Designing a Tool for Configuring an Intelligent and Flexible Web-Based System

Diego Magro and Anna Goy

Dipartimento di Informatica, Università di Torino
Corso Svizzera 185; 10149 Torino; Italy
{magro,goy}@di.unito.it

Abstract. Several *intelligent* Web-based systems need complex reasoning mechanisms and problem solving capabilities in order to perform their tasks. These systems should protect their users against the complexity of their inference engine. Such a protection should be offered both to the final users and to the domain experts that instantiate the Web systems on the different particular domains. We claim that to this purpose an intermediate representation layer, with the role of filling the gap between the implementation-oriented view of the domain (needed by the reasoning module) and the human-oriented view of the same domain (suitable both for the final users and for the domain experts), is required. This intermediate layer also provides a representation which is independent from the specific reasoning approach adopted. In the paper we discuss these issues by referring to a specific example, i.e. the STAR system (a Web-based system which exploits a configuration engine to support the user in organizing a personalized tour) together with STAR-IT, a tool that supports the instantiation of STAR on different domains.

1 Introduction

People surfing the Web require an active and intelligent support, even in complex tasks (e.g., the Web should provide financial or medical advice, leasure and entertainment offers, travel plans and so on). For this reason, Web-based systems need to be equipped with complex reasoning mechanisms and problem solving capabilities.

To avoid ad hoc solutions, the exploitation of existing problem solvers is recommended, also to ensure a certain degree of generality and flexibility which guarantees the possibility to configure (in the following we will use the term *instantiate*) the system on different domains.

Current systems exploiting problem solving techniques (e.g., configuration systems [1, 2]) are typically designed for expert users, and they are based on an implementation-oriented representation of the domain. However, when these services are offered on the Web, the interaction needs of typical Web surfers, which include a large number of non-expert users requiring a natural and user-friendly interaction, cannot be ignored.

This scenario poses three main challenges to be faced when designing and developing an intelligent Web-based system that supports the user in a complex problem solving task:

- Non-expert users should not be exposed to the complexity of the knowledge representation and of the reasoning mechanisms used by the problem solver. In fact, both the knowledge representation and the reasoning mechanisms could be quite complex, expecially if the system exploits an existing problem solver (a configurator, a planner, or whatever), which was designed to be general with respect to the specific application domain.
- The system should not be committed to any specific approach, i.e., it should be designed in order to work with different problem solvers of the same type (e.g. different planners, possibly representing the domain knowledge in different ways), as well as with different types of problem solvers, since a same complex task may have more than one sound formalizations (e.g., some tasks may be formalized both as planning problems or as a scheduling problems). In other words, the reasoning engine should be wrapped within an independent module, in order to support the interoperability of the system with other modules, implementing different reasoning approaches.
- Another important aspect concerns the instantiation of the Web system on a particular domain and its maintenance. Such tasks are usually performed by a domain expert who is not aware of the reasoning techniques nor of the particular knowledge representation used by the problem solver. Thus, a tool supporting the domain expert in these tasks is needed.

We claim that this three goals require an intermediate representation layer, with the role of filling the gap between the implementation-oriented view of the domain, needed by the reasoning module, and the human-oriented view of the same domain, needed by the frontend of both the instantiation tool and the final system. This intermediate layer also provides a representation which is independent from the specific reasoning approach adopted. In other words, such a layer will guarantee both the tranparency of the reasoning mechanisms for the user (domain expert and final user) and the independency of the frontend of the system (intantiation tool and final system) from the specific reasoning techniques adopted.

The design of such a layer requires a domain ontology, based on the concepts used by people to reason about that domain, coupled with a translation module, that provides the mapping between these concepts and the representation needed by the actual reasoning engine.

In the following we will discuss in more detail this solution by referring to a specific example, i.e. STAR (Smart Tourist Agenda Recommender), a Web-based system which exploits a configuration engine to support the user in organizing a personalized tour in a given geographical area. In particular, we will discuss the main functionality of STAR-IT, a tool that supports the instantiation of STAR on different domains.

In Section 2 we introduce STAR, while in Section 3 we describe the istantiation tool; Section 4 concludes the paper.

2 STAR

In the travel and tourist domain different systems have been developed to support the users to plan long trips, involving flights scheduling and hotel reservation, or even to

select a set of tourist activities (e.g. [3–5]). However, in such systems the task of organizing a coherent agenda is totally left to the user.

The organization of a tourist agenda is a task that people are quite used to deal with. However, accomplishing such a task in a satisfactory way can be complex and time consuming, since it requires to access different information sources (tourist guides, the Web, local newspapers, etc.), to consider various kinds of constraints (the available time, the physical locations in which the different activities take place, and so on), and to mentally backtrack many times (e.g. when discovering that an interesting museum is closed, that two historical buildings are too far from each other to be visited in the same morning). Within our prototype STAR [6], we have tried to automate the task of organizing a tourist agenda by defining it as a configuration problem and exploiting a configuration engine based on the $\mathcal{FPC}$ framework [7], to find a solution that fulfils user requirements.

Intuitively, a configuration problem [8] is the task of assembling a (sub)set of predefined components in order to build a complex entity that meets a set of requirements. The component types are given a priori and no new type can be introduced during the configuration process. Usually, a set of constraints define the valid combinations of components. These constraints may restrict the types or the number of components allowed as parts of the complex entity, or they may express more complex relations among the components.

STAR's architecture currently includes three main modules: the Frontend, the Backend and the Knowledge Base. The Frontend consists of two components: the User Interface exploits XML/XSL technology to dynamically generate HTML Web pages, displayed on the browser; the Frontend Manager is a Java Servlet that takes the role of a mediator between the Backend and the interaction with the user.

The Knowledge Base contains a $\mathcal{FPC}$ declarative representation of a generic agenda as a complex (i.e. configurable) entity in which tourist items (e.g. buildings, restaurants, etc.) are the basic components and denote the corresponding activities (e.g. visits to buildings, lunch in a restaurant, etc.).

The Backend encompasses three submodules: a $\mathcal{FPC}$ configurator (the reasoning engine); the translator of the user's requirements into a set of $\mathcal{FPC}$ (input) constraints; and the Backend Manager, which handles the interaction with the Frontend and coordinates the activities of the other two Backend submodules. All these submodules are implemented in Java.

STAR's reasoning engine takes as input both the knowledge base and the input constraints (representing the user requirements) in order to configure a suitable agenda to suggest to the tourist (i.e. the final user of the system).

3 STAR-IT

STAR-IT is a tool supporting the domain expert in instantiating STAR on particular domains. Two STAR applications may differ with respect to various dimensions, which represent the degrees of freedom of the instantiation activity. Here, we focus on those dimensions that are most closely related to the problem solving capability: the descrip-

tion of the general structure of an agenda and the definition of the properties that the final user can impose as requirements.

We will show how the domain expert can instantiate a system like STAR without being a knowledge engineer, and in a way that is totally independent from the specific inference engine actually exploited by the system. Figure 1 depicts the instantiation process.

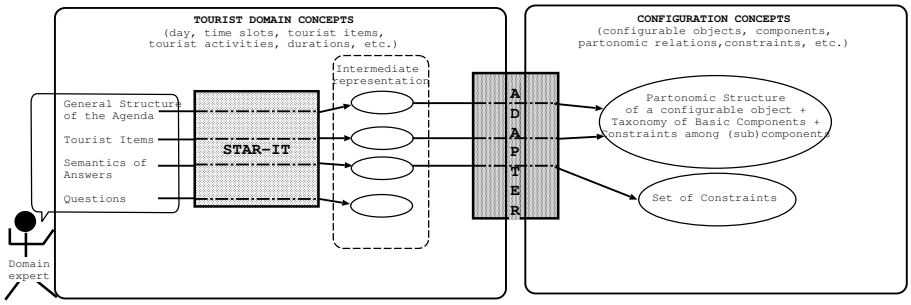

Fig. 1. Instantiation Process

Thanks to STAR-IT, the domain expert can describe the general structure of an agenda and its possible content by means of common concepts, which include: the time slots of the day (e.g. morning, afternoon, etc.); the tourist items (e.g. buildings, restaurants, events, etc.) that are involved in the tourist activities (e.g. visiting buildings, eating, attending events, etc.); the possible durations of the activities; the daily opening hours and the actual tourist attractions availability in particular days; the distance between the physical locations in which the activities take place.

STAR-IT produces an intermediate representation based on these concepts, which expresses the user-oriented view of the domain and is independent from the specific reasoner. A module, called Adapter, plays the role of a bridge between the problem solver and the rest of the system. Such module takes as input the intermediate representation and it produces a knowledge base in the problem solver format (therefore the Adapter depends on the particular problem solver). In the current prototipe, a $\mathcal{FPC}$ general configurator plays the role of the problem solver and the Adapter translates the intermediate description of the general agenda into a $\mathcal{FPC}$ knowledge base. Such a knowledge base is expressed by means of the usual configuration concepts: configurable and basic components and subcomponents, partonomic relations, constraints among (sub)components and so on. The domain expert can be unaware both of the configuration ontology and of the ways the configuration concepts are actually represented within the configurator.

In the following we show a fragment of the internal representation, based on an XML language, adopted by STAR-IT at the intermediate layer:

```
<DAILY_AGENDA>
  <TIME_SLOTS>
    <TIME_SLOT>
```

```
      <TS_NAME>Morning</TS_NAME>
      <TS_MAX_DURATION>12</TS_MAX_DURATION>
      <TS_PREFERRED_DURATION>10</TS_PREFERRED_DURATION>
      <ACTIVITIES>
        <ACTIVITY>
          <A_NAME>havingBreakfast</A_NAME>
          <A_MIN_NUM>0<A_MIN_NUM> <A_MAX_NUM>1</A_MAX_NUM>
          <A_ITEM_TYPE>Bar</A_ITEM_TYPE>
          ...
        </ACTIVITY>
        <ACTIVITY>
          <A_NAME>visitingAttractions</A_NAME>
          <A_MIN_NUM>0<A_MIN_NUM> <A_MAX_NUM>10</A_MAX_NUM>
          <A_ITEM_TYPE>TouristAttraction</A_ITEM_TYPE>
          ...
        </ACTIVITY>
        <ACTIVITY> <A_NAME>havingLunch</A_NAME> ... </ACTIVITY>
        <ACTIVITY> <A_NAME>doingShopping</A_NAME> ... </ACTIVITY>
        <ACTIVITY> <A_NAME>attendingEvent</A_NAME> ... </ACTIVITY>
      </ACTIVITIES>
    </TIME_SLOT>
    <TIME_SLOT> <TS_NAME>Afternoon</TS_NAME> ... </TIME_SLOT>
    <TIME_SLOT> <TS_NAME>Evening</TS_NAME> ... </TIME_SLOT>
  </TIME_SLOTS>
  ...
</DAILY_AGENDA>
```

The part enclosed in the DAILY_AGENDA tags describes the general structure of a one-day agenda. This fragment, for instance, says that a day is partitioned in three time slots (Morning, Afternoon and Evening). The morning should better not exceed 10 time units (the domain expert can define her own time unit; in this example, each time unit roughly corresponds to half an hour) and it must not be longer than 12 time units. The set of all the possible activities is specified for each time slot of the day; e.g. in the morning a tourist can have breakfast, visit up to 10 tourist attractions and so on. The tourist items that can be involved in the activities are described in the section enclosed between the AVAILABLE_TOURIST_ITEMS tags and they are organized into a taxonomy, as shown in the following:

```
<AVAILABLE_TOURIST_ITEMS>
  <CLASSES>
    <CLASS>
      <C_NAME>Bar</C_NAME> <SUBCLASS_OF>FoodPlace</SUBCLASS_OF>
      ...
    </CLASS>
    ...
  </CLASSES>
  <ELEMENTS>
    <ELEMENT>
      <E_NAME>HappyBar</E_NAME> <IS_A>Bar</IS_A>
```

```
    </ELEMENT>
  <ELEMENTS>
<AVAILABLE_TOURIST_ITEMS>
```

The Adapter translates this user-oriented representation into the representation required by the configurator. In particular, in the current prototype, it builds a $\mathcal{FPC}$ partonomy representing the configurable daily agenda which, in this example, has three configurable direct components corresponding to the morning, the afternoon and the evening. These components have a set of partonomic roles representing the allowed activities and that can be filled with the tourist items, which are the basic components of the configuration. Moreover, a set of $\mathcal{FPC}$ constraints restrict the set of valid configurations; e.g. the maximum and the preferred duration of the morning is coded by two weighted constraints (one with $k = 10$ and $w \neq \infty$ and the other with $k = 12$ and $w = \infty$), as follows:

$$\Sigma(\{\langle havingBreakfast, duration\rangle, \langle visitingAttractions, duration\rangle, \langle havingLunch, duration\rangle, \langle doingShopping, duration\rangle, \langle attendingEvent, duration\rangle\}) \leq k, weight = w$$

STAR-IT supports the domain expert also in defining the possible (final) user's requirements on a tourist agenda, i.e. which properties of the agenda are influenced by user's preferences and needs (e.g. having/not having breakfast, visiting historical museums, etc.), expressed in an initial interview (Web form) that STAR proposes to the user.

STAR-IT enables the domain expert to list the set of questions that STAR will ask to the final user, along with a finite set of possible answers. Each answer represents a user requirement whose meaning is specified during the instantiation phase.

To express the semantics of each answer, the domain expert has to identify which time slots, which activities, and which kinds of tourist items the requirement refers to; moreover, she has to specify the actual content of the requirement. For instance, in the following we show the intermediate representation of a requirement stating that the tourist is interested to visit many baroque buildings either in the morning or in the afternoon.

```
<REQUIREMENT>
  <TIME_SLOTS>
    <TIME_SLOT>Morning</TIME_SLOT> <TIME_SLOT>Afternoon</TIME_SLOT>
  </TIME_SLOTS>
  <ACTIVITIES>
    <ACTIVITY>visitingAttractions</ACTIVITY>
  </ACTIVITIES>
  <TOURIST_ITEMS>
    <ITEM_TYPE>
      <CLASS>Building</CLASS>
      <RESTRICTIONS>
        <RESTRICTION>
          <ATTRIBUTE>Style</ATTRIBUTE> <VALUE>Baroque</VALUE>
        <RESTRICTION>
```

```
      </RESTRICTIONS>
    </ITEM_TYPE>
  </TOURIST_ITEMS>
  <CONTENT>many</CONTENT>
</REQUIREMENT>
```

In the current prototype, the intermediate representation of each requirement is translated into a set of $\mathcal{FPC}$ constraints.

4 Conclusions

In this paper we have presented STAR-IT, a tool supporting the instantiation of STAR (a Web-based system supporting the user in organizing a personalized tourist agenda) on new domains, and we have discussed how STAR-IT faces two main challenges: (i) it hides the technological complexity to the user (i.e. the domain expert), who needs not to be aware of the actual inference mechanisms underlying the system; (ii) it is not committed to a specific reasoning module, therefore modules based on different inference techniques can be easily plugged in the system. In particular, we have shown how an intermediate representation layer, representing the human-oriented view of the domain, can be exploited to achieve these goals.

A first prototype of STAR-IT has been implemented (as a Web-based Java application) and it has been used to instantiate a STAR application on the city of Turin. The current prototype is still only a research prototype and feedback from real users is needed in order to enhance the usability of the tool and to refine the approach.

References

1. Mailharro, D.: A classification and constraint-based framework for configuration. AI EDAM **12** (1998) 383–397
2. Fleischanderl, G., Friedrich, G.E., Haselböck, A., Schreiner, H., Stumptner, M.: Configuring large systems using generative constraint satisfaction. IEEE Intelligent Systems (1998) 59–68
3. Torrens, M., Faltings, B., Pu, P.: *SmartClients*: Constraint satisfaction as a paradigm for scaleable intelligent information systems. Int. Journal of Constraints **7** (2002) 49–69
4. C.Knoblock: Building software agents for planning, monitoring, and optimizing travel. In: Proc. of ENTER 2004. (2004)
5. Ricci, F., Arslan, B., Mirzadeh, N., Venturini, A.: Itr: a case-based travel advisory system. In: Proc. of ECCBR 2002. (2002) 613–627
6. Goy, A., Magro, D.: Dynamic configuration of a personalized tourist agenda. In: Proc. of IADIS Int. Conf. WWW/Internet 2004. (2004) 619–626
7. Magro, D., Torasso, P.: Decomposition strategies for configuration problems. AIEDAM, Special Issue on Configuration **17** (2003) 51–73
8. Sabin, D., Weigel, R.: Product configuration frameworks - a survey. IEEE Intelligent Systems (1998) 42–49

5 REQUIREMENTS OF THE REVIEWERS AND CHANGES TO THE PAPER

REVIEWER 1:

- **Requirement:** Section 2 - A figure, namely a block diagram of the STAR prototype is required. The reasoning mechanism of STAR must be supported by a simple but relevant example.
  Change: A description of STAR'S architecture has been added.
  Due to space restriction, we could insert neither a figure nor an example. However, references have been made to papers where both figures and examples can be found.
- **Requirement:** Section 3 - A figure, namely a block diagram of the STAR -IT prototype is required. Also a simplified figure of the general structure of the agenda
  Change: A figure depicting the instantiation process has been added
- **Requirement:** Section 4 Conclusions - a more accurate explanation of the actual development stage of STAR-IT is required.
  Change: a first STAR-IT prototype has been implemented and used to build a STAR instantiation. This has been stated in the Conclusions Section

REVIEWER 2:

- **Requirement:** In my view the three fundamental assertions from the abstract belong in the introduction - in fact they are repeated here so are not needed in the abstract.
  Change: the abstract has been re-written
- **Requirement**: In any case you should make it clear that these assertions are not new, but are similar to those on which expert systems are based.
  Change: none. Actually, the similarities and the differences would deserve a deeper discussion, which is out of the scope of the paper
- **Requirement**: You should say what stage of development STAR is at. By the conclusions it seems as if planning is at an advanced stage but little concrete technical development has been achieved.
  Change: a first STAR-IT prototype has been implemented and used to build a STAR instantiation. This has been stated in the Conclusions Section
- **Requirement**: It is nopt stated how STAR is going to be implmented: Java, ASP, PHP, server side, browser side? Will it be based on a SQL database? How will state information be preserved? etc More detail of this sort is required.
  Change: Section 2: information about the technologies the various modules of STAR are based on has been added
- **Requirement**: Avoid abbreviations such as w.r.t. **Change**: occurences of "w.r.t." abbreviation eliminated
- **Requirement**: I would be happy for this paper to be published if it was made clear that it was a "work in progress" paper and also if plans for future development were more firmly described. Concrete ideas about the way in which the challenging aspects will be approached are alos required.
  Change: a research prototype has been implemented. An evaluation with domain experts is needed in order to enhance the usability of the tool and to refine the approach. This is stated in the Conclusions Section