Mixed-Initiative, Multi-Source Information Assistants

Craig A. Knoblock*
Univ. of Southern California
knoblock@isi.edu

Steven Minton
Fetch Technologies
minton@fetch.com

José Luis Ambite
Univ. of Southern California
ambite@isi.edu

Maria Muslea
Univ. of Southern California
mariam@isi.edu

Jean Oh
Univ. of Southern California
jeanoh@isi.edu

Martin Frank
Univ. of Southern California
frank@isi.edu

ABSTRACT

While the information resources on the Web are vast, the sources are often hard to find, painful to use, and difficult to integrate. We have developed the Heracles framework for building Web-based information assistants. This framework provides the infrastructure to rapidly construct new applications that extract information from multiple Web sources and interactively integrate the data using a dynamic, hierarchical constraint network. This paper describes the core technologies that comprise the framework, including information extraction, hierarchical template representation, and constraint propagation. In addition, we present an application of this framework, the *Travel Assistant*, which is an interactive travel planning system. We also briefly describe our experience using the same framework to build a second application, the *WorldInfo Assistant*, which extracts and integrates geographic-related data about countries thorughout the world. We believe these types of information assistants provide a significant step forward in fully exploiting the information available on the Internet.

1. INTRODUCTION

People use search engines today to find information, but in many cases what people actually want is an application that allows them to access a set of related sources, extract the information they need, and integrate the data in ways that allows them to solve their problems. The future of the Web involves going beyond traditional information-retrieval-based search engines to much more advanced integration frameworks. In this paper we present a framework for building information assistants, which support the construction of domain-specific applications that extract and integrate data to support a specific task. These are not search engines, but rather applications that organize and integrate data to support a particular task.

Consider the problem of travel planning on the Web. There are a huge number of travel sites, each of which provides different types of information. You can go to one site and get hotel and flight information, another site to get the airports that are closest to your destination, a third site to get directions to your hotel, and yet a fourth site to find out the weather in the destination city. It is a tedious process to go to each of these sites, repeatedly entering the same information about dates, addresses, etc. Instead you would like the best of the Internet-related travel sources combined into a single integrated environment that can help you plan a trip.

We have developed the Heracles framework to solve these types of information gathering and management problems. We have applied our general framework for creating information assistants to build an example travel assistant. The resulting system helps a user plan out a business trip from beginning to end. When you start the system, it first looks up the upcoming meetings in your calendar. After you have selected a meeting, it extracts the dates for the meeting, looks up the location, checks the weather, and even makes a recommendation about whether you should fly or drive to the meeting. It goes beyond what is provided in most travel web sites by comparing the costs of various choices, such as the costs of taking a taxi to the airport or parking a car there. It helps select a hotel based on your meeting location. And can even help you plan your trip to help minimize the total cost or the total time you will be away from home. In short, it provides all of the information that you need to plan a trip and links this information together to provide a unified framework to quickly and efficiently work out the details of a trip and make the appropriate reservations.

The Heracles framework for building information assistants consists of three components. First, we have a set of tools for creating what we call wrappers that allow web sources to be queried as if they are databases. This is critical for turning the many human-readable web sources into data sources that can be used and integrated within our system. Second, we have developed an interactive, hierarchical constraint propagation system that provides the reasoning system for integration applications. Third, we have built an interactive graphical user interface that allows an end user to interactively control the entire process. In this paper we describe each of these components and describe our experience in applying the framework to several different applications.

*Mailing address: 4676 Admiralty Way, Marina del Rey, CA, USA 90292-6601

Copyright is held by the author/owner.
WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

2. THE TRAVEL PLANNING ASSISTANT

We applied the Heracles system to the travel domain, integrating the information sources that are needed for travel planning into a single application, the *Travel Assistant*. This real-time application can plan a trip in a semi-automated fashion, gathering live data from web sources and other programs.

When the user initiates a new trip, the system extracts a list of people from his Outlook calendar that he has scheduled meetings with in the next two months. When the user selects whom to meet with, the system retrieves the person's company name and the address of the company from the Outlook Address Book. The company address is used as the default destination of the trip. The starting address is retrieved from the user's personal profile. The system also retrieves the dates, start time, and end time of the meeting from the Outlook calendar. As soon as the dates of the meeting and the address of the destination are available, the system retrieves the weather forecast from the Yahoo! weather site (weather.yahoo.com).

Figure 1 shows the selection of the meeting in the *Travel Assistant*. There is a set of boxes showing values, which we call *slots*. A slot holds a current value and a set of possible values, which can be viewed in the pull down list if we click the arrow at the right edge of the slot. For example, there are two slots in the first line: a slot `Person` and another slot `Company Name`, holding values `Jim Hendler` and `DARPA`, respectively.

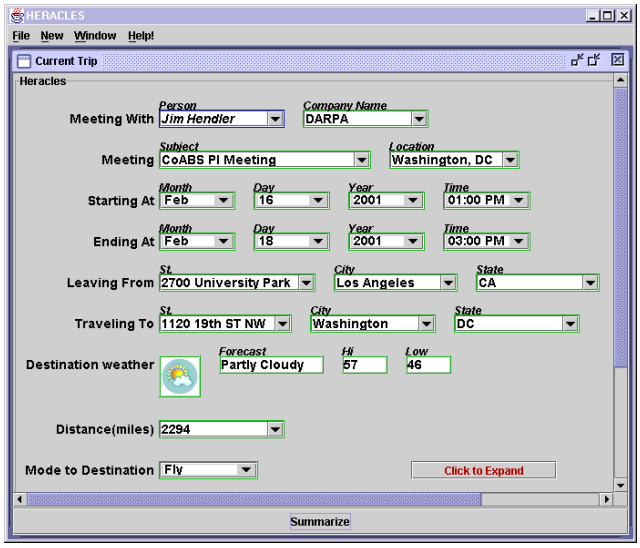


Figure 1: Meeting Details Retrieved Directly from Calendar

Once the system has the details of the meeting, the next step is to determine how to get to the destination. There are three possible modes of transportation: `Fly`, `Drive` or `Take a Taxi`. Each mode has different choices and information, which are organized into groups of slots called *templates* and then arranged hierarchically into templates and *subtemplates*. For example, the `Fly` subtemplate introduces information about the closest airports and available flights, whereas the `Drive` subtemplate provides maps and driving directions.

The system recommends the transportation mode based on the distance between the origin and destination. The system computes the distance by first geocoding (determining the latitude and longitude) of the origin and destination addresses using the MapBlast Web site (www.mapblast.com). Then, using the geocoded information, a local constraint computes the distance between the two points by applying a distance calculation formula. In our example, the distance between Los Angeles and Washington D.C. is 2,294 miles, so the system recommends that the user `Fly`.

Once the user confirms that he wants to fly, the system shows the details of the `Fly` subtemplate (Figure 2). Based on the destination address passed from the top-level template, the system queries the Travelocity Web site (www.travelocity.com) to get airports in the Washington D.C. area ordered by distance. The system recommends DCA (National Airport) because DCA is the closest airport to the meeting location. Once the airports have been selected, the system then queries ITA Software (www.itasoftware.com) to retrieve the flights from LAX to DCA.

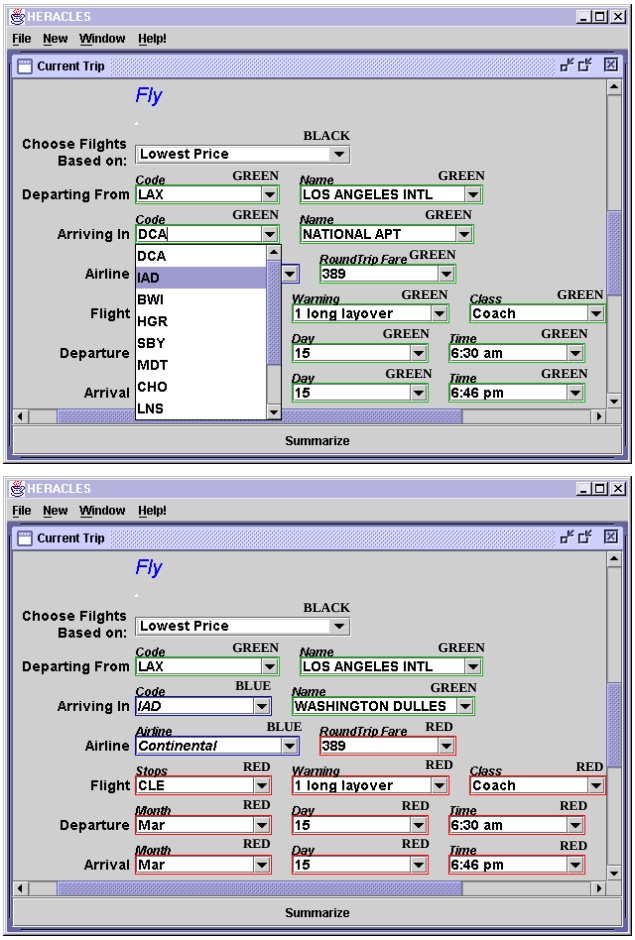


Figure 2: Slot Colors Show the Processing Status and where the Information Came From

The interface informs the user of its processing status using the colors shown in Figure 2.[1] A slot can appear in four

[1]The words `black`, `blue`, `red`, and `green` do not appear in the actual interface, but are simply labeled in the paper for black and white printers. A color version of the paper is available from http://www.isi.edu/info-agents/heracles.

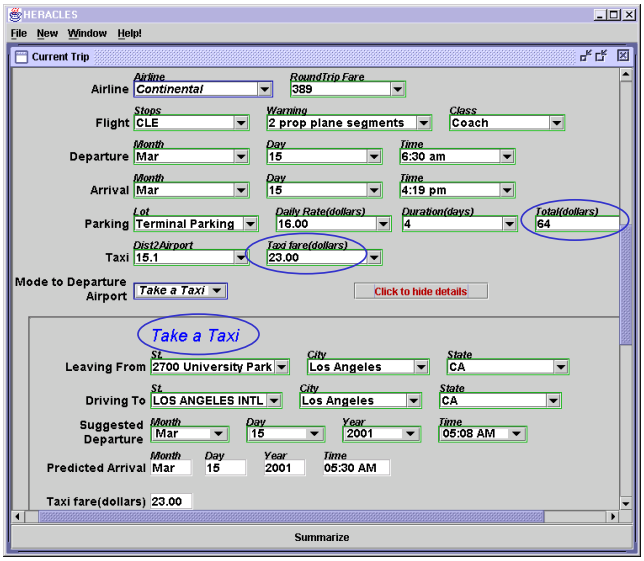
different colors as its status changes. A slot is initially **black** when it has only the default set of values. A slot becomes **blue** if it is the user who entered a value for it. While the system is computing a possible value for a slot, it turns **red**, and once the system produces a suggested value it changes color to **green**.

As shown in Figure 2, when the user changes DCA (National) to IAD (Washington Dulles), the arrival airport slot becomes blue because it is the user's choice. The system starts to get a new set of flights from LAX to IAD using the ITA Software site. The slots holding the flight information turn red until the new data is returned from. When the system finally recommends the new values for the slots, they become green.

The user can always override what the system suggests. The system initially finds flights for any airlines, but if the user fixes the airline to be `Continental`, for instance, the *Travel Assistant* will present only the Continental flights. The user's choices also serve to narrow the possible options.

The *Travel Assistant* helps the user evaluate tradeoffs that involve many different pieces of information and calculations. For example, Figure 3 illustrates how the system recommends the mode of transportation to the departure airport. This recommendation is made by comparing the cost of parking a car at the airport for the duration of the trip to the cost of taking a taxi to and from the airport. The system computes the cost of parking by retrieving the available airport parking lots and their daily rates from the AirWise site (www.airwise.com), determining the number of days the car will be parked based on scheduled meetings, and then calculating the total cost of parking. Similarly, the system computes the taxi fare by retrieving the distance between the user's home address and the departure airport from the Yahoo! Map site (maps.yahoo.com), retrieving the taxi fare from the WashingtonPost Taxi Fare site (www.washingtonpost.com), and then calculating the total cost. In the figure, the system recommends taking a taxi since the taxi fare is only $23.00, while the cost of parking would be $64.00 using the `Terminal Parking` lot. When the user changes the selected parking lot to `Economy Lot B`, which is $5 per day, this makes the total parking rate cheaper than the taxi fare, so the system changes the recommendation to `Drive`.

The system actively maintains the dependencies among slots so that changes to earlier decisions are propagated throughout the travel planning process. For example, Figure 4 shows how the Taxi template is affected when the user changes departure airport in the higher-level `Fly` template. In the original plan (top template of Figure 2), the departure time from the origin airport (LAX, Los Angeles International) is 6:30 AM. The user's preference is to arrive an hour before the departure time, which means that he would need to arrive at LAX by 5:30 AM. Since according to Mapblast driving takes 22 minutes from his home to LAX, the system recommends that he leaves home at 5:08 AM. When the user changes the departure airport from LAX (Los Angeles International) to LGB (Long Beach), the system retrieves a new map and recomputes the driving time to the Long Beach airport. Changing the departure airport also results in a different set of flights. The recommended flight leaving from LGB departs at 6:55 AM, and the driving takes 34 minutes from his home to LGB. In order to arrive



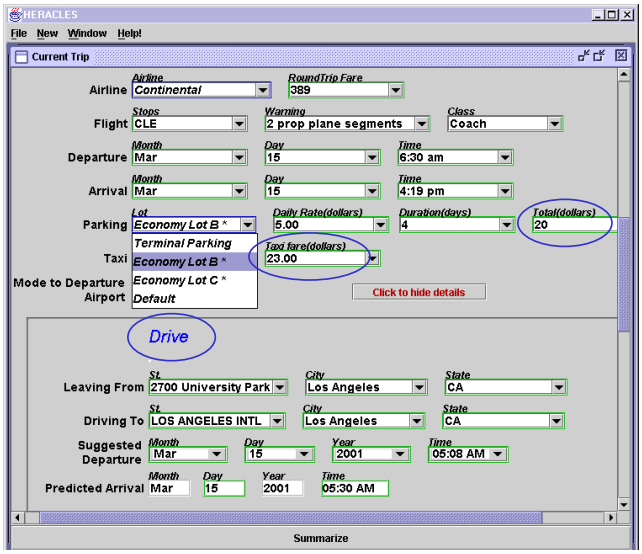


Figure 3: Compares Cost of Driving versus Taking a Taxi

at LGB by 5:55 AM, the system now suggests he leave home at 5:21 AM.

The *Travel Assistant* allows the user to control the various tradeoffs between the many choices that need to be made in planning a trip. For example, when selecting the hotel, the user can specify that he wants the cheapest one possible, the one closest to the airport, or the one that is closest to their meeting. Figure 5 shows the expansion of the `Hotel` subtemplate. The system retrieves the hotel, address, and fax information from the ITN site (www.itn.com) and the map from the airport to the hotel from the Yahoo! Map site. In the figure, the system initially suggests the hotel that is closest to the meeting and when the user changes that to the one closest to the airport, the system recomputes the set of hotels (ordered by distance), suggests the closest one, showing the address, price, and maps.

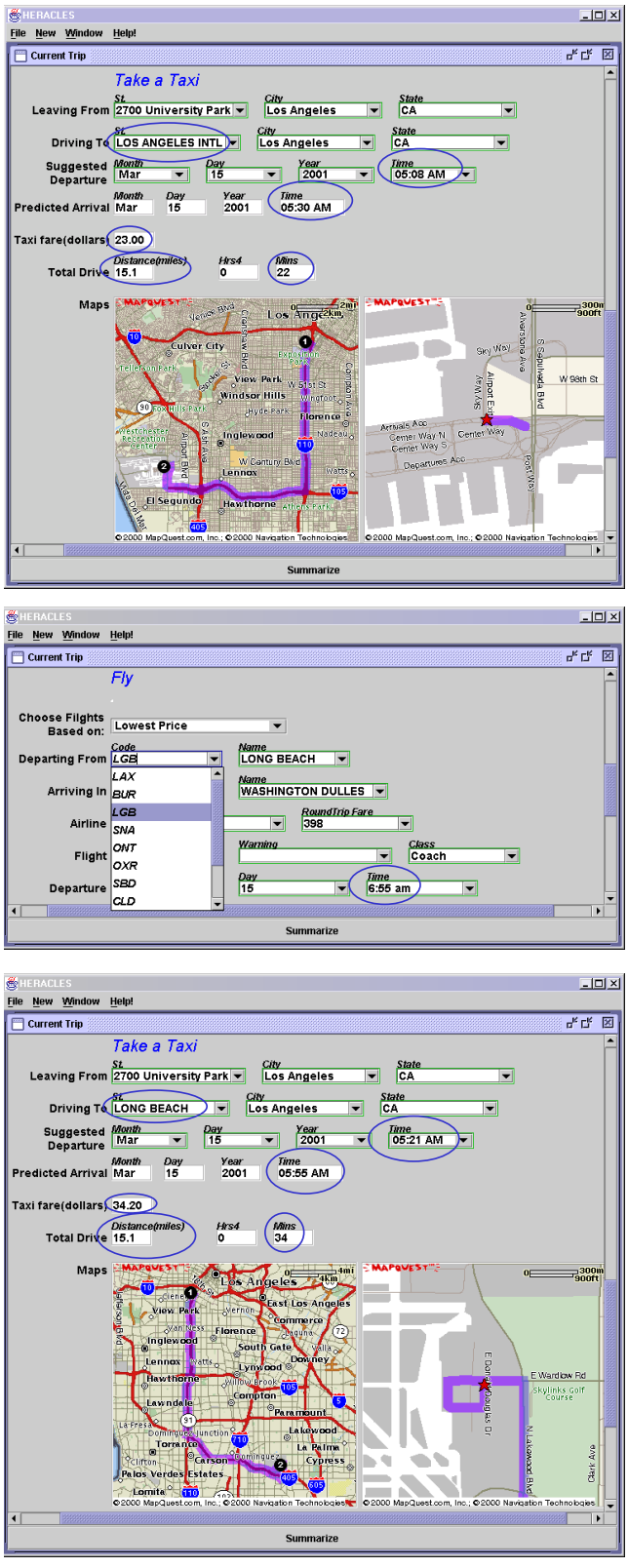


Figure 4: Change in Selected Airport Propagates to Drive Subtemplate

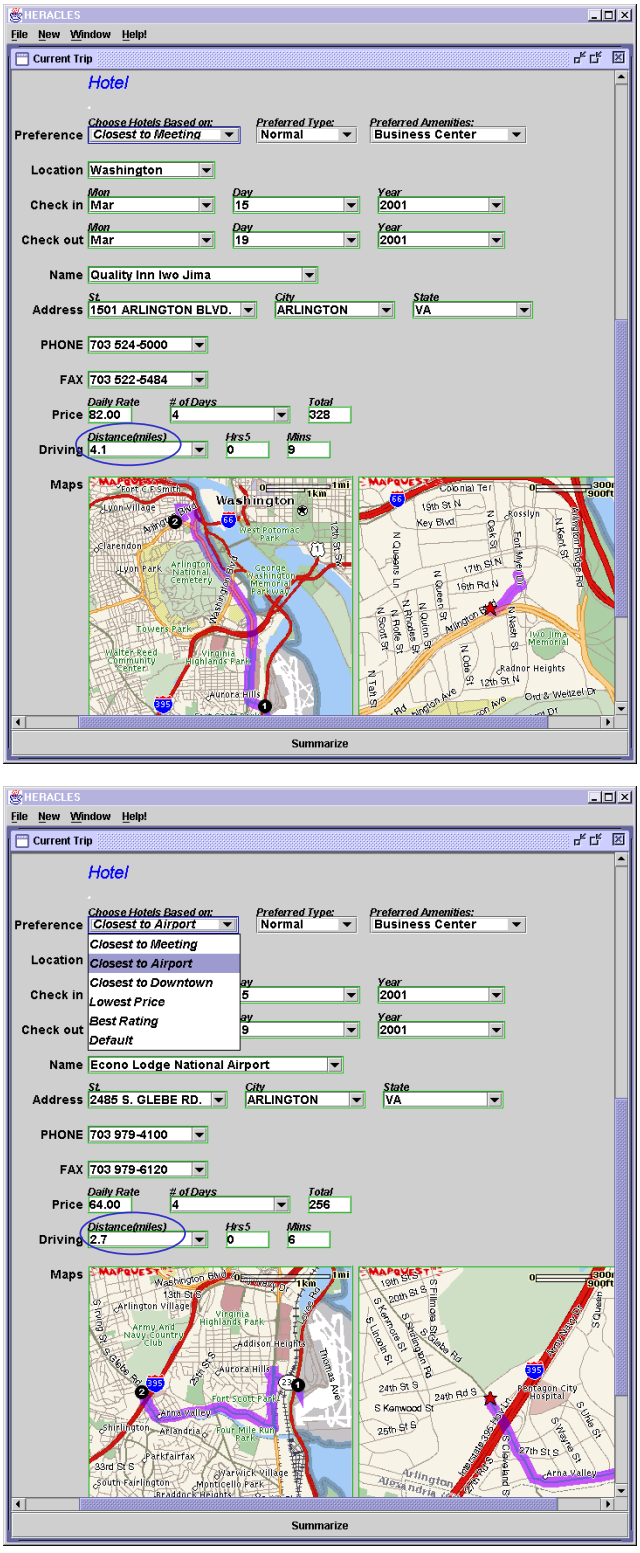


Figure 5: User Can Specify High-Level Hotel Preference

3. ACCESS TO WEB SOURCES

Access to on-line data sources is a critical component of our information assistants. In the *Travel Assistant* there is no data stored locally in the system. Instead all information is accessed directly from web sources. To do this we build wrappers that turn web sources into structured data sources. This allows the system to reason with the data and integrate the information with other data sources. As XML becomes more widely used, the access to data will become easier, but it will be a long time before most of the required data will be available as structured sources.

A wrapper is a program that turns a semi-structured information source into a structured source. This idea is shown in Figure 6 where the Yahoo! weather source is dynamically turned into an XML data source. Since the weather data changes frequently it would not be useful to download this data in advance. Instead the wrapper provides access to the live data, but provides it in a structured form. Once we have built such a wrapper, the *Travel Assistant* can send HTTP requests to the wrapper and get back XML tuples.

We have developed a set of tools for semi-automatically creating wrappers for web sources [8]. The tools allow a user to specify by example what the wrapper should extract from a source. The examples are then fed to an inductive learning system that generates a set of rules for extracting the required data from a site. The user interface for the wrapper learning system is shown in Figure 7. The window in the upper right shows the original web page, the window in the upper left shows the labeled data for this page, and the bottom window shows the learned extraction rules. Beyond just creating the rules, we have also developed techniques for ensuring that the system is extracting the right data [7], monitoring the source to ensure the it continues to function properly [5], and automatically repairing wrappers in response to format changes in a site [3].

Once a wrapper for a site has been created, one can use that site programmatically. For example, with the wrapper for Yahoo! Weather, we can now send a request to get the weather for a particular city and it will return the corresponding XML data with the weather for that city. As we mentioned earlier, there is no data stored in the application. This minimizes the work involved in maintaining an assistant and ensures that the assistant has access to the latest information.

4. CONSTRAINT NETWORKS FOR MANAGING INFORMATION

The critical challenge for the Heracles system is in integrating multiple information sources, programs, and constraints into a cohesive, effective tool. We have seen some examples of these diverse capabilities in the *Travel Assistant*, for example, retrieving scheduling information from a calendar system, computing the duration of a given meeting, and invoking a web wrapper to find directions to the meeting.

Constraint reasoning technology offers a clean way to integrate multiple heterogeneous subsystems in a plug-and-play approach, maximizing overall efficiency via rapid information propagation between components. The basis for our approach is a constraint representation where we model each piece of information as a distinct variable[2] and describe the relations that define the valid values of a set of variables as constraints. A constraint can be implemented either by a local procedure within the constraint engine, or by an external component that interfaces to the constraint engine (such as a wrapper or an off-the-shelf scheduler).

Using a constraint-based representation as the basis for control has the advantage that it is a declarative representation and can have many alternative execution paths. Thus, we need not commit to a specific order for executing components or propagating information. The constraint propagation system will determine the execution order in a natural manner. The constraint reasoning system propagates information entered by the user as well as the system's suggestions, decides when to launch information requests, evaluate constraints, and compute preferences. All of these tasks run as asynchronous processes to give the user as much support as possible without interfering with his work.

In order to manage the complexity and capture the task structure of the application, closely related variables and constraints are encapsulated into templates. The templates are organized hierarchically so that a higher-level template representing an abstract task (e.g., Trip) may be decomposed into a set of more specific subtasks, called subtemplates (e.g., Fly, Drive, etc). This hierarchical task network structure helps to manage the complexity of the application for the user by hiding less important details until the major decisions have been achieved.

In this section, we describe the representation for the variables, the constraints, the hierarchical templates, and the constraint propagation algorithm.

4.1 Constraint Network Representation

A constraint network is a set of variables and constraints that inter-relate and define the valid values for the variables. Heracles represents all the pieces of information in a given application and their inter-relationships as a constraint network. Figure 8 shows the fragment of the constraint network of the *Travel Assistant* that addresses the selection of the method of travel from the user's initial location to the airport. The choices under consideration are: driving one's car (which implies leaving it parked at the airport for the duration of the trip) or taking a taxi. We will use this example throughout the section to illustrate the components of a constraint network and the constraint propagation algorithm.

4.1.1 Variables

Each distinct piece of information in an application is represented as a variable in the constraint network. Each variable takes values from a given domain. At any point in time a subset of the domain of a variable constitutes its set of *possible* values. The possible values represent the system's view of the available choices for that variable, that is, those that are currently consistent with all the constraints.

In addition, each variable may have an *assigned* value. The assigned value of a variable can be set directly by the user or selected by the system from the possible values (see Section 4.3). If the user selects a value it remains as the

[2]In the example of Section 2 we have referred to each piece of information presented to the user as a slot. We use the term slot for user interface purposes. Each slot has a corresponding variable defined in the constraint network, but there may be variables that are not presented to the user (see Section 5).

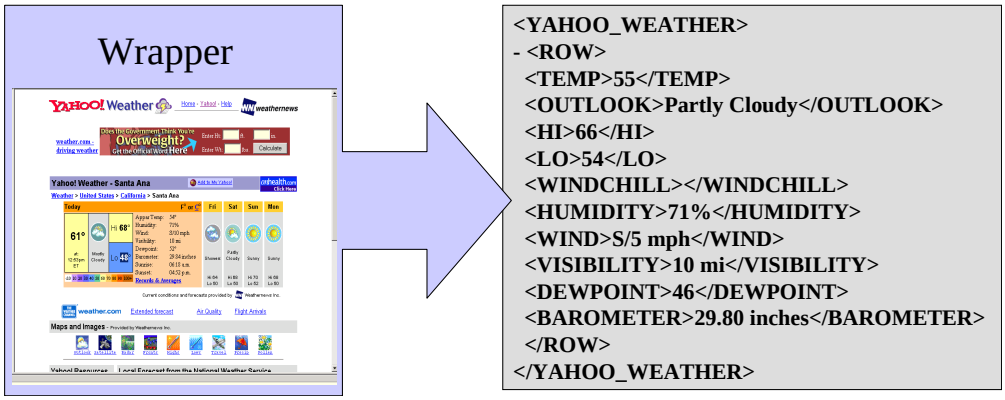


Figure 6: A Wrapper Turns Yahoo! Weather into an XML Data Source

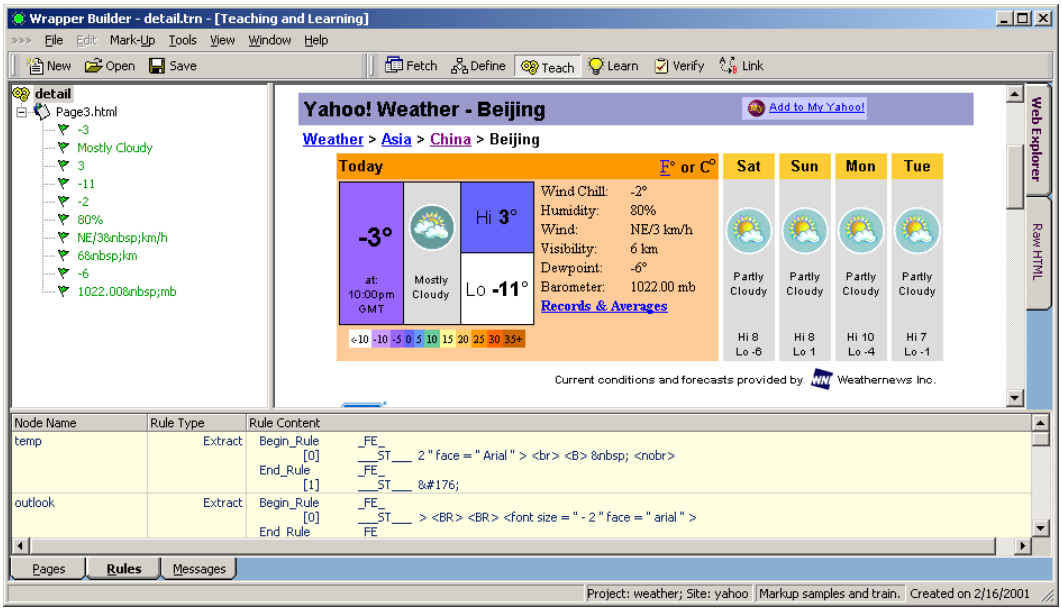


Figure 7: The Graphical User Interface for Building a Wrapper

assigned value unless it is changed again by the user or it becomes inconsistent with the constraints.

In the sample network of Figure 8 the variables are shown as dark rectangles and the assigned values as white rectangles next to them. The variables capture the relevant information for this task in the application domain, such as the `DepartureDate`, the `Duration` of the trip, the `ParkingTotal` (the total cost of parking for the duration of the trip), the `TaxiFare`, and the `ModeToAirport`. The `DepartureAirport` has an assigned value of `LAX` (Los Angeles International), which is assigned by the system since it is the closest airport to the user's address.

4.1.2 *Constraints*

Conceptually, a constraint defines the valid combinations of values for a set of variables. A *primitive constraint* is a n-ary predicate that relates a set of n variables by defining the valid combinations of values for those variables. A primitive constraint is a computable component which may be implemented by a local table look-up, by the computation of a local function, by retrieving a set of tuples from a remote wrapper, or by calling an arbitrary external program.

In the sample network of Figure 8 the constraints are shown as rounded rectangles. For example, the `compute-Duration` constraint involves three variables (`DepartureDate`, `ReturnDate`, and `Duration`), and it's implemented by a function that computes the duration of a trip given the departure and return dates. The constraint `getParkingRate` is implemented by calling a wrapper that accesses a web site that contains parking rates for airports in the USA.

In Heracles, the set of possible values of each variable is determined by a *domain expression*. The primitive constraints that operate on the same variable are combined to form a domain expression. The grammar for domain expressions is:

```
Exp = (AND Exp Exp) |
      (OR Exp Exp) |
      (DLIST Exp Exp ...) |
      PrimitiveConstraint
PrimitiveConstraint =
      (PREDICATE var1 var2 ... varN)
```

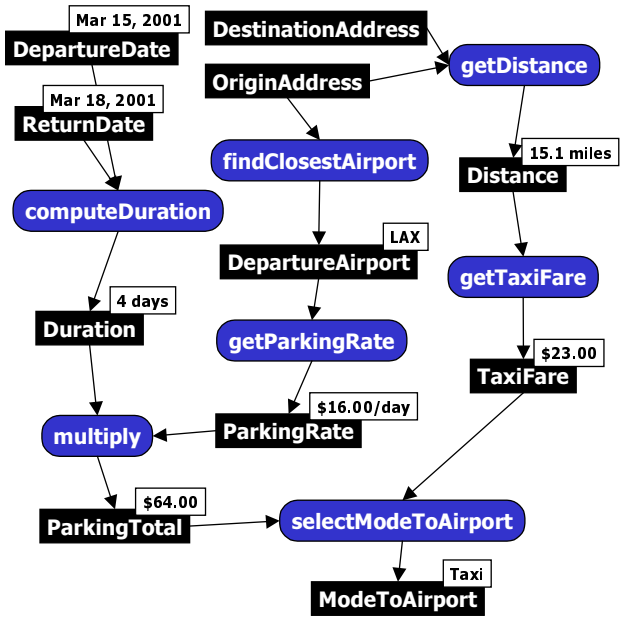


Figure 8: Constraint Network Comparing Driving Versus Taking a Taxi

The semantics of domain expressions is as follows: A conjunction (AND) of domain expressions evaluates to the intersection of the corresponding value sets. A disjunction (OR) evaluates to the union of value sets. A decision list (DLIST) takes the values of the first expression that evaluates to a non-empty value set. A domain expression can also be a primitive constraint and it evaluates to its corresponding possible value set.

For example, in the travel domain we might be able to compute accurate distances for some cities but not for others, which affects the computation of the driving time necessary to get to the airport. If we have a source that geocodes addresses, we can compute an accurate distance and driving time. However, there may be some smaller cities where the data is not available for geocoding. In this case we might want to assume a default value, say half an hour. These two constraints can be combined using a DLIST domain expression so that the system attempts the most accurate method first.

Each variable can also be associated with a preference constraint. Evaluating the preference constraint over the possible values produces the *assigned* value of the variable. Preference constraints are often implemented as functions that impose an ordering on the values of a domain. Preferences are *soft* constraints because they do not affect the consistency of the network only the desirability of the values. An example of a preference in the business travel domain is to choose a hotel closest to the meeting.

4.2 Hierarchical Template Representation

In order to modularize an application and deal with its complexity, we organize the variables and constraints into a hierarchy of templates. This allows us to group the variables into more manageable units and provides an overall organization to the information that is presented to the user. For example, the top-level template of the *Travel Assistant* (shown in Figure 1) includes a set of variables associated with who you are meeting with, when the meeting will occur, and where the meeting will be held. In addition, it has several other variables that can be expanded to fill in more details about the trip, such as how you will get to the meeting and where you will stay once you get there. Each expansion corresponds to a subtemplate that has its own set of variables and constraints and may in turn be composed of lower level subtemplates.

A template is comprised of a name, parameters, variables, constraints, and expansions. The name uniquely identifies the template. The parameters specify the variables that can be passed into or out of a template. The variables have corresponding domain expressions (as defined in Section 4.1.2). The constraints define the set of primitive constraints that are used to compose the domain expressions. And the expansion specifies how a template is elaborated into the appropriate subtemplates based on the assigned value of the expansion variable.

A fragment of the specification of the `Trip` template is shown in Figure 9. The fragment focuses on the `ModeToDestination` decision, the variable definitions involved, the corresponding set of constraints, and the expansion that determines which subtemplate to call based on the value of this variable. The variables involved are `Origin`, `Destination`, `Distance`, and `ModeToDestination`. The `getDistance` constraint computes the distance between the origin and destination addresses by calling the `YahooMap` wrapper. The `selectModeToDest` constraint suggests a value for `ModeToDestination`. If the `Distance` is greater than 200 miles it suggests `Fly`, otherwise `Drive`. Note that this constraint never suggests taking a taxi, but the user may select this choice since it is an option in the `ChooseTemplate` statement and will appear in the interface.

```
template Trip() {
   variables: ...
      Distance = getDistance(Origin, Destination),
      ModeToDestination = selectModeToDest(Distance),
          ...
   constraints:
      getDistance(X,Y) {
         WrapperRemote(
            domain = travel;
            wrapper = yahoomap;
            query = "select distance from yahoomap
                     where origin='X' and
                           destination='Y')}
      selectModeToDest(Distance) {
         if Distance > 200 then Fly else Drive;}
      ...
   expansions:
      choosetemp(ModeToDestination) {
         "Take a Taxi" : Taxi(Origin, Destination)
         "Drive" : Drive(Origin, Destination)
         "Fly" : Fly(Origin, Destination, Distance)}}
```

Figure 9: Fragment of the Trip Template

The hierarchical organization of the templates for the *Travel Assistant* is illustrated in Figure 10. This figure shows the top-level `Trip` template and the subtemplates that can be expanded from this template. There are three subtasks that must be achieved for a trip: getting to the destination, finding an accommodation, and continuing the trip (if

necessary). These decisions are associated with the three *expansion* variables: `ModeToDestination`, `ModeHotel`, and `ModeNext`. Since *all* these subtasks must be achieved for a successful trip, we label the subtask decomposition with an `AND`. Each of these subtasks can be achieved by several alternative means. The figure shows the choices as `OR` branches. For example, the `ModeToDestination` is a variable that can take the values `Fly`, `Drive`, or `Taxi`. For each of these possible values, there is a corresponding subtemplate that will be expanded and used to achieve the subtask. Also, note that the choices for `ModeNext` are recursive instantiations of the `Trip` template. In this way the system can handle trips of any number of legs.

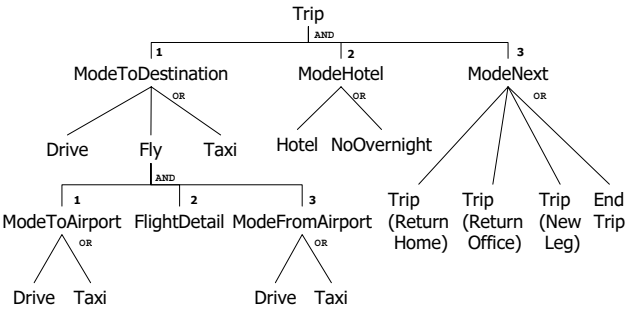


Figure 10: Example of the Hierarchical Organization of Templates

4.3 Constraint Propagation

The domain expressions define a directed graph on the variables. In the current version of the system, the constraint graph must be a acyclic, which means that information flows in one direction. This directionality simplifies the interaction with the user. The structure of the graph is given by the dependency relations present in the domain expressions. Variable Y depends on variable X if the domain expression of Y mentions a constraint that includes variable X. The idea here is that if variable X changes, variable Y may have to be recalculated.

The constraint propagation algorithm proceeds as follows. When a given variable is assigned a value, either directly by the user or by the system, the algorithm recomputes the possible value sets and assigned values of all its dependent variables. This process continues recursively until there are no more changes in the network. More specifically, when a variable X changes its value, the system evaluates the domain expression of each variable Y dependent on X. This may generate a new set of possible values for Y. If this set changes, the preference constraint is evaluated selecting one of the possible values as the new assigned value for Y. If this assigned value is different from the previous one, it causes the system to recompute the values for further downstream variables. Values that have been assigned by the user are always preferred as long as they are consistent with the constraints.

Consider the sample constraint network in Figure 8. First, the constraint that finds the closest airport to the user's home address assigns the value `LAX` to the variable `Departure-Airport`. Then, the constraint `getParkingRate`, which is a call to a web wrapper, fires producing a set of rates for different parking lots.[3] The preference constraint selects terminal parking which is $16.00/day. This value is multiplied by the duration of the trip to compute the `ParkingTotal` of $64 (using the simple local constraint `multiply`). A similar chain of events results in the computation of the `TaxiFare`. Once both the `ParkingTotal` and the `TaxiFare` are computed, the `selectModeToAirport` constraint compares the costs and chooses the cheapest means of transportation, which in this case is to take a `Taxi`.

There are some issues in terms of how aggressively the system should propagate constraints. We have considered four general constraint propagation strategies: propagation on confirmation, propagation within a template, one-level look-ahead propagation, and full propagation.

- *Propagation on confirmation* only starts constraint propagation when the user actually explicitly inputs or approves the values. The advantage of this mechanism is that there is no wasted computation, as every constraint evaluation (which could involve a potentially expensive computation) is guaranteed to be relevant to the user. The disadvantage is that it can be tedious for the user to approve every value suggested for the user and confusing from a user-interface perspective.

- *Propagation within a template* immediately fires constraints as soon as possible but without crossing template boundaries. This approach offers a good tradeoff between computation by the system and input by the user. The system reasons thoroughly within a template assigning values for all the variables in the template. However, it waits for confirmation from the user in the proposed template expansion since a user may choose a different expansion for reasons not represented in the system. This is the default propagation strategy in Heracles.

- *One-level look-ahead propagation* fires constraints both within the current template as well as one-level down from this template. In the cases where the system correctly predicts the users' selections, this approach makes the system appear very responsive. The disadvantage is the increased resource requirements.

- *Full Propagation* evaluates the whole constraint network. The advantage is that this method is that the system can make recommendations based on values propagated up from subtemplates. The disadvantage is that the system may consume significant resources on paths that turn out to be irrelevant.

5. USER INTERFACE DESIGN

We automatically generate the user interface for our information assistants from the hierarchical representation described earlier in this paper. We want to share some of the lessons learned about our human interface design and redesign over the last two years in this section, which is an elaboration of [2].

The greatest challenge in developing the user interface is that it is mixed-initiative: either the user or the system can provide the value for most of the slots. For example,

[3]For simplicity we assume in the example that all domain expressions only involve one constraint.

the email address of a meeting contact can typically be automatically computed (from a user's Outlook address book, or from the company White Pages, for example) – but it can also be hand-typed. The user interface must convey which fields were provided by the system and which fields were provided by the user, and most importantly, must assure the users that they are in control and making steady progress on their task. (Imagine a mixed-initiative user interface where the data flow between fields is unclear or apparently cyclic and the system overwrites data provided by the user!)

Figure 11 shows our initial design, which we based on the spreadsheet metaphor because our data flow from multiple Web sources is similar to the data flow in spreadsheets, and many users are already familiar and comfortable with that metaphor. Unfortunately, there was a key problem with that metaphor – having two fields in the same vertical column (e.g. San Diego and 28 in the screenshot) suggests a relationship where there is none. It was also difficult to visually distinguish subsections (we tried to do that by indenting the first column, but that was too subtle).

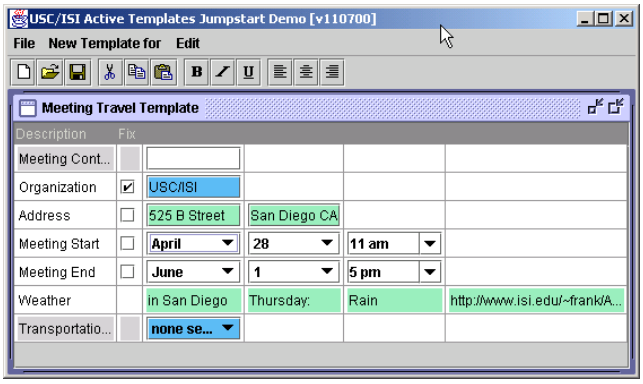


Figure 11: User Interface Based on the Spreadsheet Metaphor

Figure 12 shows our subsequent design. What it shares with the original design is that fields whose values are provided by the user have a blue border, system-provided fields a green border, and fields currently being computed by the system a red border. In the black-and-white screenshot, all fields are green except the first one, provided by the user, which is blue. In addition, every field has a "locked" checkbox – whenever a user enters any field manually it is auto-locked and its value is never changed by the system. For example, in Figure 12 the first field is automatically locked because the value was selected by the user; the system is free to change any other field without asking the user. The user could explicitly ask the system to recompute a value by unchecking the checkbox, which then triggers a recomputation. We carried the idea of "locking" even further by letting the user lock entire groups of fields. For example, in the `Starting At` row of the screen shot, the entire date can be locked via the first checkbox and just the month in the second checkbox.

As Figure 1 demonstrates, we made a number of changes from that attempt but kept that basic design. The color-coding of the fields' origin proved to be valuable and understandable, but the locking checkboxes and their hierarchical nesting to be confusing and space-consuming. Fields are now still implicitly locked when the user manually pro-

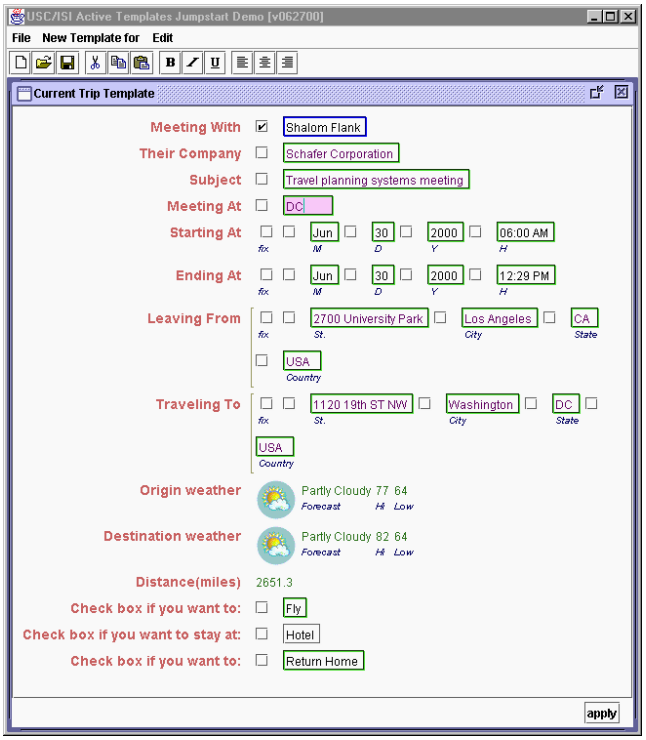


Figure 12: User Interface with Checkboxes

vides a value but no checkbox is shown. Instead, the user can "unlock" the field by selecting "Default" from the drop-down menu. We also no longer automatically expand new sub-sections of the template, even if the system can safely determine the triggering value. For example, even though the system is quite confident you will fly to a meeting more than 2000 miles away, we do not expand the subsection and start displaying flights – it is a correct inference but too distracting and confusing to the user.

One remaining problem with the automatically generated form is that it can be very large since it does not use screen space as efficiently as if a human designer had laid it out by hand. We plan to address these issues by repeating less information in the subsections, packing the fields more tightly, using a more sophisticated layout mechanism, substituting windows for expanding sub-sections, and by possibly moving some of the more auxiliary supporting information (e.g., Weather) to an output-only Web page in the background.

6. THE WORLDINFO ASSISTANT

In Heracles one can build a new information assistant by defining a new hierarchy of templates and a new set of constraints. The templates define the type of information needed in the new assistant, while the set of constraints link together this information. If Web sources are used in the new application, the necessary wrappers for those sources must also be created.

In order to show the generality of our infrastructure, this section describes another example of an information assistant, the *WorldInfo Assistant*. The *WorldInfo Assistant* is an application that brings together a large variety of geo-referenced information. For a user-specified location, it retrieves information like weather, news, holidays, maps, air-

ports, geospatial points of interest, etc. Where available it also retrieves satellite images and plots georeferenced information on them. Georeferenced information includes geographical features such as rivers, lakes, deserts, etc., and a variety of man-made features, such as hospitals, bridges, factories, etc. In addition to selecting values from the slots, the user can navigate through the satellite images and maps by recentering, zooming, and panning.

A variety of Web sources are used to provide the information necessary for this application. For example, georeferenced points of interest are retrieved from USGS.gov (United States Geological Survey) and Mapblast.com, images are retrieved from TerraServer.com and SpaceImaging.com, weather data is retrieved from weather.Yahoo.com and Weatherbase.com, holiday information from Holidayfestival.com, etc.

The top-level template and weather subtemplate of the *WorldInfo Assistant* is show in Figure 13. The template allows the user to select the location and date of interest. In the figure the user has chosen Asia as the region, China as the country, Hong Kong as the city, and May 2001 as the date. Based on this input, the application retrieves relevant information related to this location and date. For example, it retrieves the current weather, a five day forecast (Yahoo! weather), and the monthly averages for May (Weatherbase).

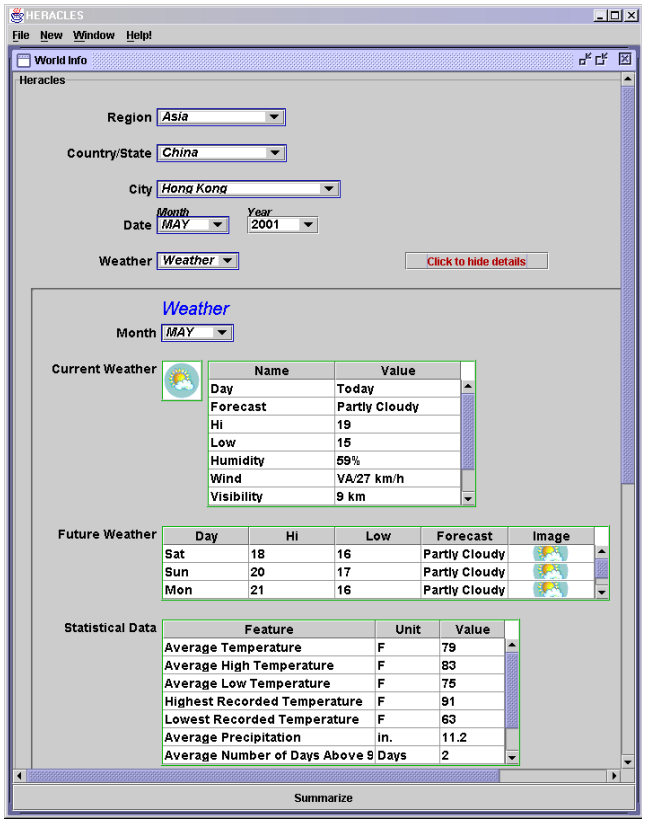


Figure 13: Top-level Template for the *WorldInfo Assistant*

As an example of integrated geospatial imagery and points, Figure 14 shows a satellite image and map of the Washington, D.C. area. The resolution of the image is `8` meters, and the 12 points of interest plotted on this image are `bridges`. The template shows an image and a map of the same area at roughly the same resolution.

The constraint reasoning system provides a coordinated view of the different types of information. The imagery, maps, and points are all linked to the center of the image, which is defined by variables for the latitude and longitude. If the user then clicks anywhere on the current image, the system updates the values for the image center, which triggers updates to the image, map, and points. Similarly, if the resolution is changed, it would trigger the system to retrieve a new set of points and change the resolution of the image and map.

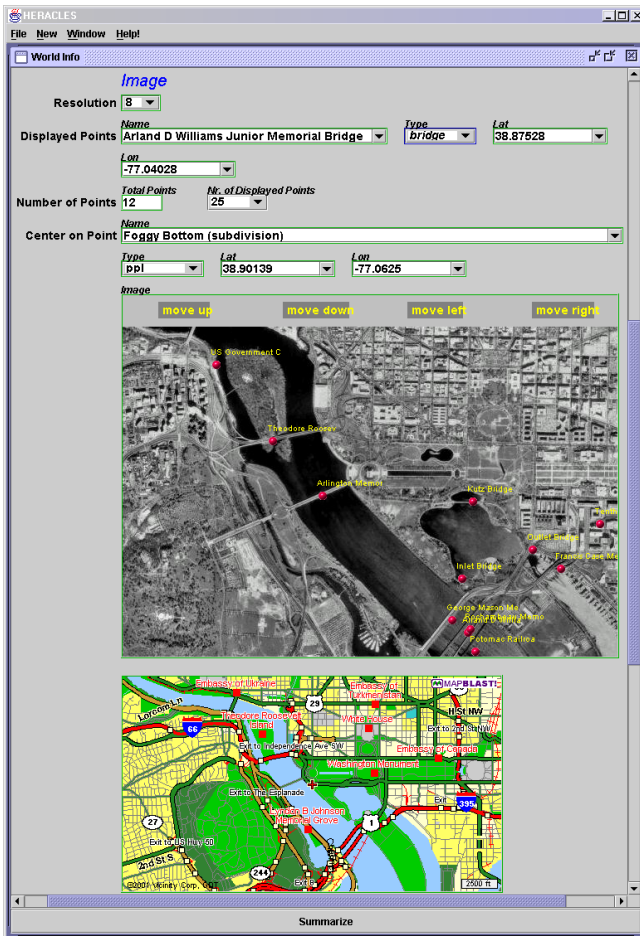


Figure 14: Integrated Satellite Image, Points of Interest, and Map

7. RELATED WORK

There has been a lot of research in the area of constraint programming [12, 11, 10], but not much attention has been paid to the interplay between information gathering, constraint propagation, and user interaction, which is the focus of Heracles. Bressan and Goh [1] have applied constraint reasoning technology to information integration. However, the specific problem they are addressing is quite different from ours. They use constraint logic programming to find relevant sources and construct an evaluation plan for a user query. In our system the relevant sources have already been

identified. We focus on user interactivity, flexible integration of information gathering and other computational constraints in an uniform framework, and on information propagation in service of the user tasks. Petrie et al. [9] developed an architecture for constraint-based agents in engineering design applications. We share the idea of combining multiple reasoning systems. However, their focus is on distributed constraint management issues, such as adding and removing constraints, and in recording the rationale for the design choices and inconsistencies. Real-time information gathering is not considered in their system.

The growth and changes to the constraint network that occur in Heracles as a result of the hierarchical expansion of templates can be seen as a form of dynamic constraint satisfaction [6]. In dynamic constraint satisfaction the variables and constraints present in the network are allowed to change with time. Heracles imposes a structure to these changes as they correspond to meaningful units in the application: the templates.

Lamma et al. [4] propose a framework for interactive constraint satisfaction problems (ICSP) in which the acquisition of values for each variable is interleaved with the enforcement of constraints. The interactive behavior of our constraint reasoner also can be seen as a form of ICSP. However, our approach includes a notion of hierarchical decomposition and task orientation. Their application domain is on visual object recognition, while our focus is on information integration

8. CONCLUSION

In this paper we have described a general framework for building information assistants and presented two compelling applications of this framework. The combination of real-time, structured access to Web sources and hierarchical constraint propagation provides a powerful paradigm for building new web-based information assistants. Both example applications that we presented go beyond what is currently available on the Web. And we believe there is a wide variety of other interesting information assistants that can be built using the same framework. Other possible assistants include a real estate assistant for locating and buying a house and a financial planning assistant for planning investments, major purchases and retirement.

There are many interesting research problems associated with building information assistants. We are currently extending the underlying constraint reasoning system to provide support for temporal constraints and resources. We are also investigating a variety of integration issues that arise, including efficient distributed spatial data integration. We are building a new version of the graphical user interface that runs in a browser, but still provides the real-time update of the information. Finally, we are exploring the addition of monitoring agents for tracking plans once they have been created, such as an agent for tracking flight schedules for cancellations or flight delays.

Acknowledgements

We gratefully acknowledge the contributions of both Doug Dyer and Shalom Flank to this work. Doug Dyer provided the original idea of an interactive template-based travel planner based on a spreadsheet paradigm. Shalom Flank worked closely with us on the original design of this system and provided detailed feedback and ideas throughout the implementation process.

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-1-0504, in part by the Rome Laboratory of the Air Force Systems Command and DARPA under contract number F30602-98-2-0109, in part by the United States Air Force under contract number F49620-98-1-0046, and in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

9. REFERENCES

[1] Stéphane Bressan and Cheng Hian Goh. Semantic integration of disparate information sources over the internet using constraint propagation. In *Workshop on Constraint Reasoning on the Internet*, 1997.

[2] Martin Frank, Maria Muslea, Jean Oh, Steve Minton, and Craig Knoblock. An intelligent user interface for mixed-initiative multi-source travel. In *Proceedings of the ACM International Conference on Intelligent User Interfaces*, 2001.

[3] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *Data Engineering Bulletin*, 23(4), 2000.

[4] Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi. Constraint propagation and value acquisition: why we should do it interactively. In *Proceedings of IJCAI-99*, 1999.

[5] Kristina Lerman and Steven Minton. Learning the common structure of data. In *Proceedings of AAAI-00*, 2000.

[6] Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, 1990.

[7] Ion Muslea, Steven Minton, and Craig A. Knoblock. Selective sampling with redundant views. In *Proceedings of AAAI-00*, 2000.

[8] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2), 2001.

[9] Charles Petrie, Heecheol Jeon, and Mark R. Cutkosky. Combining constraint propagation and backtracking for distributed engineering. In *Workshop on Constraint Reasoning on the Internet*, 1997.

[10] Vijay Saraswat and Pascal van Hentenryck, editors. *Principles and Practice of Constraint Programming.* MIT Press, Cambridge, MA, 1995.

[11] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.

[12] van Hentenryck. *Constraint Staisfaction in Logic Programming.* MIT Press, 1989.