From Unstructured Web Knowledge to Plan Descriptions

Andrea Addis and Daniel Borrajo

Abstract Automated Planning (AP) is an AI field whose goal is to automatically generate sequence of actions that solve problems. One of the main difficulties in its extensive use in real-world application lies in the fact that it requires the careful and error-prone process of defining a declarative domain model. This is usually performed by planning experts who should know about both the domain in hand, and the planning techniques (including sometimes the inners of these techniques or the tools that implement them). In order planning to be widely used, this process should be performed by non-planning experts. On the other hand, in many domains there are plenty of electronic documents (including the Web) that describe processes or plans in a semi-structured way. These descriptions mix natural language and certain templates for that specific domain. One such examples is the *www.WikiHow.com* web site that includes plans in many domains, all plans described through a set of common templates. In this work, we present a suite of tools that automatically extract knowledge from those unstructured descriptions of plans to be used for diverse planning applications.

1 Introduction

This paper tries to build a gap between two fields: automatically extracting information from the Web and AP. As for information extraction, we are assisting a continuous growth in the availability of electronically stored information. In particular, the Web offers a massive amount of data coming from different and heteroge-

Andrea Addis
Department of Electrical and Electronic Engineering, University of Cagliari, Italy,
e-mail: addis@diee.unica.it

Daniel Borrajo
Department of Computer Science, University Carlos III of Madrid, Spain
e-mail: dborrajo@ia.uc3m.es

neous sources. Most of it is in an unstructured format as natural language (blogs, newspapers). However, the new, most overshadowing and noteworthy web information sources are being developed according to the collaborative web paradigm, also known as Web 2.0 [17]. It represents a paradigm shift in the way users approach the web. Users (also called prosumers) are no longer passive consumers of published content, but become involved, implicitly and explicitly, as they cooperate by providing their own content in an *architecture of participation* [6]. Such knowledge repositories are semi-structured mixing some structure with natural language descriptions and predefined ontologies as in Wikipedia,[1] eBay,[2] and Amazon,[3] or IMDb.[4] In Wikipedia, a style template has to be filled in for each category belonging to a hierarchical structure of topics. In commercial sites as eBay, the online auction and shopping website, Amazon, an American-based multinational electronic commerce company website, a predefined list of mandatory attributes within its category is provided for each article. Also a more specialized knowledge base, IMDb, the Internet Movie Database, provides a list of standard attributes such as authors, director, or cast for each stored movie. In the spirit of Wikipedia, WikiHow[5] is a wiki-based web site with an extensive database of how-to guides. They are provided in a standard format (template) consisting of a summary, followed by needed tools (if any), steps to complete the activity, along with tips, warnings, required items, links to related how-to articles, and a section for sources and citations. Nowadays, thanks to advanced publishing tools the semi-structured knowledge base is more common, but not yet dominant. Therefore, it is becoming a primary issue to support applications that require structured knowledge to be able to reason (as in the form of ontologies), in handling with this enormous and widespread amount of web information. To this aim, many automated systems have been developed that are able to retrieve information from the Internet [1, 7], and to select and organize the content deemed relevant for users [3, 4]. Furthermore there has been some work on ontology learning [25, 16] pointing out how it is possible to solve the problem concerning the lack of structure of which the web often suffers. Thus, the structured format of the extracted knowledge is usually in the form of hierarchies of concepts (see for example the DMOZ project[6]) and this can help on developing many different kinds of web-based applications, such as specialized or general purpose search engines, or web directories. Other applications need information in the form of individual actions more than structured hierarchies of concepts, or in the form of plans.

On the other hand, as for AP, making it a widely used technology requires its usage by non-planning experts. Currently, this is quite far from being a reality. So, there is a need for techniques and tools that either allow an interaction with domain experts in their usual language, or automatically (or semi-automatically) acquire

[1] http://www.Wikipedia.org

[2] http://www.eBay.com

[3] http://www.Amazon.com

[4] http://www.IMDb.com

[5] http://www.WikiHow.com

[6] http://www.dmoz.org

knowledge from current sources of plans and actions described in semi-structured or unstructured formats. In the first case, there has been some work on knowledge acquisition tools for planning as GIPO [20], techniques for domain models acquisition [13, 22, 24], or tools that integrate planning and machine learning techniques [26, 11]. In the second case, there has been very little work on building plans from human generated plans or actions models described in semi-structured or unstructured formats, as filling natural language descriptions on templates. Another field that could also benefit from this automatic (or semi-automatic) acquisition of plans is the area of goals/activities/plan recognition [21], where most of its work assumes the existence of plan libraries that are manually coded. Examples are in the health environment [19], helping aging persons to perform their daily activities [18], or assisting a user on performing bureaucratic or tourism related actions [8].

In this chapter, we want to describe some work to deal with the lack of tools to automatically build plans and action models from semi-structured information, and the existence of this kind of knowledge in the Web, as is the case of WikiHow. We believe this is an important step toward a massive usage of planning technology by users in that they can share plans as they are doing now through WikiHow in natural language, and then automatic tools build planning technology on top of those plans. This applies also to other domains as workflow applications, where most big organizations have written processes [14, 5], or hospitals, where many standard procedures are described also in semi-structured formats. Also, we will encourage research in this topic by suggesting potential relevant tools to be built on top of our research for improving the construction of planning and plan recognition tools. This work extends and revises the work by Addis et al.[2] on recovering plans from the web. The main extension consists on adding a more sophisticated tool (i.e., *the Plan Builder*) to translate the article into a final structured plan.

The remainder of the paper is organized as follows: first, we provide an introduction to AP. Subsequently, the proposed architecture is depicted, and the constituting subsystems are separately analyzed. Then, the experiments and their results are presented and evaluated. Finally, we draw some conclusions and outline future research.

2 Automated Planning

AP is the AI field that provides techniques and tools to solve problems (usually combinatorial) that require as output an ordered set of actions. The inputs to planners are: a domain model, that describes, among other components, the available actions in a given domain; and a problem instance, that describes, among other components, the initial state and the goals to be achieved. Both input files are currently represented using a declarative standard language called PDDL (Planning Domain Description Language), that has evolved from the first version in 1998, PDDL1.0, to PDDL3.1

used in the last International Planning Competition.[7] Given the inputs (domain and problem descriptions), planners return an ordered set of actions (usually in the form of a sequence), that is called a plan.

As an example, Figure 1 shows an example of an action definition, taking an image, in a domain that specifies how a set of satellites should take images from space. As it can be seen, the language is based on predicate logic, and also allows numerical computations. In this work, our goal is to generate some declarative representations of the domains. Then, there are currently tools that are able to obtain PDDL models from the kind of output that we generate within this work. Figure 2 shows an example of an output plan in this domain.

```
(:action take_image
  :parameters (?s - satellite ?d - direction ?i - instrument ?m - mode)
  :precondition (and (calibrated ?i)
                     (on_board ?i ?s)
                     (supports ?i ?m)
                     (power_on ?i)
                     (pointing ?s ?d)
                     (power_on ?i)
                     (>= (data_capacity ?s) (data ?d ?m)))
  :effect (and (decrease (data_capacity ?s) (data ?d ?m))
               (have_image ?d ?m)
               (increase (data-stored) (data ?d ?m))))
```

Fig. 1 Example of action definition in PDDL.

```
Solution:
    0: (SWITCH_ON INSTRUMENT0 SATELLITE0) [1.000]
    1: (TURN_TO SATELLITE0 GROUNDSTATION2 PHENOMENON6) [1.000]
    2: (CALIBRATE SATELLITE0 INSTRUMENT0 GROUNDSTATION2) [1.000]
    3: (TURN_TO SATELLITE0 PHENOMENON4 GROUNDSTATION2) [1.000]
    4: (TAKE_IMAGE SATELLITE0 PHENOMENON4 INSTRUMENT0 THERMOGRAPH0) [1.000]
    5: (TURN_TO SATELLITE0 STAR5 PHENOMENON4) [1.000]
    6: (TAKE_IMAGE SATELLITE0 STAR5 INSTRUMENT0 THERMOGRAPH0) [1.000]
    7: (TURN_TO SATELLITE0 PHENOMENON6 STAR5) [1.000]
    8: (TAKE_IMAGE SATELLITE0 PHENOMENON6 INSTRUMENT0 THERMOGRAPH0) [1.000]
```

Fig. 2 Example of resulting plan in the Satellite domain.

[7] http://ipc.informatik.uni-freiburg.de/

3 From Unstructured Plans to Action and Plan Models

In this section we describe the Plan Acquisition Architecture (PAA) that performs the acquisition of plans and actions from semi-structured information. Then, we will describe the information source that we have used in this paper for showing how it works.

3.1 The PAA

The set of tools we have built are components of a modular structured architecture, which automatically browses some specific category from the ones represented in WikiHow, analyzes individual plans in those web pages, and generates structured representations of the plans described in natural language in those pages. This work intends to fill the lack of automatic tools to build plans using the semi-structured knowledge, more and more present over the web, which is similar to what is currently done in the Semantic Web, Information Retrieval or Ontology Learning fields.

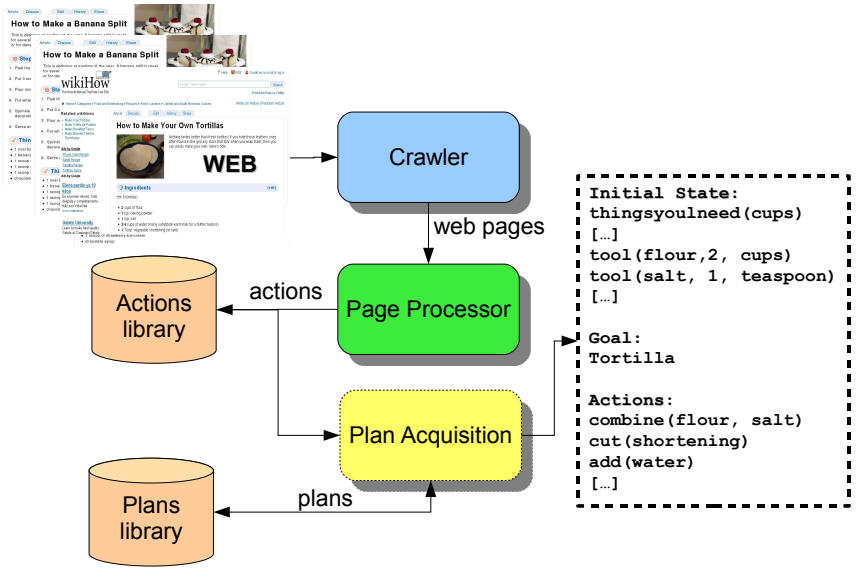

Fig. 3 PAA at a glance.

Figure 3 highlights how the different subsystems, each wrapping different technologies, retrieve and transform the semi-structured information provided by web articles into a semantically overt, declarative structured output. The output contains labeled and recognizable objects (e.g., work tools, ingredients), actions (e.g., cut,

peel, fry), and plans (e.g., sequences of instantiated actions), so that they may be reused for planning purposes.

The *Crawler* subsystem is devoted to crawl and store pages and category sections of a web site. The *Page Processor* subsystem is currently the core of PAA. It is aimed at analyzing web pages, storing only the relevant semi-structured contents into an action library after performing an initial pre-processing. In particular (i) the goal, (ii) the initial state (in terms of required tools), (iii) the actions, (iv) tips (to be exploited as heuristic hints), (v) warnings (to be exploited as plan build constraints), and (vi) articles related to the selected page/plan are stored for each input page. The *Plan Acquisition* subsystem processes this information to extract plans. The tools belonging to each subsystem, depicted in Figure 4, will be separately analyzed.

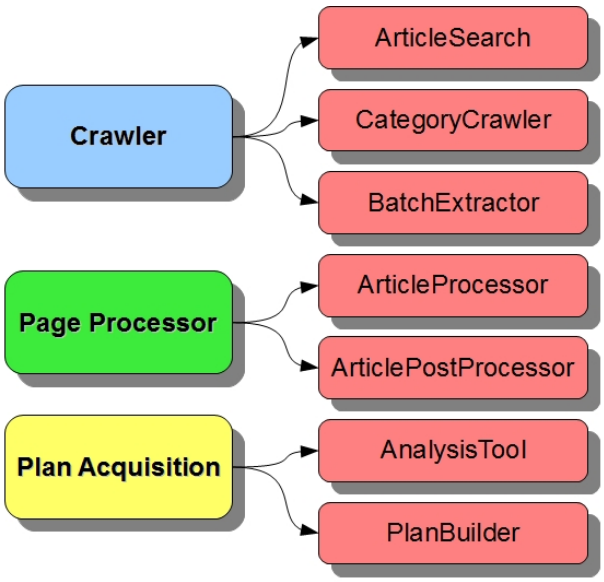

Fig. 4 Subsystems of PAA and corresponding tools.

3.2 WikiHow: a Knowledge Source for Extracting Plans

WikiHow is a collaborative writing project aimed at building the world's largest and highest quality how-to manual, and it has been used as a benchmark to test the PAA ability on structuring information in the form of plans. WikiHow currently contains more than 70,000 articles written, edited, and maintained primarily by volunteers. Each article contains the necessary tools and describes the sequence of actions required to reach the goal the page is concerned with. As an example, let us take a look

at the page *Make Your Own Tortillas* [8], reported in Figure 5, to better understand how the different subsystems parse its different sections. The relevant ground items that can be contained in each WikiHow web page are actions, tools, and related web pages (relatedwps), represented as A, T, and WP respectively. The Page Processor is the subsystem entrusted to process the different sections of the page. The name of the web page is identified by a `<div id=NAME>` HTML tag. Since the name of the how-to web page corresponds to the instructions to achieve something, this will be translated to the goals of the web page. For instance, if we find a how-to web page on "How to make a chocolate cake", then the goal of that AP problem will be translated to something equivalent to how-to-make-a-chocolate-cake. Each section must be associated with a type, being one of the following:

Fig. 5 WikiHow sample web page

[8] http://www.wikihow.com/Make-Your-Own-Tortillas

- **actions**: a sequence of actions, representing the necessary steps to reach the goal. They will form the actions set of the AP domain model. Examples of actions are *combine flour and salt*, *cut in shortening*;
- **tools**: a set of tools needed to reach the goal with different semantics depending on the selected category. Examples are ingredients for the *cuisine* category or mechanical tools for the *building stuff* category. They will be the objects that will appear in the AP problems, more specifically in the initial state. Examples of tools are: *2 cups of flour*, or *1 spoon of salt*;
- **relatedwps**: other web pages related to the described task. Examples of related pages are *how to make Tortilla de Patatas*, *how to make Flour Tortillas*, *how to make Tacos*. This is usually not used within planning, but they open new possibilities for planning purposes, such as suggesting potentially relevant plans.

In Table 1, we show the equivalence between WikiHow concepts and AP concepts. Within this paper, we covered all aspects except for automatically generating heuristics and constraints from tips and warnings.

wikihow	planning
page	plan
title	goal
ingredients	initial state
tools	initial state
steps	instantiated actions
tips	heuristics
warnings	constraints, heuristics
related pages	related plans

Table 1 Equivalence between WikiHow concepts and AP concepts.

Since the Steps, Tips and Warnings sections, that are of type *actions*, the *ThingsYoullNeed* section of type *tools*, and the *RelatedWikiHow* section of type *relatedwps* are suggested by the WikiHow template layout, they occur in almost every page. They are parsed by default by the Page Processor, whereas further sections (e.g., *Ingredients* of type *tools*, usually added by the person that describes a recipe) have to be explicitly declared.

4 The Crawler

The Crawler subsystem is devoted to find an article using natural language or to find all the articles belonging to a specific category. Given the set of natural language queries, and the HowTo categories, the Crawler can perform the following functions:

- ArticleSearch: given a user input stated as a natural language query, it finds all relevant articles, sorted by relevancy rank. Furthermore, it selects the highest

ranking page and processes it. As an example, if the user enters the query *Make Tortilla*, pages like:

- http://www.wikihow.com/Make-Your-Own-Tortillas
- http://www.wikihow.com/Make-Tortilla-de-Patatas
- http://www.wikihow.com/Make-Tortilla-Pizzas
- http://www.wikihow.com/Make-Tortilla-Snacks
- ...

are suggested as relevant, and the first one (e.g., Make Your Own Tortillas) is automatically parsed by the ArticleProcessor (described later on)

- CategoryCrawler: finds all the articles belonging to a specific category. For instance, if the user enters the category recipes:
  http://www.wikihow.com/Category:Recipes
  the Crawler will find the 3144 recipes that currently belong to its 167 subcategories.[9]
- BatchExtractor: applies the page processing to all pages belonging to a category. It stores results in a file representing the category or in a database. Currently it can handle JDBC-compliant databases.

5 The Page Processor

The Page Processor subsystem is devoted to deal with a web page extracting the contents deemed important for AP purposes; its tools are devised to facilitate the recognition of sections, together with the contents type. Currently, the Page Processor includes the ArticleProcessor and the ArticlePostProcessor tools. The aim of the ArticleProcessor is to process a given HowTo article in order to extract the useful information in terms of semistructured relations. It also keeps some information about the page structure and the raw contents for further potential processing purposes. On the other hand the ArticlePostProcessor, that integrates semantic tools, builds an augmented plan in the form of predicates containing unstructured components. Exploiting the entire knowledge base, this information will be used to build plans.

5.1 The ArticleProcessor

Given as input a web page, the ArticleProcessor returns a tuple $< a,t,r >$ where a is a sequence of actions, t is a set of tools, and r is a list of related web pages. Each tuple can be seen as an augmented plan with information on its actions, a, initial state t, and related plans r. This processing phase tries to remove all noisy

[9] Values on September 10, 2009

information while avoiding to loose the relevant one required for further processing. The ArticleProcessor embeds an HTML parser devoted to cope with several errors, such as the ones related to the `<div>` closure tag, incoherences with the `id` attribute declarations, changes on the main structure of the page, or bad formatted HTML code.

This subsystem incorporates the current standard tools for processing natural language, such as stemming procedures, which remove inflectional and derivational suffixes to conflate word variants into the same stem or root, or stopwording procedures which remove words with a low information content (e.g., propositions, articles, common adverbs) from the text. The semantic analysis is performed by using WordNet,[10] a lexical database considered the most important resource available to researchers in computational linguistics, text analysis, and related areas. Its design is inspired by current psycholinguistic and computational theories of human lexical memory [10].

The raw contents of a sentence is also preserved to permit further tools to reparse it. As for the sections of type *actions*, the action itself of each sentence is recognized by identifying the verb or the corresponding compound. Furthermore, a set of parameters related to the action are stored and separated from the redundant part of the sentence. More specifically, both actions, tools and relatedwps can have related parameters. Next, we define the most relevant parameters of each type of information.

The parameters related to actions are:

- **action**: the main action, represented by a verb or a compound
- **components**: the components of the action
- **components-st**: the stopwording+stemming of the *components* field
- **plus**: sentences related to the action considered redundant
- **plus-st**: the stopwording+stemming of the *plus* field
- **raw**: the raw contents of the sentence

As an example of actions parsing, given two of the input sentences in the Tortillas Web page "Combine flour, salt, and baking powder in a large medium large bowl." and "Cut in shortening until lumps are gone.", the output of the parser would be:

ACTION:combine; COMPONENTS:flour; PLUS:salt, amp baking powder in a large medium large bowl; COMPONENTS-ST:flour; PLUS-ST:bowl larg bake powder amp medium salt; RAW:combine flour, salt, amp baking powder in a large medium large bowl. and

ACTION:cut; COMPONENTS:in shortening; PLUS:until lumps are gone; COMPONENTS-ST:shorten; PLUS-ST:lump gone until; RAW: cut in shortening until lumps are gone.

As for the elements of type tools, the information concerning *Quantity*, *Unit of measure* (e.g., units, grams, centimeters, cups, spoons) and *name of the ingredients* is stored. So their parameters are:

- **quantity**: the quantity/measure of the tool

[10] http://Wordnet.Princeton.edu/

- **type**: the unit of measure of the tool
- **tool**: the name of the tool
- **tool-st**: the stopwording+stemming of the *tool* field

As an example, given the sentence "$< b > 2 < /b >$ cups of flour", taken from the ingredients section of the *How To Make Your Tortillas* web page, the parser would generate: *QUANTITY:2; TYPE:cups; TOOL:of flour; TOOL-ST:flour; RAW:* $< b > 2 < /b >$ *cups of flour.*

In the case of the relatedwps, only the name and the HTTP URL are stored. They serve as indexes in our plan data base for accessing other plans. As an example of related web pages for the article Make Your Own Tortillas, it would generate the following two relations:

- URL:http://www.wikihow.com/Make-Your-Own-Tortillas; NAME:Make Your Own Tortillas; and
- URL:http://www.wikihow.com/Make-Tortilla-de-Patatas; NAME:Make Tortilla de Patatas; (in Figure 6)

Fig. 6 WikiHow sample of a related web page.

5.2 The ArticlePostProcessor

The ArticlePostProcessor rounds off progressively the collected information in order to build the plans structure in terms of steps (i.e., actions) and initial state (i.e., tools). This kind of representation is still not good enough for planning purposes, because it is expressed in a propositional representation, but it will be necessary during the further steps to build the plans corresponding to the articles in predicate logic. This will be possible taking into account the entire knowledge base.

Thus, given a web page, the ArticlePostProcessor builds its corresponding augmented plan. For each action and tool, the ArticlePostProcessor uses the information retrieved by the ArticleProcessor, encompassing in particular the semantical annotation, in order to define the end result. The plan representation contains information about

- The goal represented by the name of the web page
- The initial state in the form of needed tools represented as a tuple <name of the section, quantity/measure, unit of measure, name of the tool>
- The actions to reach the goal represented as a tuple <name of the section, ordinal number of the action, action name, action tools (if any)>

As an example, the following is an extracted plan for the web page How To Make Your Own Tortillas:

- **Goal**: make tortilla
- **Initial state**:
  - tool(ingredients,2,cup,flour):
  - tool(ingredients,1,tsp,salt):
  - tool(ingredients,1,cup,water):
  - ...
- **Plan**:
  - action(steps,1,combine,{flour,salt});
  - action(steps,2,cut,{shorten});
  - action(steps,3,make,{indentation});
  - action(steps,4,add,{water});
  - action(steps,5,work,{mixture});
  - ...

6 The Plan Acquisition

The Plan Acquisition subsystem includes tools that allow to create plans from web pages and to build new plans. The tools belonging to the Plan Acquisition subsystem are: (i) the PlanBuilder which aim is to build plans in predicate logic, and (ii) the AnalysisTool that embodies a suite of statistical tools.

6.1 The Plan Builder

Starting from the ArticlePostProcessor output we need to express the contents of the article in predicate logic (close to PDDL). The idea is to reduce the article to a list of two kinds of information:

- **input-tool**: *predicate that describes the tool properties*
- **action**: *action description with action properties*

These two predicates (in this first step, actions are represented as predicates) are sufficient to express all the information we need to build a plan for each article. However, we need to solve some problems due to the unstructured representation of the components given as output from the ArticlePostProcessor. The first problem is that sometimes tools cited in an action are not defined in the *tools* section. For instance, in an action as "put water", usually the water has not been defined as a tool (ingredient). Sometimes, also the opposite is true: a tool defined in the *tools* list, is not used within the actions. In these two cases, the tool is added/removed to/from the tools list.

Another problem, also due to the use of natural language to define the how-to pages, consists on tools being defined (written) in different ways, specially when analyzing different articles. For instance, a "boiled egg" in the recipes context has been found as *boiled egg* or also *eggs boiled* as well as *water hard boiled eggs*. To solve this problem, a partial string matching algorithm has been used. Considering the stemming of the words composing the name of the tool, the algorithm acts so:

- a list of known tools is generated from the most common to the least, associating an identifier to them
- when a new tool has been found, if it "fuzzy" matches with an already indexed one, it is labeled with the same identifier. A specific matching threshold can be specified as parameter.

The result is a table as Table 2. This table shows, for example, that both *olive virgin high quality oil*, and *pure olive virgin oil* are recognized as *olive virgin oil* and their identifier (for relational purposes) is 87.

A third problem relates to different ways of specifying quantities, so we have normalized them, reducing all of them to a double number. For instance, $\frac{1}{3}$ is stored as 0.33. When all information to build the predicates has been collected, predicates will be:

- **Input tool**: *tool(type, article_id, tool_id, quantity_normalized, quantity_type)*
- **Action**: *action_name(article_id, action_step, tool_id)*

where *type* is the type of the tool if known (ingredient, instrument, etc.), *article_id* is the id of the article (i.e., the id of the recipe), *tool_id* is the standard id of the tool, *quantity_normalized* is the *quantity* of the tool expressed in units of type *quantity_type* that belongs to a predefined enumerated set (i.e., spoon, grams, teaspoon, glasses, etc.), *action_name* is the name of the action (i.e., combine, cut, put,

tool-id	tool-name	tool-original-name
...	...	...
87	olive virgin oil	pure olive virgin oil
87	olive virgin oil	olive quality extra virgin oil
87	olive virgin oil	olive virgin high quality oil
87	olive virgin oil	olive best virgin oil
...	...	...
...	...	...
175	fine chopped onion	chopped fine onion
175	fine chopped onion	white chopped fine onion
175	fine chopped onion	chopped fine large onion
175	fine chopped onion	chopped fine medium size onion
...	...	...

Table 2 Example of index of tools.

etc.), and *action_step* is the ordinal position of the action in the actions list. If the action requires more than one parameter, we define several action predicates with the same action_step (i.e., `combine(water, flour)` $\rightarrow$ `combine(water), combine(flour)`).

Tables 3, and 4 show examples of the generated output. They can be computed for the plans corresponding to all the articles in WikiHow or all the ones belonging to a subcategory (i.e., recipes that use onions, or recipes for making tortillas).

action_name(article_id,step_id,tool_id)
...
use(2903,3,56)
get(2911,1,103)
spread(2911,2,103)
put(2906,16,128)
...

Table 3 A fragment of the output file: Actions.txt.

input(type, article_id, ingredient_id, quantity, quantity_type)
...
input(ingredient,10,634,1,cup)
input(tool,5,651,1,teaspoon)
input(ingredient,5,978,1,units)
...

Table 4 A fragment of the output file: Tools.txt.

6.2 The AnalysisTool

The AnalysisTool contains data mining and statistical algorithms. Statistics extracted by this tool could be useful to understand which component or action are most likely to be used or applied in a specific context. This could be used to build new plans, or understand which are the most common subsequences of actions. Experiments have been performed exploiting actions, goal, tools frequency tables, and goal $\rightarrow$ action and goal $\rightarrow$ tool correlation tables. If we express the correlation between X and Y as $C(X,Y) = F$, where F is the value of the frequency of how many times the object X appears in the context Y, an example of correlation between goal components and actions performed in the context of the recipes category is:

- C(cake, pour) = 19.12934%
- C(sandwich, put) = 19.01585%
- C(cream, add) = 17.94737%
- C(cake, bake) = 14.81189%

This highlights that, for instance, it's likely (with probability above 19%) to perform the action *put* when the goal is to make a *sandwich*. It will be also useful to analyze particular subsequences of plans in specific contexts (e.g., what are the most likely actions to be performed on an onion in recipes needing oil?).

7 Results

PAA has been developed in Java using the version 1.6.0.11 of the Sun Java Development Kit. NetBeans 6.5.1[11] has been used as IDE. For the experimental phase, a GUI and a Glassfish WebService integrating the architecture have been deployed. Most of the libraries and the distributed computing capabilities rely on the X.MAS framework [3].

We have applied PAA to three WikiHow categories. In Table 5 we show some data on the analysis we have performed over different categories within WikiHow. For each category, we show the number of different plans (pages) parsed, the number of subcategories found, and the number of extracted different actions.

The error on unrecognized actions (meaning that the current version of PAA could not be able to semantically parse some sentences and recognize their structure) is about 2%. In general, it is difficult to assess PAA performance, mainly due to the fact that there is no known gold standard; that is correct representations in terms of plans of those pages, so that we could automatically compare against. Hence, with regard to the acquired plans, we did some ad-hoc analysis by manually inspecting some output plans.

In order to perform a first measure of the accuracy of the system the matching between the article and the output of the *ArticlePostProcessor* was analyzed for a set

[11] http://www.Netbeans.org

Recipes: *http://www.wikihow.com/Category:Recipes*	
3144	recipes parsed/acquired plans
167	sub-categories found
24185	different individual actions
Sports: *http://www.wikihow.com/Category:Team-Sports*	
979	recipes parsed/acquired plans
22	sub-categories found
6576	different individual actions

Table 5 Some data on performed analysis

of 40 random taken articles (566 single actions). The accuracy has been calculated taking into account the total number of recognized actions: an action is recognized if its name and the tool match the corresponding row of the article. If an action is missed or it is not well translated, this is considered as a non-match.

Then the accuracy is calculated as $Accuracy = \frac{(matching\ actions)}{(total\ actions)}$. The error can be defined as $Error = (1 - Accuracy) = \frac{non-matching\ actions}{total\ actions}$. An example of measurement took on the WikiHow article "*Make your own tortillas*" is shown on table 6.

Article row	**Action(s)**	**Results and Notes**
1) Mix flour, salt, & baking powder in a large medium/large bowl.	mix(flour, salt)	2 correct, 1 mistake (mix baking powder was missed)
2) Cut in shortening/lard until lumps are gone.	cut(shortening)[12]	1 correct
3) Make a hole in the center of the dry ingredients.	make(hole)	1 correct
4) Add water, about a half a cup at a time, and work mixture into a dough. The dough should be slightly sticky but not hard. You can add slightly more water or flour if needed.	add(water), work(mixture)	2 correct
5) Cover and set aside for 10 minutes.	cover(set), set(aside)	2 mistakes (set and aside are not components)
6) Make the dough into balls about the size of eggs.	make(dough)	1 correct
7) Using a rolling pin, roll each dough ball into about a 6 inch circle.	roll(dough)	1 correct
8) Heat griddle or skillet on medium heat without grease.	heat(griddle)	1 correct
9) Cook tortilla 1/2 to 1 minute (if it starts to bubble, that's long enough).	cook(tortilla)	1 correct
10) Flip tortilla to the other side and cook for a few seconds.	flip(tortilla), cook(few)	1 correct, 1 mistake (few is not a component)
11) Continue until all your dough is cooked.	continue(until)	1 mistake (until is not a component)
12) Then you can eat!	eat()	1 correct

Table 6 Measuring the accuracy of the system, an example with the article "make your own tortillas"

This first analysis shows that the system performed rather well on plan extraction (above 68% of accuracy), considering the complexity of the semantic analysis tasks and the need to handle many outliers. In fact, parsing an HTML page, even if automatically generated from a php engine, is not trivial due to *code injection* during the compiling of the predefined structure, and to the addition of different sections depending on the context (e.g., *ingredients* in recipes, or *work tools* in machinery). Besides, sentences are structured in different ways and filled with different kinds of contents more than "simple" steps. For instance, some people add a lot of non descriptive text (e.g., from a step for the "Make Vodka" how-to: *Column stills produce purer alcohol because they essentially redistill the alcohol in a single pass*). Others add playful goals with facetious suggestions (e.g., from the "Microwave a Peep" how-to: *Don't hurt yourself with the fork. You will be a slave to the Peeps if you eat it at all*). Moreover, somebody slangs or adds actions not related to the goal (e.g., *Stir and enjoy!*). The preprocessor has to handle all this, other than trying to manage compound forms, exploiting redundant descriptions of the action in the sentence and attempting to discover tools not explicitly cited.

To test the validity of the subsystems composing the architecture, the system has been tested by five selected users. The users monitored the behavior of the system through a web service that integrates the tools over a two-week period. By conducting regular interviews with each user to estimate her/his satisfaction we could verify the correctness of the process. All users stated their satisfaction with the system that succeeded in translating most of the articles into understandable plans. They also allowed to spot most of the processing problems.

Clearly, it is impossible to perform perfectly well. However, we reached our goal to have a good tradeoff between information retrieved from a web page and information lost during the filtering process. Also, we obtained a reasonably good tradeoff between semantical comprehension and introduction of errors. Thus, even if the integrated technologies that compose our tools are subject to future improvements, they already gave us a base on which to work and play on collected information in the next future.

8 Conclusions and Future Work

In this chapter, we described a work aimed at bridging the gap between the need of tools for automatically building plans and action models from semi-structured information and the existence of this kind of knowledge in the Web (e.g., WikiHow). We believe this work can encourage further research in this topic that can greatly help the massive application of planning to many real-world human related tasks. Experimental results show that the tools performed well on extracting plans, thus establishing a preliminary base on which to work.

As for future work, the *analysis of common subsequences* on multiple plans in the same domain will be performed. We will base this analysis on previous work on planning as macro-operators [12], n-gram analysis of natural language tools [9],

or association rules learning [15]. This can be further used for performing case-based planning by recovering subsequences that include some specific object. For instance, subsequences of recipes that use a particular ingredient or tool in general. Also, it could be used by tools that help on inputing recipes on the WikiHow by suggesting previous plans subsequences for the ingredients. Furthermore, we plan to include a Planner subsystem that contains tools necessary to exploit the collected knowledge base (e.g., the plans library) *to build new plans*. Other uses of this library will be aimed at performing *plan recognition* from data coming from sensors and at matching such data against the plans recovered from the web, or at *acquiring complete action models* with preconditions and effects from the input plans [23]. We will also exploit different knowledge bases as *eHow*[13] an online knowledge resource offering step-by-step instructions on *how to do just about everything*. eHow content is created by both professional experts and amateur members and covers a wide variety of topics organized into a hierarchy of categories; or *HowToDoThings*[14] another hierarchically organized knowledge base of how-to manuals, in order to make our architecture even more general and independent from the particular web site.

References

1. Addis, A.; Armano, G.; and Vargiu, E. 2008. WIKI.MAS: A multiagent information retrieval system for classifying Wikipedia contents. *Communications of SIWN* 3(June 2008):83–87.
2. Addis, A.; Armano, G.; and Borrajo, D. 2009. Recovering Plans from the Web *Proceedings of SPARK, Scheduling and Planning Applications woRKshop, ICAPS'09*.
3. Addis, A.; Armano, G.; and Vargiu, E. 2008. From a generic multiagent architecture to multiagent information retrieval systems. In *AT2AI-6, Sixth International Workshop, From Agent Theory to Agent Implementation*, 3–9.
4. Birukov, A.; Blanzieri, E.; and Giorgini, P. 2005. Implicit: an agent-based recommendation system for web search. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 618–624. New York, NY, USA: ACM Press.
5. Rodrguez-Moreno, M. D.; Borrajo, D.; Cesta, A.; and Oddi, A. 2007. Integrating Planning and Scheduling in Workflow Domains *Expert Systems with Applications* 33(October 2007):389–406.
6. Burdman, J. 1999. *Collaborative Web Development: Strategies and Best Practices for Web Teams*. Addison-Wesley Longman Ltd.
7. Camacho, D.; Aler, R.; Borrajo, D.; and Molina, J. 2005. A multi-agent architecture for intelligent gathering systems. *AI Communications, The European Journal on Artificial Intelligence* 18(1):1–19.
8. Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodriguez, A.; Fernández, S.; Arias, J. D.; and Borrajo, D. 2008. SAMAP: An user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(34):1318–1332.
9. Dunning, T. 1994. Statistical identification of language. Technical report.
10. Fellbaum, C. 1998. *WordNet An Electronic Lexical Database*. Cambridge, MA ; London: The MIT Press.

[13] http://www.eHow.com/

[14] http://www.howtodothings.com/

11. Fernndez, S.; Borrajo, D.; Fuentetaja, R.; Arias, J. D.; and Veloso, M. 2007. PLTOOL. A KE tool for planning and learning. *Knowledge Engineering Review Journal* 22(2):153–184.
12. Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.
13. Gil, Y. 1991. A domain-independent framework for effective experimentation in planning. In *Proceedings of the Eighth International Workshop (ML91)*, 13–17.
14. Hoffmann, J.; Weber, I.; and Kraft F. M. 2009. Planning@SAP: An Application in Business Process Management *Proceedings of SPARK, Scheduling and Planning Applications woRKshop, ICAPS'09*.
15. Kavsek, B.; Lavrac, N.; and Jovanoski, V. 2003. *Lecture Notes in Computer Science*, volume 2810. Springer Verlag. chapter APRIORI-SD: Adapting Association Rule Learning to Subgroup Discovery, 230–241.
16. Manzano-Macho, D.; Gmez-Prez, A.; and Borrajo, D. 2008. Unsupervised and domain independent ontology learning. combining heterogeneous sources of evidence. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*.
17. O'Reilly, T. 2005. *What is Web 2.0, Design Patterns and Business Models for the Next Generation of Software*. O' Reilly.
18. Pollack, M.; Brown, L.; Colbry, D.; McCarthy, C.; Orosz, C.; Peintner, B.; Ramakrishnan, S.; and Tsamardinos, I. 2003. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems* 44:273–282.
19. Sánchez, D.; Tentori, M.; and Favela, J. 2008. Activity recognition for the smart hospital. *IEEE Intelligent Systems* 23(2):50–57.
20. Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using gipo. *Knowl. Eng. Rev.* 22(2):117–134.
21. Tapia, E. M.; Intille, S. S.; and Larson, K. 2004. *Pervasive Computing*. Springer Berlin / Heidelberg. chapter Activity Recognition in the Home Using Simple and Ubiquitous Sensors, 158–175.
22. Wang, X., and Veloso, M. M. 1994. Learning planning knowledge by observation and practice. In *Proceedings of the ARPA Planning Workshop*, 285–294.
23. Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.
24. Yang, H. C. 2005. A general framework for automatically creating games for learning. In *Proceedings of the fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, 28–29.
25. Zhou, L. 2007. Ontology learning: state of the art and open issues. *Inf. Technol. and Management* 8(3):241–252.
26. Zimmerman, T.; and Kambhampati, S. 2003. Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward *AI Magazine* 24(Summer 2003):73–96.