get_coordinates
    :parameters (?u - user ?z -zone ?m - source ?c - communication )
    :effect (and (coordinates_taken ?z) (sensed  ?z)
                 (increase (time) (source-time ?m))
                 (increase (cost) (source-cost ?m)) ...))
```

The *generate_map_from_vector_map* action has similar parameters: the user that perform the composition, the zone to be analyzed and the type of map to retrieve from internet (maps from NASA, free maps, maps by subscription). Some of the map services allow services a limited number of accesses. Therefore, each access decreases the number of available accesses. Consequently, when the number of services reaches zero it is necessary to update the subscription using the *update_subscription* action.

```
(:action generate_vector_map
    :parameters (?usr - user ?z - zone  ?mt - maptype)
    :precondition (and (sensed  ?z) (> (access ?mt) 0))
    :effect (and  (generated_map ?z)
                  (increase (time) (/ (time-map ?mt) 2))
                  (increase (cost) (/ (cost-map ?mt) 2))
                  (decrease (access ?mt) (access_quote))))
```

```
(:action update_subscription
    :parameters (?usr - user ?z - zone ?m1 - map_step1  ?mt - maptype)
    :precondition (and (mapa1 ?m1) (sensed  ?z) (= (access ?mt) 0))
    :effect (and  (increase (access ?mt) (acccess_update))
                  (increase (time) (update_time))
                  (increase (cost) (update_cost))))
```

The same logic is applied to model the *inform* action. This action is on charge of sending alarms and important information to specific zones according to the device, network coverage, warning level or user preferences. The number of messages or calling credit is limited and eventually it is required to top it up. Due to lack of space we will omit the description of the actions *inform* and *top_up*, although they are similar to *generate_map_from_vector_map* and *update_subscription*. Similarly, other actions are also needed in the domain but here they will be omitted for brevity.

4 Architecture of the Framework

The architecture of AUTO is depicted in Figure 2. The modules may be deployed in different machines so the processes load can be distributed. The access method can be both voice or text so AUTO can be accessed from a broad range of devices. In the literature, other alternatives have been proposed for user request treatment, such as Mashups [9] or service creation environments [10]. However, natural language offers a better mechanism for end users without expertise to express their requests [11].

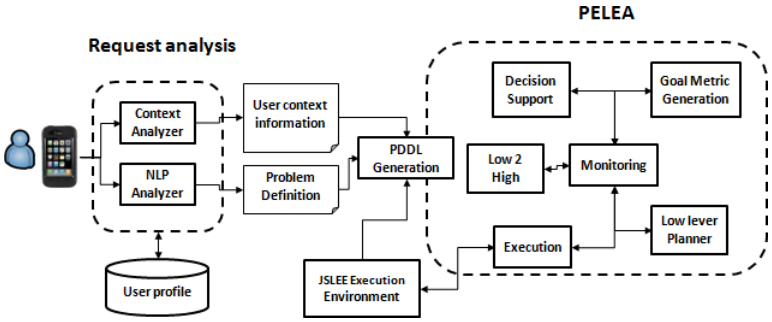

Fig. 2. Overall architecture of AUTO.

The Natural Language Analysis decomposes the request in constitutive parts and infers semantically which words are verbs (possible actions), nouns (possible parameters), control flow or context information, whose function will be described later. The Context Analyzer uses three dimensions: user preferences,

device capabilities and situational context that define parts of the initial state of the planning problem. Additional information may be added to the request from a database containing the profile of the user. The PDDL Generation module makes a translation from the processed request into a problem in PDDL. The data of the problem is obtained not only from the request but also from the information about the available services, which is why the PDDL Generation module requests information to the JSLEE Execution Environment at problem generation.

The automatically generated problem in PDDL is the input sent to PELEA, which performs the service composition using a domain-independent planner. The PDDL domain does not change, so PELEA uses the same domain definition for different service composition requests. In this work no monitoring is done, so currently the only task of PELEA is computing the plan and sending it to the JSLEE Execution Environment. In situations in which the status and characteristics of the services may change, though, PELEA would monitor the execution and replan or repair the current plan as needed.

AUTO uses a robust execution environment for telecommunications applications called Java Service Logic Execution Environment (JSLEE). The integration between PELEA and JSLEE is done in the execution module and allows the execution and the sensing of the state of the service composition. The execution module makes a dynamic association between the plan tasks and the JSLEE Service Building Blocks (SBB). SBBs are the basic components of the JSLEE architecture and call external Web services or Telco functionalities. The association between automated generated plans and SBB is done at run time, so changes in the environment can be easily dealt with.

5 Natural Language to PDDL

In AUTO, the transformation from natural language to PDDL is performed by a module composed by a set of underlying components. The functioning and implementation of this module has already been described elsewhere [7], so we will only provide here a rough description. The processing of the request in natural language is a set of sequential steps. First, the input is split into tokens, generating simple lexical units from complex sentences. Next, individual units are filtered and tagged according to their grammatical category. Additionally, each token is labeled as either "Control", "Functional" or "Situational" and classified according to three dimensions: device, user and situation. Next, information gathered from User Profile using the user's ID is added to the request. Finally, the request is translated into a planning instance from which a service composition is computed.

Every time a user makes a request a PDDL problem is generated. In this problem, the objects are defined, the predicates and functions initialized, the goals set and the metrics specified. The following is an example of how objects are defined and the functions are given values:

```
(:objects
    corpoamazonia - user
    farm1 farm2 farm3 - zone
    sms gps db - source
    sms mms voice - communication
    google nasa esri - maptype ...)

(:init
    (= (time) 0) (= (cost) 0) (= (topup_cost) 1)
    (= (source-time sms) 10) (= (source-time gps) 5)
    (= (source-cost sms) 5) (= (source-cost sms) 8) ...)
```

The goals are included as predicates that have to be achieved. The user preferences are encoded as a linear combination of the relevant criteria, namely *time* and *cost* because the standard way of encoding preferences in PDDL3, soft goals [5], offers no additional expressive power over regular metrics [12].

```
(:goal (and (informed farm1) (informed farm2) (informed farm3) ))
(:metric minimize (+ (* 1 (time)) (* 1 (cost)))))
```

6 Experimentation

To check the viability of the use of automated planning in AUTO some experimentation was done. First, we checked whether the system was able to scale up to requests that involve a high number of goals and services. An automatically generated instance was modified to increase the number of goals (farms that must be warned). The number of goals was increased from 20 to 400 in intervals of 50. Services, communication means, maptypes,... were left unmodified. The metric used for these problems was to minimize *time* plus *cost*. The planner used was CBP [13], a planner based on Metric-FF [14] with new heuristics designed to deal with numeric metrics.

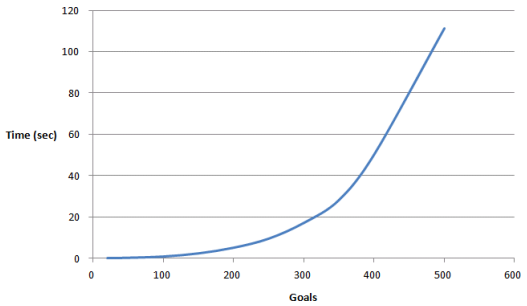

Fig. 3. Time needed to find composed services for an increasing number of goals.

Figure 3 shows how the time increases with the number of goals.In the experimentation, CBP behaves as expected, as the time increases exponentially. However, CBP is still capable of finding a plan with 500 goals in less than 2 minutes while minimizing the metric. This is a performance competitive with "ad-hoc" algorithms for services composition [15].

Another of the advantages of automatic planning is the possibility of dealing with metrics by just changing the definition of the metric in the problem file. The aim was to obtain a broad range of plans that give priority to minimizing *time* over *cost* and vice versa so they suit the preferences of the user. This was done by modifying the weights of the aforementioned linear combination of *time* and *cost*. More precisely, the metric is $\alpha t + \beta c$, in which t is *time* and c is *cost*. This means that the plans should be diverse and adapt themselves well to changes in the parameters α and β, similar to multi-objective optimization algorithms. Figure 4 shows a set of non-dominated plans obtained by modifying α and β in the metric. As in multi-objective optimization, it forms a pareto front. This means that CBP is sensible to the changes and the plans are diverse enough to offer a valuable alternative to the user when needed.

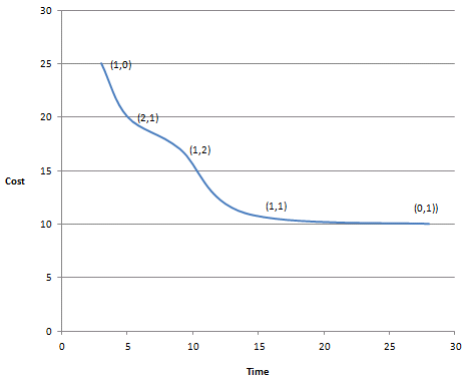

Fig. 4. Pareto front of solutions obtained by modifying α and β in the metric (in parenthesis).

7 Conclusion

In this paper we extend AUTO by integrating automated planning into an existing architecture as the deliberative process. This is a more general approach than the previously used technique that allows a greater flexibility both in terms of design and the implementation of the underlying algorithms. The main contribution is the automatic generation of planning instances in PDDL from natural language requests. Furthermore, we allowed the use of preferences modeling them as metrics of the planning instance. In the near future, we want to continue our

research developing mechanisms for the automation of planning domain generation. Further experimentation with different planners will be done too so an increase in performance can be achieved. Additionally, we are extending the preferences criteria in order to get a better personalized experience for the user. Finally, a complete integration with PELEA will be done, so features that it already offers, like monitoring and replanning, can be used by AUTO.

References

1. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of E-services that export their behavior. In: ICSOC. Volume 2910 of Lecture Notes in Computer Science., Springer (2003) 43–58
2. Oh, S.C., Lee, D., Kumara, S.R.T.: A comparative illustration of AI planning-based web services composition. SIGecom Exchanges **5**(5) (2006) 1–10
3. Yelmo, J.C., del Álamo, J.M., Trapero, R., Martín, Y.S.: A user-centric approach to service creation and delivery over next generation networks. Computer Communications **34**(2) (2011) 209–222
4. Hoffmann, J., Bertoli, P., Pistore, M.: Web service composition as planning, revisited: In between background theories and initial state uncertainty. In: AAAI, AAAI Press (2007) 1013–1018
5. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artif. Intell. **173**(5-6) (2009) 619–668
6. Guzmán, C., Alcázar, V., Prior, D., Onaindía, E., Borrajo, D., Fernández-Olivares, J., Quintero, E.: PELEA: a domain-independent architecture for planning, execution and learning. In: Scheduling and Planning Applications woRKshop, Freiburg, Germany. (2012)
7. Ordonez, A., Corrales, J.C., Falcarin, P.: Natural language processing based Services Composition for Environmental management. In: 2012 7th International Conference on System of Systems Engineering (SoSE 2012). (2012)
8. Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Nau, D.S., Wu, D., Yaman, F.: SHOP2: An HTN planning system. CoRR **abs/1106.4869** (2011)
9. Zhao, Z., Bhattarai, S., Liu, J., Crespi, N.: Mashup services to daily activities: end-user perspective in designing a consumer mashups. In: iiWAS, ACM (2011) 222–229
10. Laga, N., Bertin, E., Glitho, R.H., Crespi, N.: Widgets and composition mechanism for service creation by ordinary users. IEEE Communications Magazine **50**(3) (2012) 52–60
11. Lim, J., Lee, K.H.: Constructing composite web services from natural language requests. J. Web Sem. **8**(1) (2010) 1–13
12. Keyder, E., Geffner, H.: Soft goals can be compiled away. J. Artif. Intell. Res. (JAIR) **36** (2009) 547–556
13. Fuentetaja, R., Borrajo, D., López, C.L.: A look-ahead b&b search for cost-based planning. In: CAEPIA. Volume 5988 of Lecture Notes in Computer Science., Springer (2009) 201–211
14. Hoffmann, J.: The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. J. Artif. Intell. Res. (JAIR) **20** (2003) 291–341
15. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of executable web service compositions from BPEL4WS processes. In: WWW (Special interest tracks and posters), ACM (2005) 1186–1187