Personalizing E-commerce Applications with On-line Heuristic Decision Making

Vinod Anupam Richard Hull Bharat Kumar
Bell Labs, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{anupam,hull,bharat}@lucent.com

ABSTRACT

This paper describes new technology based on on-line decision support for providing personalized customer treatments in web-based storefronts and information sites. The central improvement over existing systems is a new paradigm for specifying decisions, based on a language that incorporates flowchart constructs, rules-based constructs, and a variety of specialized constructs to facilitate reasoning based on heuristics and partial information. Reports about decisions made by a program in this language have structure that is conceptually close to the structure of that program. This makes it easy for business analysts and managers to tune the programs to enhance business performance.

To illustrate the benefits of our approach this paper describes the May-I-Help-You (MIHU) prototype system, that monitors a customer's progress through a web storefront, and may choose to proactively intervene in order to help close a sale. The intervention might offer a discount or promotion, or give the customer a "May I Help You" window, that offers an opportunity to have text chat, voice chat, and/or escorted browsing with a Customer Service Representative (CSR). In MIHU, the decision about whether to offer live assistance is fully automated, taking into account not only the business value of a given customer interaction, but also the current availability of CSRs to help realize this opportunity.

Keywords

B2C E-commerce, personalization, pro-active intervention, Vortex rules system

1. INTRODUCTION

The advent of e-commerce is forcing radical changes to the landscape of marketing and customer care. Customers are demanding increased flexibility and convenience in accessing information about products, in ordering them, and obtaining service for them. At the same time, businesses are attempting to support (a) "segment of one" marketing and service to large masses of people [18, 20], including intelligent targeted advertising, and intelligent mechanisms to identify and take advantage of profitable and loyal customers; and (b) meaningful dialogues with customers so that quality of service can be improved before customers switch to a competitor. These needs are not restricted to B2C e-commerce; web-sites in B2B e-commerce that are accessed by employees of a business must also provide effective, personalized service. This paper introduces a new approach to personalizing web-based e-commerce sites called DFP (Decision Flow Personalization), that is based on the use of on-line decision support. A central contribution is the use of a novel language for specifying decisions, that supports flowchart constructs and a specialized construct called "Decision Flow", that combines rules-based constructs and a variety of specialized constructs to facilitate reasoning based on both heuristics and partial information. The DFP approach is illustrated here by describing the MIHU (May-I-Help-You) prototype system, that proactively offers live assistance to web storefront customers.

A fundamental challenge in supporting personalization through on-line decision support is to create a high-level language for specifying decisions that supports sophisticated reasoning but which at the same time is accessible to business analysts and managers. As a starting point for our work, we interviewed business experts on customer care and personalization to understand the features that they need from a decision support language. The following requirements were determined:

(a) Ability to use both formal (e.g., chaining of rules) and heuristic (e.g., giving scores based on *ad hoc* combinations of various factors) styles of reasoning;

(b) Ability to use rules where appropriate, and to use flowchart constructs where appropriate;

(c) Ability to work with partial and/or incomplete information;

(d) Possibility for hierarchical, modular structuring;

(e) Ability to bring in outside information (e.g., access to customer profiles, the results of bulk statistical analysis);

(f) Ability to invoke side-effect functions (e.g., database updates, triggering workflows).

WWW10 May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

(g) A clear and intuitively natural semantics;

(h) A natural correspondence between reports on decisions made and the structure of how the decisions are specified (i.e., primarily the structure of the rule sets); and

(i) The language can be "owned" or controlled by business analysts and managers, without relying on programmers that translate the decision specifications into a highly technical format.

Some additional systems requirements were determined:

(j) The on-line decision engine should permit changes to decision specifications with no interruption in service; and

(k) User-friendly authoring of decision policies, including rules.

As detailed in Section 5, the rules-based decision specification languages used in existing approaches for e-commerce personalization (e.g., Manna [15], Blaze [2]) satisfy some but not all of these requirements because of their limited expressive power, and other approaches to decision specification (e.g., expert systems, logic programming) fail to satisfy some of the requirements because they are too rich.

To fill this void we use a new paradigm for specifying decisions, called Vortex. An early version of this paradigm is described in [12], where the focus was on the flexible specification of workflows that incorporated business heuristics. Central to the Vortex paradigm is the notion of "Decision Flow", which is a novel combination of rules constructs and workflow-like constructs. As detailed in Section 3, in a Decision Flow the emphasis is on computing attribute values. Some of these are targets of the decision (e.g., should a discount be offered to a customer) and others are intermediate to the decision (e.g., the likelihood that this customer will leave the site before completing the deal). Rules may be used to compute the values of individual attributes, and rules may be used to control what attributes are to be computed. (For example, attributes not relevant to a specific decision can be ignored.) Reports about decisions made can show the values of the target and intermediate attributes, and have structure close to the structure of the Decision Flow.

Decision Flows permit complex reasoning about a broad array of data about web sessions and customers. To take full advantage of this it is important to have access to rich information about the pages a customer is visiting, including the underlying intent of the pages (e.g., is it a catalog page, an instructions page, a shopping cart page) and the content delivered in them (e.g., what is the quality of a search result). Section 4 outlines and compares different approaches for gaining access to that information.

To illustrate the core technology and benefits of DFP this paper introduces the May-I-Help-You (MIHU) prototype system. This system is aimed at reducing the number of abandoned e-commerce transactions. Industry statistics [7] indicate that in the U.S. market, for every online B2C transaction that is completed there are nearly four times as many that are abandoned. Further, 7.8% of the abandoned transactions could be converted into sales by using live Customer Service Representative (CSR) interaction. This translated into $6.1 billion in lost e-commerce sales in 1999, and could lead to a cumulative loss of more than $173 billion in the subsequent 5-year period.

The MIHU system monitors a customer's progress through a web storefront, and uses stored and real-time information to infer values such as the current business value of the session and the frustration level of the customer. The MIHU system can proactively offer the customer discounts or targeted promotions. Further, the MIHU system can offer the customer a "May I Help You" window, which invites the customer to interact with a Customer Service Representative (CSR), through text chat, voice chat, and/or escorted web browsing. The decision server accesses both stored information and real-time information, including the current availability of, and load on, the CSRs.

Lucent Technologies is developing a product, called Contact Assist, that will support the functionality of the MIHU prototype system, including both an engine based on Vortex and a flexible mechanism for gathering information from a web server. Contact Assist will be available in mid 2001.

The DFP approach can be used in a wide variety of e-commerce applications involving personalization and customization, including the offering of carefully targeted promotions and discounts, helping with navigation through catalogs or self-help material, guiding a customer through an ordering process, and conducting automated dialogues with the customer. It can also be used in non-commercial web-based applications, including context-aware searching tools and automated customization of portals.

Organization. As noted above, the MIHU system will be used to illustrate the main features of our approach. For this reason, we begin in Section 2 by describing the MIHU system at a high level. Section 3 describes the Vortex and Decision Flow paradigms and illustrates their use in connection with the MIHU system. Section 4 describes approaches for incorporating on-line decision servers, such as Vortex, into web-based e-commerce sites. Section 5 considers related work. Section 6 discusses future research directions.

2. EXAMPLE APPLICATION: MAY I HELP YOU

In a department store, customers are free to browse. In a good department store, a salesperson will sometimes approach customers with the gentle question "May I help you?". In an excellent department store the timing and manner in which this question is asked is guided largely by the browsing behavior of the customer. The May-I-Help-You (MIHU) system provides a functionality for web-based storefronts that is analogous to this kind of service in excellent department stores. The MIHU system has an important advantage over a department store salesperson, which is that many businesses know the identity of customers during their visits on the web.

MIHU is a Customer Relationship Management system that interfaces to a business' web storefront. MIHU can keep track the interaction of a customer with the storefront. To be more specific, using the high-level Vortex language business analysts and managers can program the MIHU system to use customer interaction information (e.g., shopping cart content, sequence of pages visited), coupled with information available in enterprise databases (e.g. customer profile, contact history, current orders, and results from off-line decision support tools) to build a model of the customer and the

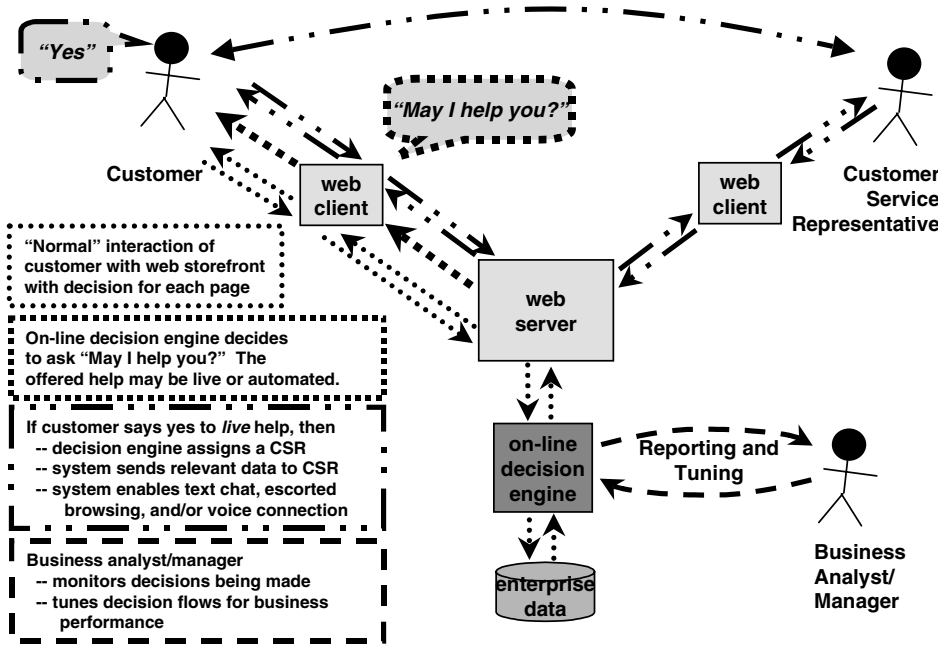

Figure 1: Overview of MIHU functionality

current interaction. Based on the individual characteristics of each customer interaction, MIHU may choose to present to the customer an icon or window offering help relevant to the current context. This help might be automated, or it might be an offer to chat with a live Customer Service Representative (CSR). In the first case, if the customer takes up the offer (e.g., by clicking on the icon or window), then appropriate context-dependent information will be delivered to the customer. In the second case, a CSR will be assigned, appropriate information will be forwarded to that CSR, and some kind of interaction with the customer will be initiated. Of course, providing live CSR help with an interactive session brings with it the opportunity to help close the sale, and also the opportunity to attempt cross-sells or up-sells.

Figure 1 summarizes the operation of the MIHU system. There are four phases or aspects to the operation. In the first phase (shown with lines having small dots) a customer has "normal" interaction with the web store-front. In particular, the web server supporting the store-front presents pages to the customer's web client, and the customer fills in blanks and submits page requests to the server. However, before the web server presents a new page to the customer, the on-line decision engine is asked whether or not the customer should be presented with the "May I Help You" option (or some other optional assistance or customer service such as a targeted discount). The decision server can access information about the customer's current web session (e.g., pages visited, shopping cart contents), and may access data from an enterprise database (including results from off-line decision support systems). The decision server may also gather information from decision engines using alternate reasoning paradigms, such as an expert system or, e.g., a specialized system for determining customer preferences.

The second phase (shown with lines having large dots) occurs if and when the decision engine determines that the customer should be given the MIHU option. In that case, the web server presents to the customer's client an applet that asks whether the customer would like assistance from a CSR.

The third phase (shown with lines having short and long dashes) arises if the customer does want assistance. In that case, some or all of three forms of interaction can be established between customer and CSR: voice conversation, text chat, and "escorted" or "collaborative" web browsing (where the CSR can select a URL and both the CSR and customer clients go to that URL, or visa-versa).

The fourth phase (shown with lines long dashes) occurs in parallel with the other ones, and at a more deliberate pace. This stage involves *tuning for business performance*, i.e., the continued examination of the decisions made for the web-storefront, with the ultimate goal of making improvements on the underlying decision policies. As will be described below, novel aspects of the Vortex language make it possible to quickly modify a Vortex program in order to achieve a desired effect.

At a superficial level, it might seem that since the MIHU system monitors a customer's progress through a web site, and peeks at the interaction between her and the web site, there are serious privacy issues involved here. However, this is not the case since the MIHU system is not getting any extra information that is not already available to the web site. However, it might be a good idea to let the customer know that such monitoring might be going on (e.g., by allowing her to opt-in when she registers with the site).

3. VORTEX AND DECISION FLOWS

This section presents a detailed introduction the Vortex language using the MIHU example, indicates how Vortex satisfies the requirements given in the Introduction, and describes the engine for executing Vortex programs. Some formal details about the Vortex language are beyond the scope of this paper, but may be found in [12].

In the current version of Vortex, programs are essentially flowcharts that may include one or more specialized nodes

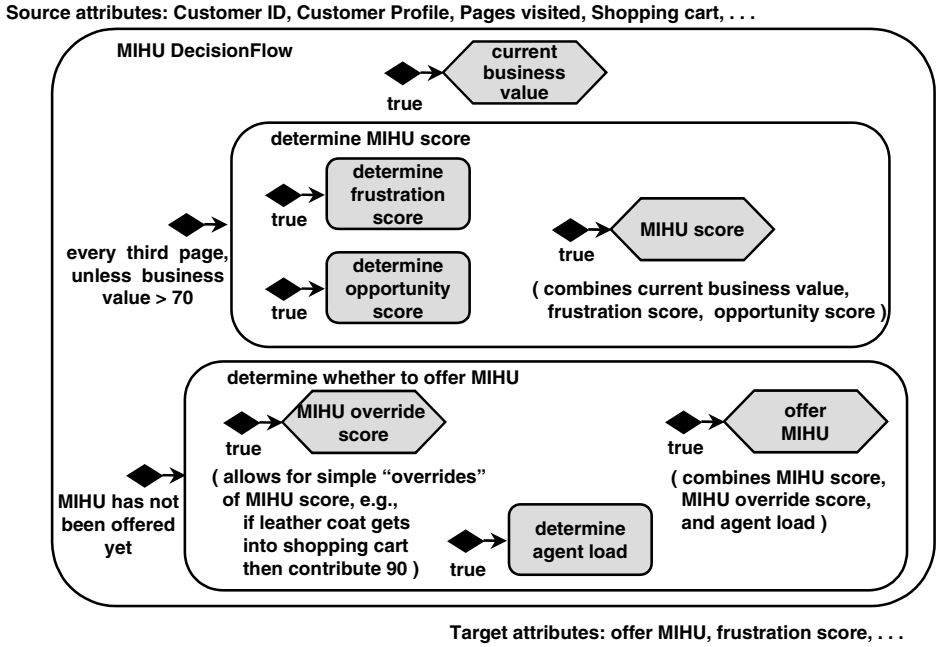

Figure 2: Representative Decision Flow for making MIHU decision (using informal syntax)

which contain "Decision Flows". Since flowchart constructs are well understood, we focus here on Decision Flows.

3.1 Vortex Decision Flows

The Decision Flow paradigm was developed for specifying complex reasoning that may involve partial information, heuristics, and multiple styles of combining information. A key criterion in the development of the paradigm was that users other than computer scientists (e.g., business managers, policy analysts, domain experts) should be able to understand the specifications of how decisions will be made, and in some cases be able to modify the specifications directly. Decision Flows support a form of incremental decision-making, that can easily incorporate a myriad of business and other factors, and specify the relative weights they should be given. Decision Flows support a rule-based style of specifying decision policies, and are more expressive than decision trees and traditional business rule systems. However, Decision Flows are less expressive than conventional expert systems as a result of a novel approach to structuring the rule set underlying a Decision Flow. This helps to simplify explanations of how a decision is made, and reduces the "ripple effect" that often arises from modifications to programs written in with expert systems or logic programming languages.

We briefly illustrate some of the basic Decision Flow constructs using Figures 2 and 4, which give a high-level picture of a representative Decision Flow that can be used for making the MIHU decision. In the MIHU prototype, a Decision Flow like this would be one node of a flowchart which is executed for each page that is served to a customer. For example, this outer flowchart might operate as follows: (a) test whether the page indicates a new session or is the continuation of an active session, (b) if a continuation then retrieve information (from a main memory database or internal data structure) about previous pages of the session (c) possibly get customer profile information, (d) execute a Decision Flow that decides whether to make an automated intervention, and (e) inform the web server about the decision made.

The two Decision Flow figures show rules and conditions using an informal, pidgin syntax. In the text-based version of Vortex, the syntax for conditions and terms is close to the C language. A GUI is provided for Vortex programmers, including wizards to help with rule construction, query construction, and the like.

Decision Flows are *attribute-centric*. In particular, a Decision Flow specification has *source attributes* or input parameters; in the example these hold information about the customer identification, customer profile and current session. The specification also includes a family of *derived attributes*, which may be evaluated during execution. Some of the derived attributes will be *target*, and embody the output of a Decision Flow; in the example this includes a boolean indicating whether to offer the MIHU functionality, and additional attributes giving characteristics of a session. The current prototype Vortex system supports data types associated with relational databases, namely scalars, tuples of scalars, lists of scalars, and lists of tuples of scalars. (We expect to incorporate XML data in the next round.)

The Decision Flow of Figure 2 shows individual attributes using hexagons (e.g., `current business score`, `offer MIHU`), and modules using rounded-corner boxes (e.g., `determine MIHU score`, `determine frustration score`). A hexagon node may contain rules that specify how an attribute is to be computed; this will be described below in Subsection 3.2. External functions such as database queries, calls to a heavy-weight decision support system (e.g., an expert system), or side-effect functions (e.g., database updates, triggering workflows) can also be included. The modules may be hierarchically organized, and may contain other modules, hexagons, and external functions.

We now use Figure 3 to explain intuitively how the MIHU Decision Flow operates. This figure shows a report present-

Session ID	Page	Business Value	Frustration Score	Opportunity Score	MIHU score	Override Score	CSR Load	Offer MIHU
250	1	70	6	0	46	0	80	false
250	2	70	14	0	54	0	81	false
250	3	70	20	0	60	0	81	false
250	4	70	24	0	64	0	79	false
250	5	85	8	51	83	0	78	true
282	1	40	6	0	25	0	83	false
282	2	40	–	–	–	0	82	false
282	3	57	–	–	–	90	82	true
282	4	57	16	29	43	–	–	–
282	5	57	–	–	–	–	–	–
282	6	71	19	37	58	–	–	–
282	7	71	25	48	58	–	–	–

Figure 3: Data from report on MIHU decisions

ing some representative decisions reached by the decision engine. The columns of this report correspond to some of the most important attributes of the Decision Flow, and each row corresponds to a single execution of the Decision Flow. During a single execution, the value of `offer MIHU` is based on three intermediate attributes: `MIHU score`, `MIHU override score`, and `CSR load`. In the example, if either of the scores is $\geq$ `CSR load`, then the MIHU functionality is offered.

The `MIHU score` attribute is based on other intermediate attributes, which focus on the current business value of the customer and session, on the estimated frustration level of the customer, and on the estimated opportunity for making money from the customer (either by encouraging the customer to purchase the contents of the shopping cart, or through a cross-sell or up-sell). The frustration score and opportunity scores in turn depend on additional intermediate attributes.

Referring to Figure 3, the first five rows of Figure 3 show how the different scoring attributes might vary over a user session. We assume in this example that the customer visited 5 pages, and placed something in the shopping cart when sending the 4th page back to the web storefront. In the Decision Flow used to generate this example, `frustration score` goes up, except when the customer places something in the shopping cart. The intuition here is that customer frustration goes down if there is a feeling of progress, e.g., after several searches a product is found and put in the shopping cart. On the other hand, the `opportunity score` generally goes up when something goes into the shopping cart, both because there is something in the shopping cart, and in some cases there are possibilities for cross-sells and up-sells.

Of course, the specific behavior of the attributes in a Decision Flow is determined by the business analysts and managers who program it. As a result, the Decision Flow for MIHU described here can be adapted to encompass any principles and heuristics that a business manager wants.

A key feature of Decision Flows is the use of *enabling conditions* or guards on the execution of attributes, modules, and external functions. For example, `determine MIHU score` has as enabling condition, expressed informally, that the module will be executed if the `current business value` is > 70, and otherwise on every third page of the web session. Likewise, `determine whether to offer MIHU` will be executed only if the MIHU option has not yet been offered to the customer.

Session 282 in Figure 3 illustrates how the enabling condition on `determine MIHU score` impacts Decision Flow executions. In this session the `current business value` is ≤ 70, and so the `determine MIHU score` is not executed on the 2nd, 3rd or 5th pages. On the 6th page the `current business value` goes above 70. So `determine MIHU score` is computed for the 6th and 7th page.

Enabling conditions are useful in at least three contexts: (a) to permit savings on resource usage (as just illustrated); (b) to avoid the computation of irrelevant attributes (e.g., once MIHU has been offered there is no need to compute `offer MIHU`); and (c) to indicate which attributes should be ignored if a realtime constraint is about to be violated.

What happens if an attribute that has been disabled is referred to by the computation of some other attribute? One design principle of Decision Flows is that any attribute may have null value, and that any attribute computation must be able to work with null inputs. This is motivated in part by the observation that data retrieval over a network is not reliable, and that many decisions must be made using partial information. Suppose in the example that the MIHU option has not been offered yet. Even if the module `determine MIHU score` is disabled, i.e., not evaluated, the rest of the Decision Flow will be executed, and a final value for `offer MIHU` will be obtained. The condition language used in Decision Flows can test whether an attribute has been disabled (i.e., the enabling condition is false). Importantly, both the condition language and the attribute computation language used in Decision Flows were designed to work in the context of partial information and null values (see [12]).

As detailed in [12], a declarative semantics is associated with Decision Flows. Under this semantics Decision Flows are viewed as input-output devices, which map a given set of source attribute values (and an underlying environment, such as any databases accessed) to a given set of target attribute values. It turns out that given a set of source attribute values (and a fixed underlying environment), a well-formed Decision Flow uniquely determines the values of the target attributes. A key factor in achieving this declarative semantics is that Decision Flows must satisfy a certain

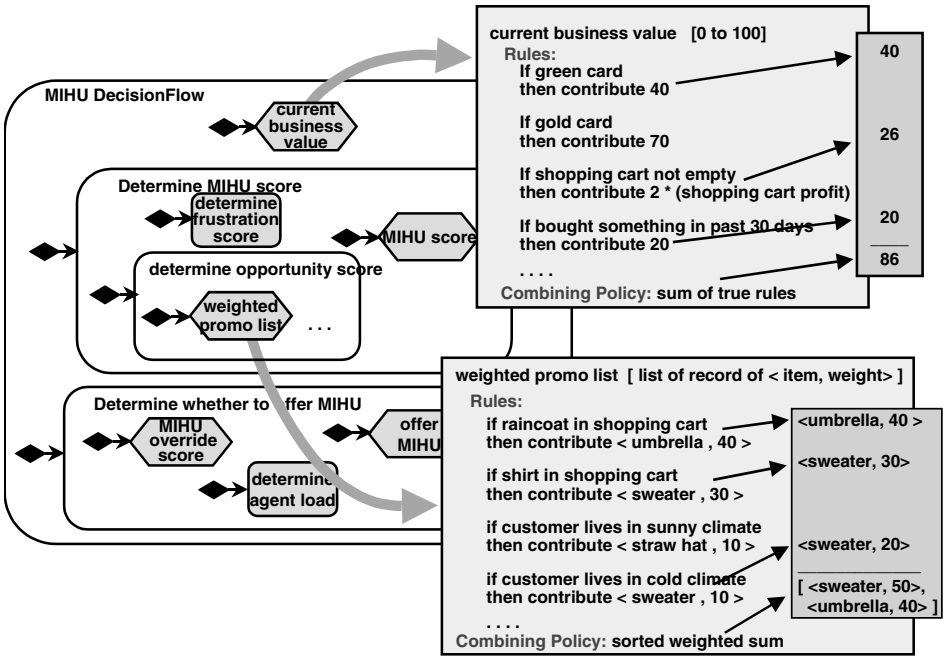

Figure 4: Attribute Rules and Combining Policies in a Decision Flow

acyclicity property. In particular, a graph can be formed for each module M, where the nodes are the top layer modules, attributes, and external functions of M, and which contains an edge from node A to node B if (i) [*data flow*] an attribute defined in A is used in the computation of B, or (ii) [*enabling flow*] an attribute defined in A is used in the enabling condition of B. For a Decision Flow to be well-formed, this graph must be acyclic for each module. In operational terms, the acyclicity condition implies that there will not be race conditions between different attributes being evaluated. Further, the acyclicity condition underlies our claim that Decision Flows are easier to understand than expert systems, and suffer less from the ripple effect.

There are three advantages to the declarative semantics just outlined. First, the semantics provides a clear and unambiguous meaning for Vortex Decision Flows. Second, people developing Vortex Decision Flows can largely ignore flow of control issues, and focus instead on the business logic they are trying to express. This provides a key difference between Decision Flows and conventional flowcharts. (The Vortex compiler will alert the user if the acyclicity condition is violated.) And third, the declarative semantics affords some possibilities for optimizations, of both response time and system throughput (see [11]).

3.2 Attribute Rules and Combining Policies

Another key feature of the Decision Flow paradigm is the tremendous flexibility given to users when specifying how an attribute should be computed. In addition to permitting external function calls (e.g., database dips, or calls to execute in a different decision support engine) the paradigm supports the use of *attribute rules* and *combining policies*. Two simple illustrations are provided in Figure 4, which shows the contents of the `current business value` and `weighted promo list` attributes. As shown there, a family of rules is associated with attribute `current business value`, each potentially contributing a number. Numbers contributed by rules with true condition are to be combined by summation. In the example, rules contribute 40, 26 and 20, resulting in final value 86.

The attribute `weighted promo list` illustrates a more interesting combining policy. The output of this module will hold a list of promo items, ordered according to how well they fit the current situation. The individual rules contribute ordered pairs, consisting of a promo item along with a numeric weight (e.g., < umbrella, 40 >). As illustrated by the second and fourth rules here, several rules might contribute to the same promo item. The combining policy for this module is to group contributed pairs by promo item, then add the weights for each promo item, and finally sort the list of resulting pairs according to the aggregated weights.

More generally, the Decision Flow paradigm offers a broad range of combining policies for aggregating the contributions of a rule set. Other combining policies involving numbers include maximum, minimum and average. As illustrated with `weighted promo list` the contributed values and the result may have structured type. In addition to supporting a family of *ad hoc* combining policies, the system supports an OQL-like [4] algebra for specifying customized combining policies.

The presence of multiple combining policies permits the use of different styles of reasoning within the Decision Flow paradigm. Decision Flows also support different styles of reasoning at a more granular level as well. We illustrate this in connection with the attributes `MIHU score` and `MIHU override score`. We have discussed how `MIHU score` involves a deliberate derivation involving many factors. In contrast, the attribute `MIHU override score` is computed by an atomic node that includes collection of simple and disjoint factors (e.g., that a particular item is in the shopping cart, or that a certain page has been visited) and uses as combining policy "maximum contributed value". If `MIHU override score` is greater than `CSR load` then the MIHU

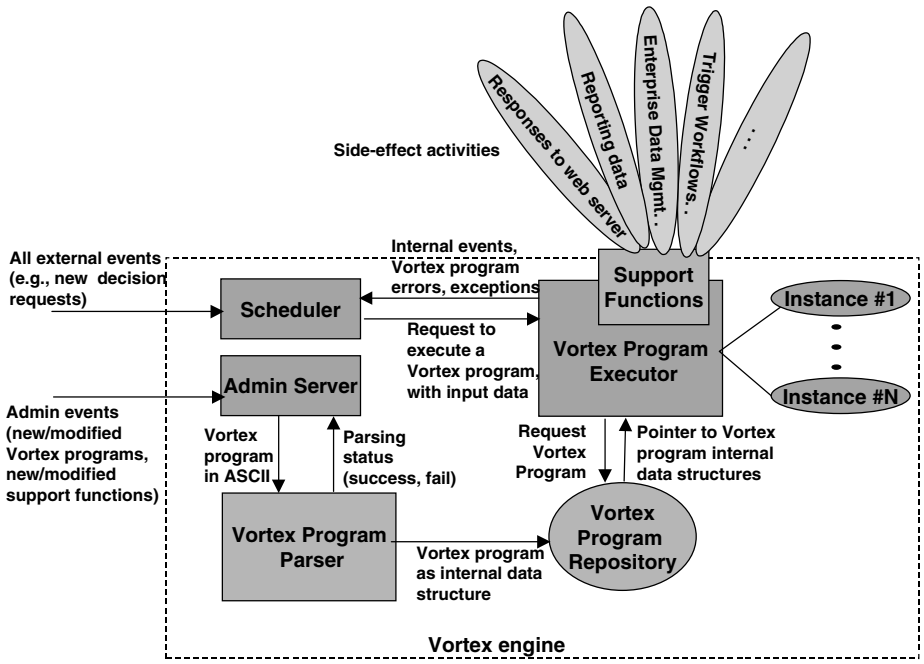

Figure 5: High-level architecture of Vortex engine

option will be offered, and so each rule in `MIHU override score` is analogous to a presidential veto or gubinatorial pardon. For example, in the second session of Figure 3 the `MIHU override score` goes to 90 on the 3rd page, perhaps because a leather coat was placed into the shopping cart; and this triggers the offer of MIHU.

3.3 Miscellaneous

This subsection gives added details about the Vortex system, and reviews it in light of the requirements given in the Introduction.

We have already seen from the preceding discussion that the Vortex language satisfies requirements (a) through (g).

Turning to requirement (h), the "attribute-centric" nature of Decision Flows makes possible reports about how or why decisions were made are conceptually close to the Decision Flow specification. In particular, reports such as in Figure 3 can be created using some or all of the attributes derived by the Decision Flow. Given a family of decisions, a user can inspect this report (either manually or using automated techniques such as regression analysis or data mining) to see whether the various attributes and criteria are given appropriate emphasis. If anomalies are found, then it is relatively easy to find the corresponding places in the Decision Flow that should be modified. Furthermore, we expect that this close correspondence between reports and Decision Flow structure will facilitate the development of self-learning tools that will work on top of Vortex.

Decision Flows reveal the key factors involved in making a decision or evaluation, and hide a substantial amount of detail about the execution. In contrast, when specifying an equivalent decision using a conventional flowchart or Petri Net formalism, the key factors and logic are obscured by the plumbing. In Decision Flows different ways of executing rules can co-exist; this contrasts with logic programming languages and conventional expert systems, which have a single execution semantics, and force the use of awkward simulations if rules are to be combined in a different way. It is these considerations along with the correspondence between reports and the structure of Decision Flows that lead us to believe that Vortex satisfies requirement (h).

We turn now to requirement (j). Figure 5 shows a high-level architecture of the Vortex engine. Vortex programs are input into the Administrative Server, which invokes a parser. This checks that the program is well-formed and compiles it into an internal data structure. When the program is to be executed, i.e., when a decision is to be made based on given input parameters, a copy of the data structure is created, and that copy is then interpreted. As a result, the Vortex program can be modified, parsed, and complied into a (new) internal data structure. The new data structure can then be used for subsequent decisions. In this way, Vortex programs can be modified without bringing the engine down. For efficiency, the Vortex engine has been implemented in C++. Furthermore, many of the specific operations of a Vortex program (e.g., arithmetic comparisons, list manipulations, external functions) are performed by "support functions", which are compiled. Additional support functions can be added to the engine without bringing it down.

With regards to requirement (k), a prototype GUI has been implemented to support specification of Vortex programs. A visual palette is provided for the Decision Flow constructs; this has appearance similar to the images of Figures 2 and 4. Wizards are provided for building up flowchart nodes, rules, attribute modules, database queries, etc.

The example MIHU Decision Flow described earlier is relatively simple, in terms of the size of the Decision Flow and the nature of the data being evaluated. We have developed richer Decision Flows that involve many modules and over 50 attributes.

4. INTEGRATION WITH WEB SERVERS

The second key component of the DFP approach to web personalization concerns creating a linkage between the on-line decision server and the web server hosting a site. We begin by discussing the kinds of information that need to be passed back and forth between web server and decision engine. We then consider different ways to convey relevant information to the decision engine, and to permit decisions made to have impact on the web server.

4.1 Raw vs. Semantic Information

The DFP approach to web personalization is based on providing relevant information to a sophisticated on-line decision engine. This subsection distinguishes between the raw data that can be obtained easily and higher-level semantic information, such as used by the Decision Flow in the example of Section 3.

At a minimum, the on-line decision engine should have access to the following kinds of information. We are not suggesting that all of this data will be used in each decision, but that it should be available if deemed relevant.

(a) *History of customer clicks*: This includes not only the web requests that the customer is making, but also the the navigation path being followed around the site, the times spent at each page, and the entries made into any forms.

(b) *Web server responses*: The response of a web storefront to a customer may be very important in understanding the customer experience. For example, to determine frustration stemming from difficult searches, it is important to know about both the number of searches performed and also the sizes of returned answers.

(c) *Enterprise data*: A broad variety of stored information may be useful to the personalization. At a minimum this will include accessing information resulting from bulk statistical analyses and information on inventory and availability times. If the customer has been identified then customer profiles and recent customer histories can also be incorporated into the personalization process.

There is also a higher, semantically rich kind of information that can be helpful as input for the decision engine. This will include information, for example, about the category of page being accessed by the customer (e.g., search request, search answer, catalog entry, shopping cart), or the intent of a page (e.g., that it includes a promotion or indicates that a certain catalog item is out of stock).

Indeed, an important part of installing a DFP system, or any web personalization system using on-line decision support, will involve creating or determining a model of the web site, that incorporates relevant models of customers, the intent of their activities on the site, the business value of those activities, indicators that a customer may abandon a transaction, etc. The example of Section 3 provides a starting point for such a model, but a variety of other factors may be brought into play. This model will be clearly visible in the program driving the decision engine, and will help to guide the kinds of information that need to be passed from web server to decision engine.

There is a trade-off between attempting to automatically infer semantically rich information from the HTML passed between customer and web server vs. manually incorporating that information into the web page generation so that it can be obtained easily by the decision engine. Attempting to infer this information automatically typically involves parsing the HTML; it will involve the development of special-purpose code and be computationally expensive. Further, its success will depend on how direct and uniform the relationship is between the actual HTML content of the web pages and their intent. On the other hand, incorporating code into the web page generation that captures the semantically rich information puts an additional burden on the web site developer, both at creation time and during maintenance. A site such as Amazon or Yahoo could have thousands of pages, some static, others generated dynamically via server-side scripting languages such as ASP/JSP, or CGI-scripts/servlets. It will be a huge effort to modify all executable scripts to add the MIHU functionality.

Sophisticated web authoring environments such as Microsoft's FrontPage or Allaire's ColdFusion Studio provide hooks so that web site authors can easily incorporate semantically rich information into the HTML generated by their code. Thus it would be straightforward for site developers to extract high-level semantic information to be passed to the Vortex engine. However, if the site has not been built using such tools, we expect that early adopters of our personalization technology will opt for parsing the HTML, and will use only some of the information actually available.

4.2 Acquiring the Information

In this subsection, we examine five techniques of gathering session information required for the decision engine, and discuss their pros and cons. A summary is presented in Figure 6. It is expected that a combination of these techniques will be required in a DFP toolkit, if it is to be deployed to support personalization for a broad variety of web sites. After presenting the techniques we make some general remarks.

Content generation scripts send high-level semantics to decision engine. Assuming that all pages that need to be tracked are generated via executable scripts/programs (which is a reasonable assumption to make for large sites), an obvious approach to obtaining meaningful semantic information would be to create or modify these scripts to gather/create the desired information, and then pass it on to the decision engine. The primary advantage of this approach is that the people developing the web pages will have the best idea of the intended semantics of the pages, and thus what the decision engine should receive. For this reason, we expect this to be the approach of choice when creating new web sites. Further advantages are that the actual HTTP requests and responses do not need to be transformed/parsed, and HTTPS connections can be handled. The primary disadvantage concerns legacy sites, where modifying all the existing scripts to generate the high level semantic data would be quite expensive. Another disadvantage is that maintenance of the site would become more cumbersome.

So how can the DFP approach be used in large legacy web sites? In such cases, the only solution might be to try and extract meaningful information from the raw HTTP requests/responses. There are various ways to do so, some of which we discuss below.

Strategy	Converting legacy sites easy	HTTPS handled?	HTTP response transformation required	HTTP response parsing required
Content generation scripts send high-level semantics to engine	No	Yes	No	No
Content generation scripts send raw HTML to engine	Yes	Yes	No	Yes
Wrapper scripts	Yes	Yes	Maybe	Yes
Proxies	Yes	No	No	Yes
Web Server Extensions	Yes	Yes	No	No

Figure 6: Comparison of Web Interaction Monitoring Strategies

Content generation scripts send raw HTML to decision engine. This is a variation of the approach mentioned earlier, however, in this case, only the raw HTTP requests/responses are forwarded by the scripts. This can be done by injecting the same (small) block of code into the scripts that generate each and every page of the web site. The advantage is that converting a legacy site to this approach is straight-forward (assuming that it was implemented with server-side scripting language such as JSP or ASP), since the only function the extra piece of code performs is to forward appropriate data (HTTP request and/or response) to the decision engine. However, since the injected code will be uniform and generic, it will not be able to extract high-level semantic information from each page. This means that detailed knowledge of what information to extract for specific (categories of) pages, and how to extract it, needs to be built, either in the decision engine or in some other process. Depending on the level of information to be extracted, this can cause maintenance problems anytime the structure of the corresponding pages change. Moreover, this approach is sensitive to the language and/or platform that the web site is implemented in, e.g., if the CGI scripts are based on C++ then it may be hard to know where to inject the code block, making the approach infeasible.

Wrapper scripts. The idea here is that the web server can be configured so that all web requests and responses (that need to be tracked) are filtered through executable scripts that perform the task of extracting the relevant information and contacting the Vortex server to determine the appropriate response. Note that these wrapper scripts could reside on the web server(s) that are supporting page requests, or could reside on separate machines. As opposed to the previous approaches, the advantage in this scenario is that the actual content generation is not affected — this method is simply layered on top. Moreover, HTTPS connections can be handled, since the wrapper script gets the customer request after it is decrypted by the server, and parses the response before it is encrypted and sent to the client. A disadvantage is that HTML pages being served would need to be transformed, since the links/forms/frames in existing pages now need to go through the wrapper, whereas other objects (e.g., pre-loaded images via Javascript) need to be accessed directly. However, it can be hard to automatically transform all underlying pages, especially if a lot of destination URL computation is done inside client-side script code, which would require the wrapper to parse the corresponding scripting language. Also, HTML pages input to the wrapper would need to be parsed and translated into higher level semantic information, either by the wrapper or the decision engine. Finally, session tracking information may be lost under this approach, if the web server is using a cookie-based scheme that tracks sessions for some but not all of the web site pages. In that case, replacing URLs so that they access the wrapper scripts may disrupt the web site's scheme for putting cookies at the customer site. To remedy this, some re-writing of the web site scripts would be required. However, this problem can be eliminated if the web server allows URL re-direction based on customizable rules. In that case, the customer could see the same URL and no HTML re-writing would be required, hence session tracking would not be a problem.

Proxies. A proxy can be inserted between a company's web site and the end user. The proxy is responsible for tracking user requests, extracting the site responses, and contacting the decision server to determine the appropriate intervention strategy. An advantage is that HTML page transformation is not required. However, there are several disadvantages to this approach. Firstly, if SSL tunneling is being used, then the proxy will need to serve as the receiving end of the tunnel, and will need to perform encrypting/decrypting of the web traffic. Moreover, it would also need to extract higher level semantic information from the HTML. Lastly, the use of one or more proxies may have impact on scalability, because the proxy servers can become a bottleneck. It will be important to have enough proxies to cover the anticipated load on the web site.

Web Server Extensions. Most popular web servers (Apache, Netscape Enterprise, Microsoft IIS) have an API (Apache modules, Netscape's NSAPI, Microsoft's ISAPI) that can be used to extend the functionality provided by the server. In particular, these can be used to attach monitoring hooks into the web server itself, thus gaining low-level access to all web interactions. The advantage is that no transformation of HTML response being generated is required, and secure connections can be handled. The disadvantages of needing to extract higher-level semantic information from HTML responses still applies. Moreover, writing server extensions is tricky (since they should not impact reliability or scalability) and server specific.

We conclude this subsection with some general remarks about these techniques and our experience with two of them.

We first consider session tracking. Three techniques are commonly used for tracking a session in web sites: encoding the session ID into the URLs sent and requested by the customer, placing cookies on the customer machine, and placing the session ID into a hidden form field. (The latter technique requires that all pages transmitted to the user are generated via form submissions.) In order for the decision

engine to know the session of a page request, the session ID must be passed to the engine along with other page information. The session ID can be sent explicitly, or it can be sent as it occurs in the HTML of the requested page, and the encoding scheme used by the web site can be used to extract the session ID.

We now turn to the issue of scalability. In particular, how do the above techniques work when a web site is supported by a web server farm rather than a single web server? There are two main issues. First, in the case of a web server farm there may also need to be a farm of decision engines. Because the log of a given customer session will generally be maintained in the main memory of a single decision engine, it will be important that all decisions about that session be made by the same decision engine, even if different web servers are being used to serve the pages. This can be accomplished by encoding the decision engine ID inside the session ID. A load-balancing strategy can be implemented to distribute customer sessions across the decision engines. Furthermore, in applications such as MIHU, if all of the decision engines reach saturation then the system can decide for some customers that they will not receive any MIHU decisions. This permits a graceful degradation of service in the face of unexpectedly high load.

The second scalability issue concerns how the added expense of transmitting information from web server to decision engine will impact performance. In all cases except for proxies, the processing involved in transmitting to the decision engine can be performed on the web server. Thus, each server will be more loaded, but no architectural problems arise. In the case of proxies (and wrapper scripts if they are implemented on separate machines) there is a possibility of the proxy becoming a bottleneck.

We have built two versions of the MIHU prototype at Bell Labs, that explored some of the issues discussed above. In the first case, we modified the content generation CGI-scripts used by the web site. In the second case, we wrote a wrapper servlet. Here, actual request URLs were passed to the wrapper servlet via its `PATH_INFO` environment variable. The servlet then performed the original request and parsed the HTML response generated to extract the relevant information. Before shipping the response to the customer, the servlet also modified the links/forms/frames in the page to go through the servlet, and inserted a `BASE` tag that pointed to the original URL so that any relative accesses (e.g., preloaded images inside Javascript) would work. None of the actual pages stored at the web site needed to be modified.

Importantly, with any of the above monitoring methods presented, the method can be phased into the site – e.g., initially the tracking can focus only on part of the site, and only on part of the relevant data.

4.3 Impacting the Web Site

So far, we have focused on tracking the user experience and using the on-line decision engine to recommend how the customer experience should be impacted. We now consider the kinds of recommendations that can be made, and how they can be acted upon.

A natural kind of recommendation is the placement of an icon or image on a page transmitted to the customer. The image could either correspond to a promotion, or could even offer the customer live agent help (for web sites with call centers). If the customer clicks on the image, the web server could assign a live agent to interact with the customer. These techniques are common in existing personalization approaches, and easily supported in our approach.

It is also possible for the decision engine to generate entire pages or frames. For example, at Bell labs we have used the Vortex engine to choose from a number of parameterized page templates, and then choose values for the parameters. In this way, simple automated conversations with the customer can be performed. Extending this technique to support more complete, highly personalized generation of web sites is an important direction, and will require the creation of a rich development environment, and perhaps another layer of abstraction on top of the current Vortex language.

5. RELATED WORK

This section compares the DFP approach with related work, including other e-commerce personalization solutions, other decision specification paradigms, and finally a system that provides some aspects of the MIHU functionality.

Existing e-commerce personalization tools based on on-line decision support use rules languages that are quite limited in expressive power. For example, Manna [15] provides an event-condition-action rules language, where the actions result in side effects outside of the rules system. There is no chaining of rules, which limits the expressive power. For example, it is not feasible in these systems to use a cluster of rules to compute a business opportunity score, another cluster of rules to compute a customer frustration score, and a final cluster of rules that combines the two scores and other information to select an appropriate action. The scripting language of Blaze [2] provides flowchart constructs, where a node in a flowchart may contain a set of rules. Inside such flowchart nodes there is no chaining of the rules. And so any rule chaining that needs to be expressed is explicit in the use of flowchart constructs between the rules nodes. As a result, users must explicitly specify essentially all flow of control for the decision-making process. For complex decisions this is too cumbersome for business analysts and managers.

What about using some other, existing decision specification system to support e-commerce personalization? Decision trees are too confining in their logic. Logic programming [14], including variants that incorporate negation (e.g., [21, 10]), while Turing complete, is both too expressive (because it is hard to develop reports that easily explain how decisions are reached) and too confining (because it forces a single semantics for combining rules which makes it hard to directly express both formal and heuristic styles of reasoning). Expert systems (e.g., OPS5 [3]) are also too expressive, because it is difficult to explain how decisions are reached, and difficult to predict the overall effect of modifying a rule. One candidate that met several of the user requirements is the RAISE system [9]; however, the adherence to a logic-programming style of rules did not support directly expressing both formal and heuristic styles of reasoning.

Personalization based on off-line decision support (e.g., Epiphany [8], Net Perceptions [17]) performs periodic bulk data analysis using data mining and statistical techniques to infer correspondences between, e.g., customer types, products already selected, and products that would be appropriate for targeted promotion. As with other on-line approaches, DFP is complimentary to the off-line approach, and decisions made using DFP can access the results of off-line analysis.

Another tool that can be used for on-line decision support is described in [16], which presents a formalism for scoring the quality of data assembled from multiple sources. The formalism uses vectors giving scores to various criteria, along with operators to combine vectors; these combinations correspond to combinations of scientific data performed in the underlying workflow. The Decision Flow paradigm can express this formalism, because it supports direct manipulation of record types, and can include the operators for combining vectors as named combining policies.

We contrast our approach with another popular approach for customizing web-sites, that is based on guiding the customer through a series of questions that are used to identify customer preferences and their relative weights (e.g., Personalogic [19]). More generally, [1] presents a framework for specifying and combining preferences, that provides high flexibility and satisfies mathematical properties such as closure under certain operations. The Decision Flow paradigm can simulate this preference model by using record structures and specialized combining policies.

The DFP approach consists of using an on-line decision engine that is separate from the web server. An alternative would be to use some form of server-side scripting language (e.g., ASP, JSP, or PHP), that could hold the business logic for on-line decision making. The fundamental problem here is that these scripting languages do not satisfy several of the requirements on the decision specification language, such as the explicit presence of rules, the ability to combine information in different ways, or the correspondence between programs and reports.

Finally, the MIHU system presented here differs from previous systems for presenting live CSR assistance to customers. For example, iContact [13] uses a simple rules mechanism to identify customer sessions on a web store-front that are "good" candidates for live CSR assistance. However, with iContact the CSRs are given a listing of these candidates, and the CSRs make the final decision about whether or not to offer live intervention. With the MIHU system the entire decision can be automated (taking into account not only the business opportunity afforded by the customer, but also the current availability of CSRs to help realize this opportunity). This permits treatment of customers which is more uniform than with the iContact approach. The MIHU system also permits more careful selection of a CSR whose skill matches the expected needs of a given web-based customer. Finally, it can support more sophistication in the use of "blended" CSRs, who spend some time with web-hosted customers and other time with traditional telephone-based customers.

6. DISCUSSION AND FUTURE WORK

This paper presents the DFP framework for personalizing web sites that is based on the use of an expressive on-line decision engine. Key requirements on the language used by the decision engine were identified, and the Vortex language with Decision Flows was introduced to satisfy these requirements. Various approaches to implementing our approach on new or existing web sites were explored, and our experiences with implementing some of them discussed. Finally, the paper described the May-I-Help-You prototype system, which applies the technology to decrease the number of e-commerce transactions that are abandoned. We close by mentioning several directions for future work.

VorteXML. The current Vortex language and engine is geared primarily towards relational data. We are working to extend Vortex so that it can also work with XML-based data, thus allowing a uniform paradigm for converting the raw data into higher-level semantic information and for analyzing that higher-level information (see [5]). There are various ways in which a web site can pass data to the decision engine. For example, the web server might do some initial processing and cleaning to transform the HTML to XML (e.g., using XPath [6]), and then pass the XML to the VorteXML engine. The ability for Vortex to specify various heuristics would be useful in converting that XML into higher level semantics, especially for web sites that represent information in non-uniform ways.

Another approach is to *annotate* the HTML content produced by a web site, by adding custom tags that are ignored by the client browsers, but which the decision engine can scan to extract the desired information. This would make the task of extraction easier, since the decision engine now only needs to look at a subset of the raw input.

There is also a move towards separating content from presentation on web sites, i.e., for each customer request, a web site would retrieve the actual content as an XML document, and then apply an XSL stylesheet to transform it into an HTML document before shipping it to the client. In such a case, the web site could simply forward the XML content to the decision engine, which would remove the task of transforming the HTML into XML.

Distributed Rules Processing. Section 4 described how DFP can be scaled, in an architectural sense, to environments with web server farms. Another challenge concerns scaling to large web sites, as found in the B2B sites of large corporations. These typically span multiple sub-organizations and span multiple geographic locations. For example, in Lucent a customer might enter the Lucent home page that is supported by web servers in New Jersey, but then access product information about the latest IP telephony switches via web servers in Illinois. Furthermore, while some decision policies might be applicable to all customers, others might be relevant only to certain products. This means that the rule sets used in connection with different locations may be overlapping but different. The challenge is to support the development of such overlapping rule sets, have appropriate rules apply to the pages being examined, and pass relevant data between geographic locations as the customer's session moves between those locations.

Reliability. We are currently experimenting with the use of fault-tolerant CORBA to provide reliability for the Vortex engine, and thus for the DFP approach. Another way to achieve reliability, and scalability for that matter, would be to implement the Vortex language as part of an application server platform (e.g., based on EJB).

Automated learning. Following the lead of companies such as Manna [15], it will be important to incorporate automated learning into any personalization technology. Because Vortex provides rules-constructs that are richer than many business rules systems, it will be more difficult to develop learning technology for Vortex. On the other hand, the structure of the rule sets and the availability of meaningful reports should provide important handles to the problem.

Acknowledgments

The authors thank Greg Anderson of Avaya Communication for suggesting "May I Help You" as a useful test case and illustration of the Vortex Decision Flow technology.

7. REFERENCES

[1] R. Agrawal and E. Wimmers. A framework for expressing and combining preferences. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 297–306, 2000.

[2] Blaze Software Home Page, 2000. `http://www.blazesoft.com`.

[3] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, Reading Massachusetts, 1985.

[4] R. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[5] V. Christophides, R. Hull, A. Kumar, and J. Siméon. Workflow mediation using VorteXML. *IEEE Data Engineering Bulletin*, 24(1), March 2001.

[6] J. Clark and S. DeRose. XML Path Language (XPath). Technical report, World Wide Web Consortium, 1999. W3C Recommendation 16 November 1999.

[7] Datamonitor, Inc. The U.S. market for internet-based customer service, 2000.

[8] Epiphany Home Page, 2000. `http://www.epiphany.com`.

[9] B. N. Grosof, D. W. Levine, H. Y. Chan, C. J. Parris, and J. S. Auerbach. Reusable architecture for embedding rule-based intelligence in information agent. In *Proceedings of the Workshop on Intelligent Information Agents*, December 1995. Held in conjunction with the ACM Conference on Information and Knowledge Management (CIKM-95).

[10] G. N. Grosof. Compiling prioritized default rules into ordinary logic programs. Technical Report RC 21472, IBM Yorktown, May 1999.

[11] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *Proc. IEEE Intl. Conf. on Data Engineering*, pages 281–292, 2000.

[12] R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. of Intl. Joint Conf. on Work Activities Coordination and Collaboration (WACC)*, pages 69–78, February 1999.

[13] iContact Home Page, 2000. `http://www.icontact.com`.

[14] J. W. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer-Verlag, Berlin, 1987.

[15] Manna Home Page, 2000. `http://www.mannainc.com`.

[16] F. Naumann, U. Leser, and J. Freytag. Quality-driven integration of heterogeneous information systems. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 447–458, 1999.

[17] Net Perceptions Home Page, 2000. `http://www.netperceptions.com`.

[18] D. Peppers and M. Rogers. *The One to One Future*. Doubleday, New York, 1993.

[19] Personalogic Home Page, 2000. `http://www.personalogic.com`.

[20] P. B. Seybold and R. T. Marshak. *customers.com*. Random House, New York, 1998.

[21] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.