Conditional Constraint Networks for Interleaved Planning and Information Gathering

José Luis Ambite, Craig A. Knoblock, and Maria Muslea,
University of Southern California, Information Sciences Institute

Steven Minton, *Fetch Technologies*

The Heracles II framework allows planning in a mixed-initiative fashion, where the user can explore alternatives and override the system suggestions as needed.

For any activity, a wealth of information is available through public and private networks. Unfortunately, such information is distributed among many sites, with different data formats, schemas, and semantics. Moreover, information access per se is of limited value. What is needed is a system that integrates and structures diverse information to support user tasks and goals. The system must gather the relevant information, evaluate trade-offs, and suggest courses of action to the user.

For example, consider travel planning. Numerous sites provide relevant information, such as

- flight schedules and fares (for example, www.orbitz.com),
- hotel locations and rates (for example, www.itn.com),
- car rental information (for example, www.hertz.com),
- weather information (for example, http://weather.yahoo.com),
- maps and routes (for example, www.mapquest.com), and
- airport parking rates (for example, www.airwise.com).

A travel planner must integrate this information with user preferences, such as for airlines or flying times (for example, avoid red-eye flights); cost constraints; and company policies, such as allowable airlines, expense caps, and per-diem or mileage reimbursement rates. Although the user can visit these sites and deal with all the constraints and preferences manually, this is extremely tedious, error prone, and time consuming. A system that queries the remote sites, accesses local information, and enforces the constraints and preferences is much more desirable.

The main requirements for such a system are planning support and interactivity. To support planning, the system must

1. gather and integrate the information in a coherent structure that captures the tasks needed for the application domain,
2. evaluate trade-offs and select among alternative courses of action, and
3. let the user explore and override system suggestions.

To provide flexible interaction, the system must

1. let the user input data or change choices anytime during planning, and
2. handle information sources that return results asynchronously.

To address these challenges, we have developed Heracles II, a framework for mixed-initiative planning and information gathering. Heracles II maps the hierarchical task structure of the planning domain into a *conditional constraint network*.[1] It also ensures correct constraint propagation in the presence of cycles, user interaction, and asynchronous sources. We have

1541-1672/05/$20.00 © 2005 IEEE
Published by the IEEE Computer Society

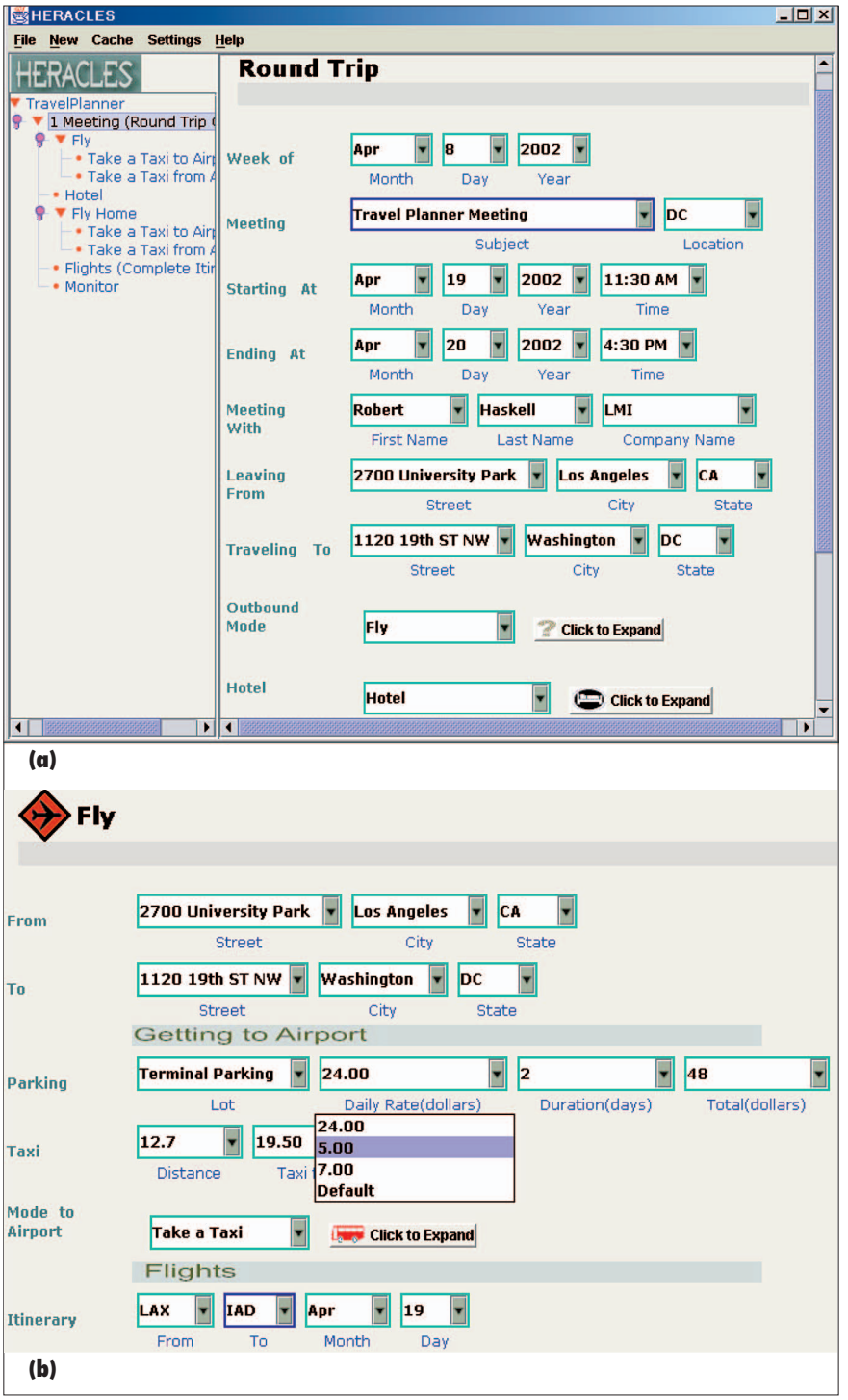

Figure 1. Travel planner templates: (a) the top-level template; (b) the Fly template.

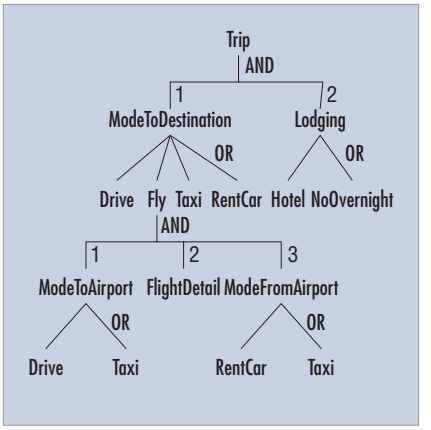

Figure 2. The hierarchical organization of templates for the travel planner.

applied the Heracles II framework to several domains including travel planning and geospatial data integration.

The original Heracles

Our initial approach, the Heracles framework, models each piece of information as a variable in a constraint network and uses constraints to integrate such information.[2] The resulting constraint network provides a coherent view of user activities and captures the relevant information and user preferences.

Any nontrivial user activity involves a very large number of variables and constraints. So, Heracles partitions the network hierarchically corresponding to the task structure of the application domain, in a manner similar to hierarchical-task-network planning.[3] The application designer groups variables and constraints related to a distinct task into a package we call a *template*.

For example, consider the templates in Figure 1. Figure 1a shows the top-level template of our travel planner, which includes the most important information about the trip such as the origin, destination, and dates of travel. The next layer of decisions includes

- the alternative means of transportation, such as flying, taking a train, renting a car, driving the user's own car, or taking a taxi; and
- choices of accommodation at the destination.

Figure 1a shows that the system suggests flying. The variables and constraints related to flying constitute another template (see Figure 1b). Heracles further decomposes each template into more specific subtemplates. For example, once the user chooses flying as the main transportation mode, the system must evaluate how to get to the airport: by taxi, by driving a car and parking it at the airport, and so on. Figure 2 shows the task/template hierarchy for the travel planner.

In Heracles, the information gathered by the system or input by the user is propagated automatically in the constraint network. Heracles also lets the user override the values the system suggests. For example, Figure 3 shows the changes to the Taxi template when the user selects a different departure airport in the Round Trip Flights template. Figure 3a

shows the information relevant to going to the airport by taxi, including cost, time estimate, map, and route from the user's location (the University of Southern California) to Los Angeles International Airport (LAX). Figure 3b shows the user selecting Long Beach Airport (LGB) instead of LAX. Finally, Figure 3c shows the Taxi template reflecting this change, including the new cost, times, and maps.

Heracles shows that a constraint-based approach is well suited to support mixed-initiative planning where user interaction and asynchronous information gathering are central requirements. However, the original Heracles has two serious limitations.

First, the template selection mechanism is hard-coded into the implementation and not integrated with the constraint network. To perform template selection in Heracles, a procedure inspects the values of *expansion variables*. For example, the ModeToDestination variable in Figure 2 (labeled Outbound Mode in Figure 1a) is an expansion variable that can take the values Fly, Drive, RentCar, or Taxi. Whenever an expansion variable in a parent template is set, Heracles adds the corresponding child template to the constraint network and removes the alternative (child) templates. This hard-coded behavior means that most of the logic for selecting among templates must appear in the parent template, even if such information logically belongs to the child templates. This diminishes template modularity and tends to create large, monolithic templates. The problem becomes ever more acute the deeper the task/template hierarchy.

The second limitation is that Heracles cannot handle cycles in the constraint network. So, the template designer must specify the constraints sometimes in an unintuitive way or forego some lines of reasoning altogether.

Heracles II

Heracles II provides solutions to these limitations while preserving the advantages of Heracles. First, Heracles II is uniformly represented as a conditional constraint network, and template selection follows naturally from the evaluation of *activity constraints*. To ensure that Heracles II always considers all information relevant to the choice of tasks/templates, we introduce the concept of a *core network* that is always active. Second, we designed a new constraint propagation algorithm that can handle cyclical networks in the presence of user interaction and asynchronous sources.

For a comparison of Heracles II to other approaches to the problem of combining planning, information gathering, and user interaction, see the sidebar, "Related Work in Interactive Planning."

Conditional constraint networks and hierarchical planning

As the complexity of a planning domain grows, designing and maintaining a monolithic network becomes infeasible. Similarly, presenting a large network for the user to interact with quickly becomes unmanageable and confusing. Templates help Heracles II avoid these problems.

Template specification. A template consists of a name, arguments, variables, constraints, and expansions. The name uniquely identifies the template. The arguments specify the input variables, which receive values from other templates or from the user, and the output variables, whose values are used in other templates. All other variables are internal to the template. Each expansion specifies how a template is elaborated into the appropriate subtemplates on the basis of the value of the expansion variable.

Figure 4 shows a fragment of the specification of the Fly template focusing on the ModeToAirport decision. The input variables include OriginAddress and DepartureDate. The output variables are SelectedFlight and FlyCost. The selectModeToAirport constraint selects the cheapest mode of transportation to the airport.

Heracles II receives as input a set of declarative XML template definitions. Figure 4 shows some of the main constructs. The values of variables can also be XML objects that are processed using XQuery. A template also includes a declarative specification of the user interface. The constraint network can control which widgets appear in the interface depending on runtime values. A full description of the XML syntax and the constraint-based control of the interface is outside the scope of this article.

Template hierarchy. The templates are organized hierarchically to model the task structure of the planning domain, to help the user understand the planning process, and to facilitate presentation of the information. Figure 2 shows the (simplified) hierarchy of our travel planner. Planning a successful trip requires achieving two subtasks: determining how to get to the destination and choosing an accommodation. These decisions are associated with two expansion variables in the travel-planning hierarchy: ModeToDestination and Lodging. Because both subtasks must be achieved, we label the subtask decomposition with an AND. Several alternative means exist for achieving each subtask. The figure shows the choices as OR branches. For example, recall the ModeToDestination expansion variable with the possible values of Fly, Drive, RentCar, or Taxi. The Fly template further decomposes into three subtemplates

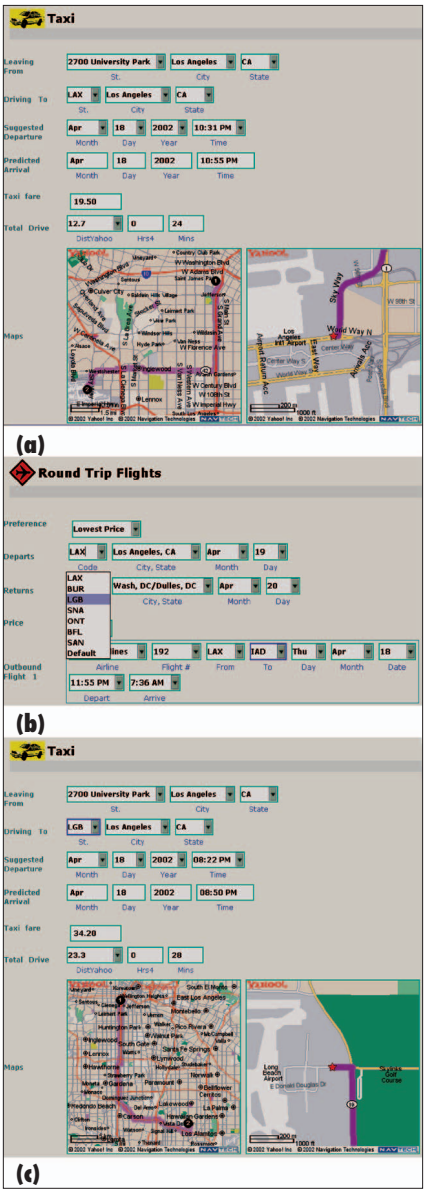

Figure 3. User interaction and constraint propagation for the travel planner: (a) the Taxi template; (b) the user changes values in Round Trip Flights; (c) the changes propagate to the Taxi template. Maps © 2002 NAVTEQ All rights reserved.

Related Work in Interactive Planning

Here we compare Heracles to other approaches for combining planning, information gathering, and user interaction.

Travel planning

The SmartClients system uses constraint satisfaction to support the user in planning and information gathering.[1] The SmartClients system has also been applied to travel planning. When the user inputs a trip's origin, destination, and dates, SmartClients automatically compiles a constraint network to explore the possible trips. To function, SmartClients must be able to access a remote database with flight information such as Sabre, so that it can retrieve all the flights between the selected cities in the given dates and populate the appropriate variables and constraints in the network. Once the network has been initialized by calling such a database, SmartClients searches for a solution trip that satisfies all the constraints, using classical constraint satisfaction algorithms.

Heracles (see the main article) and SmartClients have several crucial differences. First, SmartClients retrieves all information beforehand, assuming that the size of the relevant sections of the flight database can be compactly encoded and efficiently transmitted to the user client. However, this approach cannot scale to larger problems that incorporate a broader range of information sources. Retrieving all the relevant information, such as flight schedules, hotel locations and rates, rental cars, and maps, before the search starts is not feasible. In contrast, Heracles accesses the external sources only during planning, for only those values that are already part of a consistent partial solution. The more focused search of Heracles is more scalable and allows for an arbitrary exploration of the information space.

Second, the user can interact with Heracles at any point during planning and can change the values of variables throughout the network, resulting in arbitrary retrievals of external information. In SmartClients, the domains of the variables in the constraint network are fixed initially, so the user can explore only solutions within such a space (or must restart the whole constraint network construction and search process).

Finally, SmartClients performs full constraint satisfaction, so it finds optimal solutions according to the user preferences. However, it does this in the smaller search space defined at initialization time. Heracles performs constraint propagation and does not attempt to find an optimal solution. However, in Heracles the user can explore the full solution space with the latest information obtained in real time from external sources and understand the different trade-offs. In our experience, letting the user interactively guide the process toward a desirable plan yields better results.

The Trip-planner agent framework employs a different approach to collaborative planning and information gathering.[2] Trip-planner is also based on a constraint network that integrates different sources related to travel planning. The user specifies his or her preferences at the start of planning. During the constraint network evaluation, the system calls different sources, guided by the user preferences. It consults the user again at predefined points during planning. For example, after Trip-planner has found the 10 cheapest flights, it prompts the user to select one. However, the user cannot interact with the constraint reasoner at any point during constraint evaluation, unlike with Heracles.

Travel Web sites such as Expedia, Travelocity, and Orbitz provide good support for information gathering and user interaction. However, they lack support for an integrated view of the (travel) planning process that satisfies complex constraints and preferences of the user.

As Table A shows, other approaches do not meet our requirements for a uniform framework that combines user interaction, information gathering, and planning and constraint-reasoning capabilities.

Other interactive-planning research

The research on collaborative[3] and mixed-initiative[4–6] planning is related in spirit to our research on Heracles. A central

that handle ground transportation at the origin and destination airports and the flight details.

Conditional constraint network. Heracles II reads the set of declarative template specifications for a given application and automatically constructs a conditional constraint network based on them.

Figure 5 shows a fragment of the constraint network for travel planning that addresses the selection of the method of travel from the user's initial location to the airport. The choices under consideration are driving a car (which implies parking it at the airport during the trip) or taking a taxi. Figure 5a shows the constraint network that the original Heracles system would evaluate to make this decision. This network would need to be in the same template although some variables naturally should appear in subtemplates. Figure 5b shows the same network in Heracles II. The variables and constraints are partitioned across several templates in a more modular way that corresponds more closely with the task structure of the domain.

Variables. Each distinct piece of information in an application is represented as a variable in the constraint network. These values are set by the system by constraint propagation or directly by the user from the graphical interface. In a conditional constraint network, each variable has not only a value, as in the classical case, but also an activity status. If a variable is inactive it does not participate in the network. Figure 5 shows the variables as dark rectangles and the values as white rectangles next to them. For example, DepartureAirport has the value LAX (Los Angeles International), which the system assigns because LAX is the closest airport to the user's address.

Classical constraints. A constraint is a subset of the Cartesian product of the domains of the participating variables. A constraint is a computable component that can be implemented by a local table look-up, by the computation of a local function, by retrieving a set of tuples from a remote source, or by calling an arbitrary external program.

Figure 5 shows the constraints as rounded rectangles. For example, the computeDuration constraint involves three variables (DepartureDate, ReturnDate, and Duration) and is implemented by a function that computes the duration of a trip given the departure and return dates. To implement the getParkingRate constraint, Heracles II calls a wrapper that accesses a Web site containing parking rates for US airports (www.airwise.com).

Table A. A comparison of travel-planning approaches.

	Planning/constraint satisfaction programming	User interaction	Information gathering
Heracles, Heracles II	✓	✓	✓
SmartClients	✓	✓	
Trip-planner	✓		✓
Expedia and other travel Web sites		✓	✓

goal in these systems is to help the user develop a plan interactively. Although each system employs different planning technology, in each case the user interactively modifies the planning process. In James Allen and George Ferguson's system, the user can perform actions such as refining a goal or rejecting an option.[3] Karen Myers and her colleagues devised a framework where the user can drop or modify constraints and tasks.[4] Manuela Veloso and her colleagues developed a system that uses a case-based planner to propose modifications to an initial plan.[5]

Unlike these systems, our Heracles framework uses a predefined set of templates that define the space of possible plans, and it uses constraint propagation instead of a general-purpose planner to reason about plans. This supports combined planning, user interaction, and information gathering, unlike the other systems, which focus only on integrating user interaction and planning.

Evelina Lamma and her colleagues propose a framework for *interactive constraint satisfaction problems* (ICSP) that interleaves the acquisition of values for each variable with constraint enforcement.[7] The interactive behavior of our constraint reasoner can be seen as a form of ICSP. However, our approach includes a notion of hierarchical decomposition and task orientation.

References

1. M. Torrens, B. Faltings, and P. Pu, "Smart Clients: Constraint Satisfaction as a Paradigm for Scalable Intelligent Information Systems," *Constraints*, vol. 7, no. 1, 2002, pp. 49–69.
2. A. Homb et al., "Trip-Planner: An Agent Framework for Collaborative Trip Planning," *Proc. AAAI-99 Workshop Mixed-Initiative Intelligence*, AAAI Press, 1999.
3. J. Allen and G. Ferguson, "Human-Machine Collaborative Planning," *Proc. 3rd Int'l NASA Workshop Planning and Scheduling for Space*, 2002.
4. K.L. Myers et al., "A Mixed-Initiative Framework for Robust Plan Sketching," *Proc. 13th Int'l Conf. Automated Planning and Scheduling* (ICAPS 2003), AAAI Press, 2003, pp. 254–265.
5. M.M. Veloso, A.M. Mulvehill, and M.T. Cox, "Rationale-Supported Mixed-Initiative Case-Based Planning," *Proc. 9th Conf. Innovative Applications of Artificial Intelligence* (IAAI 97), AAAI Press, 1997, pp. 1072–1077.
6. M.H. Burstein and D.V. McDermott, "Issues in the Development of Human-Computer Mixed-Initiative Planning Systems," *Cognitive Technology: In Search of a Humane Interface*, B. Gorayska and J. Mey, eds., Elsevier Science, 1996, pp. 285–303.
7. E. Lamma et al., "Constraint Propagation and Value Acquisition: Why We Should Do It Interactively," *Proc. 16th Int'l Joint Conf. Artificial Intelligence* (IJCAI 99), vol. 1, Morgan Kaufmann, 1999, pp. 468–477.

In Heracles and Heracles II, most constraints have an implementation in only one direction. For example, we can find out the parking cost at a given airport, but not which airports have parking lots that cost less than $7 a day. This is one reason why Heracles performs only constraint propagation instead of full constraint satisfaction.

Activity constraints. An activity constraint controls the activity status of a variable given the values of other variables. For example, consider an activity constraint $ac(v_1, ..., v_{n-1}, v_n)$ that is computed by the rule

$$(v_1 = k_1) \wedge ... \wedge (v_{n-1} = k_{n-1}) \rightarrow active(v_n)$$

That is, the activity constraint will make variable v_n active whenever the variables $[v_1, ..., v_{n-1}]$ take the values $[k_1, ..., k_{n-1}]$, respectively.

Template selection using activity constraints. Heracles II uses activity constraints and expansion variables to select among alternative templates. It automatically generates activity constraints based on the expansion section of each template specification. For each expansion variable, it adds an activity constraint that acts as a multiplexor, making the selected template active and making the alternative templates inactive. To achieve this effect, Heracles II uses no special mechanism other than the normal evaluation of the activity constraints in the conditional network.

For example, in Figure 5b, the expansion variable ModeToAirport in the Fly template selects between the Drive and Taxi subtemplates. The activity constraint, represented in the figure by the dashed lines from the ModeToAirport variable to the Drive and Taxi subtemplates, is

$$\text{Fly.ModeToAirport} = \text{"Taxi"} \rightarrow \text{active(Taxi.TaxiFare)} \wedge \text{active(Taxi.Distance)} \wedge \neg\text{active(Drive.Map)} \wedge \neg\text{active(Drive.Directions)}$$

The core network. To ensure that template selection is responsive to changing user inputs or new information from asynchronous sources, Heracles II maintains the subset of the network that affects the computation of the expansion variables, called the *core network*, always active. Otherwise, when a template becomes inactive its variables cannot affect the rest of the network and the choices would lack relevant information.

For example, in Figure 5b, all variables and constraints, except Map, Directions, and getMapDirections, belong to the core network because they contribute to the computation of the ModeToAirport expansion variable. This is why the activity constraint we just mentioned does

```
<template name="Fly">
    <args> <in>OriginAddress</in> <in>DepartureDate</in> ...
           <out>SelectedFlight<out/> <out>FlyCost</out></args>
    <vars>
        <var name="OriginAddress"/> <var name="FlyCost"/>
        <var name="TaxiFare"/> <var name="ParkingCost"/>
        <var name="ModeToAirport"/> ... </vars>
    <constraints>
        <constraint name="selectModeToAirport" type="XQueryConstraint">
            <args> <in>TaxiFare</in> <in>ParkingCost</in>
                   <out>ModeToAirport</out> </args>
            <output><xquery><![cdata[
                <row>  <TaxiFare>{$TaxiFare}</TaxiFare>
                       <ParkingCost>{$ParkingCost}</ParkingCost>
                       <ModeToAirport>{   if ($TaxiFare >= $ParkingCost)
                                          then "Drive"
                                          else "Taxi"}</ModeToAirport>
                </row>]]></xquery></output></constraint> ... </constraints>
    <expansions>
        <expansion><var name="ModeToAirport"/>
            <template-call name="Drive" printname="Drive and Park">
                <in>DepartureAirport</in> <in>Duration</in>
                <out>ParkingCost</out> </template-call>
            <template-call name="Taxi" printname="Take a Taxi">
                <in>OriginAddress</in> <in>DepartureAirport</in>
                <out>Taxifare</out> </template-call>
    </expansion> ... </expansions> </template>
```

Figure 4. A fragment of the Fly template specification.

not make inactive the core variables **ParkingRate** and **ParkingTotal** of the **Drive** template. This is also why we did not include the case for **Fly.ModeToAirport = Drive** in the activity constraint, because all the variables of the **Taxi** template are core variables. We call those variables not in the core network *information variables* because their values do not affect task choices but provide additional information to the user.

Heracles II determines the core network automatically by reachability analysis. It searches the constraint network, starting from the expansion variables and traversing constraints from output to input variables until it reaches the "source" variables (that is, those variables that are not the output of any constraint and must be set by the user). The variables and constraints visited in this search constitute the core network.

Analysis. To understand the savings that the conditional constraint network and the template selection mechanism provide, consider a task hierarchy of depth d where each task has a single decision point (expansion variable) with two possible alternatives (subtemplates). This hierarchy induces a binary **OR** tree with $2^d - 1$ nodes. Further assume that each template has c core constraints and i information constraints. After Heracles II evaluates the core network and decides on the top-level choice, all the information variables and constraints in one of the top two subtrees remain inactive. Moreover, this behavior repeats at each level of the selected subtree. So, the only information constraints (and variables) that become active are those of the selected course of action, a total of $(d + 1)i$. This saves the evaluation of an exponential number, $(2^d - d)i$, of information constraints. In practice, the core network is often a small subset of the constraint network. So, the Heracles II template selection mechanism considerably reduces constraint evaluation effort.

Interactive constraint propagation

The basic constraint propagation algorithm proceeds as follows. When either the system or the user assigns a value to a variable, the algorithm fires all constraints that have that variable as an input. This might cause the output variables to change their value (or activity status), and the process continues recursively until no more changes occur in the network.

For example, consider the network in Figure 5. First, the constraint that finds the airport closest to the user's home address assigns the value **LAX** to the variable **DepartureAirport**. Then, the constraint **getParkingRate**, which is a call to a Web wrapper, produces the rate $16.00 a day. The algorithm multiplies this value by the duration of the trip to compute the **ParkingTotal** of $64.00 (using the simple local constraint **multiply**). A similar chain of events results in the computation of **TaxiFare**, based on the distance between the origin address and the airport. Once the algorithm has computed **ParkingTotal** and **TaxiFare**, the **selectModeToAirport** constraint compares the costs and chooses the cheapest means of transportation, which in this case is a taxi.

Constraint propagation in Heracles II can be seen as following a cyclic directed graph. Since the user can change the value of a variable in the network at any time and remote sources return data asynchronously, Heracles II must take special care to prevent infinite loops, to ensure that all the appropriate constraints are fired and the values propagated, and to disregard obsolete values.

To address these requirements, Heracles II's constraint propagation algorithm includes a time-stamping mechanism. The algorithm annotates each variable with a *user-time*, an integer incremented every time the user inputs a new value or changes a value. In addition, it annotates each variable with the set of variables that have contributed to its value (that is, those variables that were visited in the chain of constraints that set the variable in the current user-time). This *visited set* is necessary to prevent cycles. Both these annotations are propagated as the algorithm evaluates the constraints along with the actual values assigned to the variables.

The algorithm for time-stamped constraint propagation follows these rules:

- R1 Whenever the user changes some value in the interface, the user-time of the corresponding variable is incremented and the visited set is set to empty.
- R2 A constraint fires whenever any of its inputs changes.
- R3 When a constraint fires, each output variable inherits the latest user-time of the input variables and gets an updated visited set consisting of the union of the input variables and their visited sets.

R4 A constraint blocks (does not fire) if there exists a variable V_o in the constraint's outputs and a variable V_i in the constraint's inputs such that *user-time*(V_o) ≥ *user-time*(V_i) and $V_o \in$ *visited*(V_i).

Figure 6 shows sample simulations of the constraint propagation algorithm. Constraints are shown as rectangles and variables as simple nodes. Arcs denote the direction of constraint propagation. Each variable annotation has the syntax "simulation-step) user-time / visited variables [value". For example, the annotation "1) 1 / ∅ [LA" of variable v_1 means that at simulation step 1 the user-time was 1, no other variables were used in the computation of its value (that is, the user set the value), and its value was "LA" (Los Angeles).

Figure 6a shows a simulation of constraint propagation in a cyclic network that picks the geocoordinates of interest (v_3) in one of our applications. The network has three constraints:

- C1, which given the name of a city (v_1), produces the geocoordinates of the city center (v_2);
- C2, which copies the geocoordinates into v_3; and
- C3, which finds the closest city (v_1) to the given latitude and longitude (v_3).

This network aims to give the user flexibility in selecting a geopoint of interest by either entering a city (v_1) and getting the city center as the coordinates of interest (v_3) or entering a latitude and longitude (v_3) and determining the closest city (v_1). The intention of C2 is to copy its input to its output only when doing so is consistent. The time-stamping algorithm helps enforce those semantics even though the implementation of C2 can be a simple copy.

The simulation starts with the user entering LA in v_1. The algorithm then propagates the corresponding city center coordinates (34N118W) to v_3 (step 3). Because v_3 has a new value, C3 is scheduled to fire. However, rule R4 blocks such firing because v_1 had already been visited for the computation of v_3 during the same user-time phase (step 4). This prevents the possible infinite cycle of propagation. With such state of the network, the user now inputs the latitude and longitude 40N70W in v_3 (step 5). C3 fires normally because the user input resets the visited variables of v_3. Propagation continues, setting v_1 to New York (NY) and v_2 to its city center (40N73W), but R4 prevents the firing of C2, again blocking a possible infinite cycle.

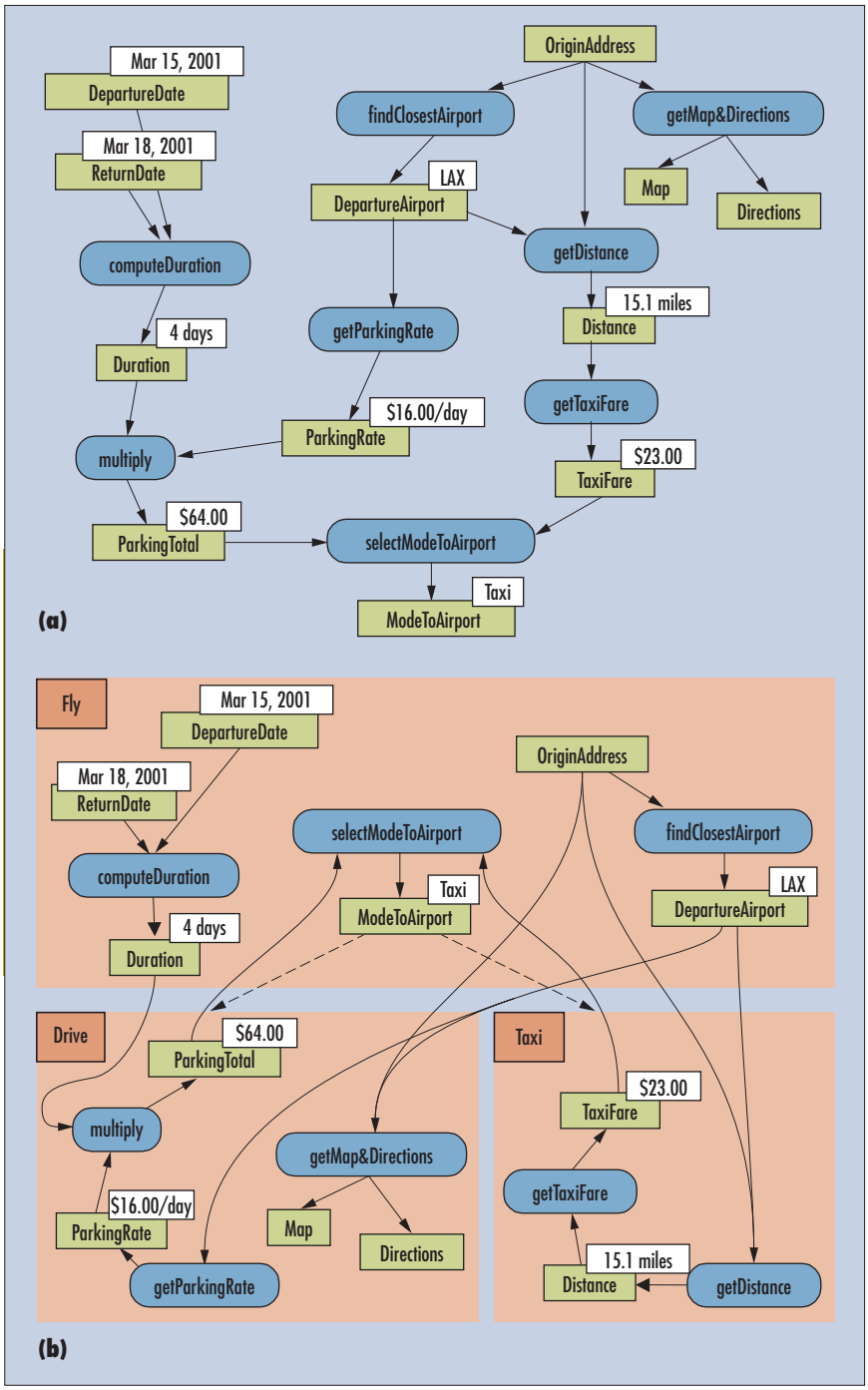

Figure 5. Driving versus taking a taxi: (a) The original Heracles evaluates a flat constraint network; (b) Heracles II evaluates a hierarchical conditional constraint network.

The algorithm requires both the user-time and visited-set annotations. Consider the acyclic network in Figure 6b, where constraint propagation reaches a given variable through different paths. When the user inputs a value in v_1, C1 fires, and the values of both v_2 and v_3 change. This causes both C2 and C3 to fire. So, by step 3, v_5 obtains a new value. However, by step 5, the updated value of v_4 makes C3 fire again. This correctly changes v_5 a second time even though v_5 has the same user-time (1) as v_3 and v_4 (the inputs

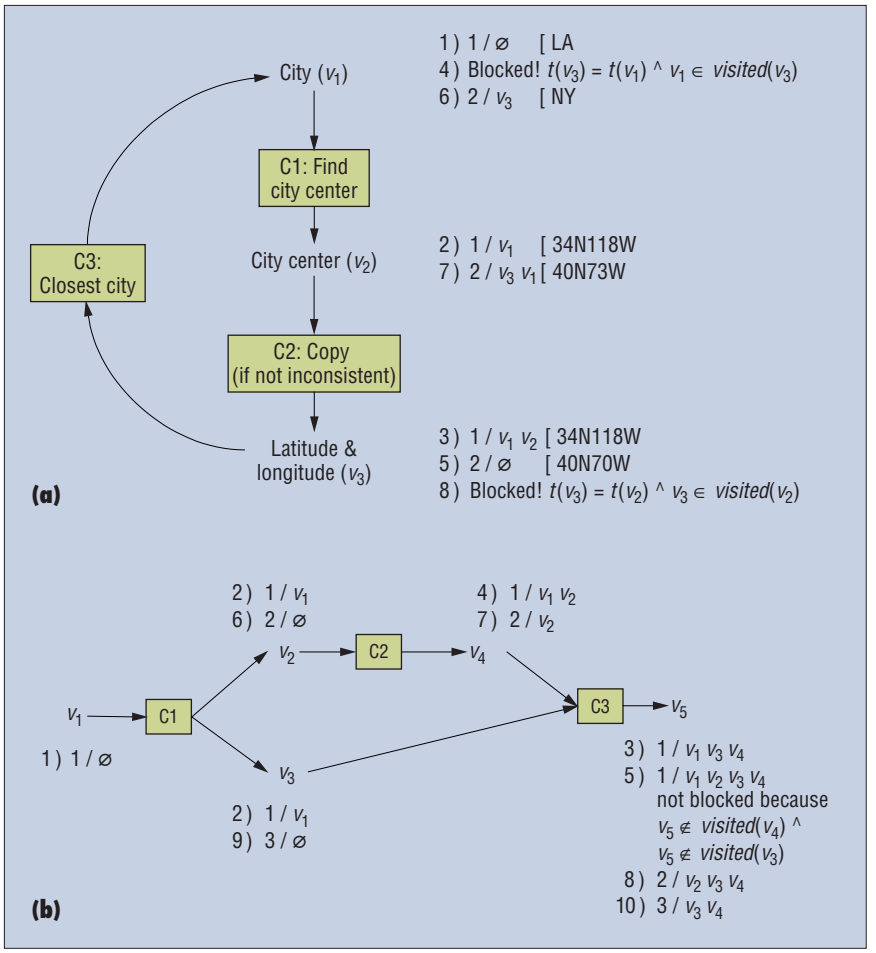

Figure 6. Interactive constraint propagation: (a) a cyclic network; (b) an acyclic network (race condition).

of C3). Since $v_5 \notin visited(v_4)$ and $v_5 \notin visited(v_3)$, the propagation through C3 is not blocked. Only propagating the user-time or only keeping track of visited variables would not produce the correct propagation. Further user inputs occur at steps 6 and 9 and produce correct propagation.

HeraclesMaps: Geospatial-data integration

To build a new application in Heracles, the designer just needs to specify a set of templates that represent the hierarchical task structure of the application along with constraints that access the relevant sources. To facilitate development, our framework includes predefined constraint types for the most common types of sources, including wrappers, databases, and XML files, and of data manipulation operations, including XQuery processing and arbitrary Java functions.

We used the Heracles II framework to rapidly build and maintain HeraclesMAPS, an application that integrates multiple geospatial and online sources. HeraclesMAPS lets a user gather information about a specific area, including satellite images, maps, topography, hydrography, transportation networks, points of interest, airport and seaport data, and weather information. HeraclesMAPS has been deployed to US Special Operations Forces.

In the top-level template of HeraclesMAPS, the user selects a location by drilling down to a continent, country, and city (populated place) or by entering latitude and longitude. This initial location triggers constraint computation. The constraint network keeps all the information consistent. For example, if the user recenters a map in a subtemplate, the application will recompute all the related maps, images, and vector and point data to match the new area of interest. Heracles II issues the corresponding information-gathering constraints that retrieve such data as the user navigates the information space.

Figure 7 shows two sample templates of HeraclesMAPS. Figure 7a shows a satellite image and the corresponding map of the port of Umm Qasr in Iraq. Vector data of transportation features such as roads and railroads appears overlaid on the image. Figure 7b shows an elevation map (the upper-left image), the area visible (yellow) and not visible (red) from a given point (upper right), and the altitude profile (lower right) between two given points (from the origin specified in the upper-right image to the target in the lower-left image). The altitude profile explains why the target point is not visible from the origin.

Despite the breadth of research in planning and constraint programming, the interplay between planning, information gathering, and user interaction that many important applications require presents a significant challenge. The Heracles II framework and the techniques we have presented in this article bring us closer to meeting this challenge.

We plan to formalize the Heracles II planning and information-gathering approach, combining ideas from conditional[1] and interactive[4] constraint satisfaction. Also, we plan to explore the trade-offs between performing constraint propagation and satisfaction in such systems. Although our experience is that full constraint satisfaction tends to confuse users as they interact with the system, local constraint satisfaction might be valuable. ⬒

Acknowledgments

This material is based on research supported partly by DARPA, through the US Department of the Interior, National Business Center, Acquisition Services Division, under contract NBCHD030010; partly by DARPA and the US Air Force Research Laboratory under contract/agreement numbers F30602-01-C-0197 and F30602-00-1-0504; partly by the US Air Force Office of Scientific Research under grant FA9550-04-1-0105; partly by the US Air Force under contract F49620-02-C-0103; and partly by a gift from Microsoft. The US government is authorized to reproduce and distribute reports for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and do not necessarily represent the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

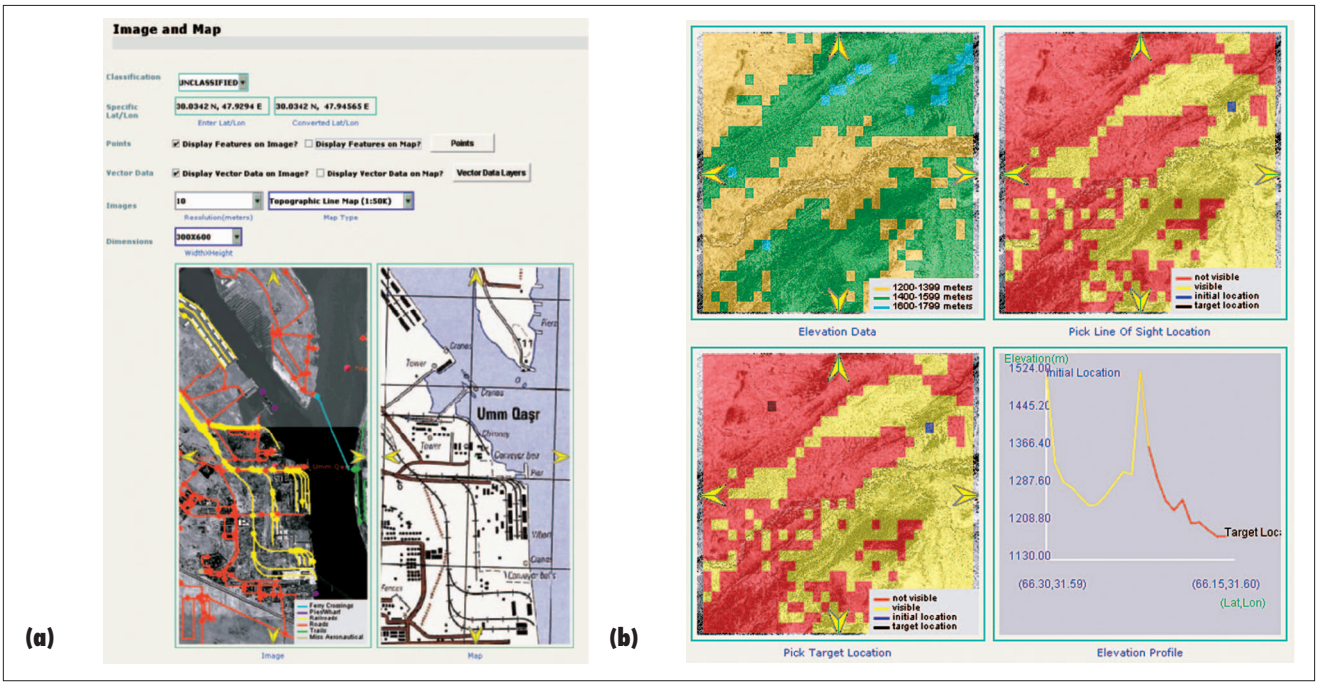

Figure 7. HeraclesMaps: (a) satellite image, vector data, and map; (b) line-of-sight computation using elevation data.

References

1. S. Mittal and B. Falkenhainer, "Dynamic Constraint Satisfaction Problems," *Proc. 8th Nat'l Conf. Artificial Intelligence* (AAAI 90), AAAI Press, 1990, pp. 25–32.

2. C.A. Knoblock et al., "Mixed-Initiative, Multi-source Information Assistants," *Proc. 10th Int'l World Wide Web Conf.* (WWW 10), ACM Press, 2001, pp. 687–707.

3. K. Erol, J. Hendler, and D. Nau, "HTN Planning: Complexity and Expressivity," *Proc. 12th Nat'l Conf. Artificial Intelligence* (AAAI 94), vol. 2, AAAI Press/MIT Press, 1994, pp. 1123–1128.

4. E. Lamma et al., "Constraint Propagation and Value Acquisition: Why We Should Do It Interactively," *Proc. 16th Int'l Joint Conf. Artificial Intelligence* (IJCAI 99), vol. 1, Morgan Kaufmann, 1999, pp. 468–477.

The Authors

José Luis Ambite is a senior research scientist at the University of Southern California's Information Sciences Institute. His research interests include information integration, automated planning, databases, and knowledge representation, and his current focus is on automatic Web service composition. He received his PhD in computer science from USC. He is a member of the AAAI and ACM SIGMOD. Contact him at USC Information Sciences Inst., 4676 Admiralty Way, Marina del Rey, CA 90292; ambite@isi.edu; www.isi.edu/~ambite.

Craig A. Knoblock is a senior project leader at the University of Southern California's Information Sciences Institute and a research associate professor in computer science. He's also the chief scientist for Fetch Technologies. His research interests include information agents, information integration, automated planning, machine learning, and constraint reasoning. He received his PhD in computer science from Carnegie Mellon University. Contact him at the USC Information Sciences Inst., 4676 Admiralty Way, Marina del Rey, CA 90292; knoblock@isi.edu; www.isi.edu/~knoblock.

Maria Muslea is a research programmer at the University of Southern California's Information Sciences Institute. Her research interests include information integration and constraint programming. She received her MS in computer science from West Virginia University. Contact her at USC Information Sciences Inst., 4676 Admiralty Way, Marina del Rey, CA 90292; mariam@isi.edu; www.isi.edu/~mariam.

Steven Minton is the chief technology officer of Fetch Technologies. His research interests include machine learning, planning, and constraint satisfaction. He received his PhD in computer science from Carnegie Mellon University. He founded the *Journal of Artificial Intelligence Research* and served as its first executive editor. He has also served as an editor of *Machine Learning*. He's a fellow of the AAAI. Contact him at Fetch Technologies, 2041 Rosecrans Ave., Ste. 245, El Segundo, CA 90245; minton@fetch.com; www.fetch.com.