SupportBeam: An Infrastructure for Web-based Customer Support

Ajay Mohindra,Tom Chefalas, Alexei Karve, Quentin Gouedard, Srikant Jalan and Steve Mastrianni

IBM Thomas J. Watson Research Center
P.O.Box 218
Yorktown Heights, NY 10598
October 9, 2001

Abstract

Telephone-based customer support has been a major source of expenditure for most companies. In recent years, with the growing popularity of the Internet and the World Wide Web, companies have started exploring ways to reduce telephone-based customer support costs by directing customers to a website. In this paper, we present the architecture, design and implementation of SupportBeam, an infrastructure to provide web-based customer support. SupportBeam uses a rules-based approach to provide a personalized web experience to users seeking answers to their questions. We use applicability rules to specify relationships between documents and products, and evaluate these rules at runtime to generate a personalized web experience. Our approach provides a scalable and flexible solution for web-based customer support with an improved user experience at low cost.

1 Introduction

Telephone-based customer support has been a major source of expenditure for most companies. According to industry estimates, on an average, a telephone call to the support desk costs a company between $20 - $30 [1]. As a result, the support-related expenses have started to affect a company's bottom line. To reduce telephone-based customer support costs, some companies have started charging customers on a per-call or a per-incident basis. In recent years, with the growing popularity of the Internet and the World Wide Web, companies have started exploring alternative ways to reduce support costs by directing customers to the company's website for support. The intent behind the support website is to provide customers with information most frequently requested on the telephone support desk. The advantage of a support website to customers is that information is accessible on a 24x7 basis. The advantage to a company is that the website reduces operational costs associated with the telephone-based support center. The resulting savings from migrating to a support website

are substantial in spite of some investment needed to operate and maintain the infrastructure associated with the website.

The effectiveness of a support website can be determined by the degree with which it reduces the number of calls to the telephone support center. This reduction is determined by the quality and ease of use of the website, the currency of available information, scalability, and robustness of the infrastructure, and ease of authoring and publishing of content on the website. In this paper, we describe the architecture, design, and implementation of SupportBeam – an infrastructure for deploying web-based customer support. We have deployed the infrastructure for handling support requests for PC hardware and software. The infrastructure can be easily extended to provide support for other types of products. The rest of the paper is organized as follows. Section 2 describes the related work in this area, Section 3 describes the vision and architecture of SupportBeam, Section 4 describes the underlying information model, Section 5 discusses the implementation and performance, and Section 6 presents the conclusions and future work.

2 Related Work

Tradionally, companies have used FAQs (Frequently Asked Questions) as means for providing support information to customers by compiling and publishing a list of most commonly asked questions on the website. The hope is that customers would consult the FAQ before calling the telephone helpdesk, thereby saving companies money in providing helpdesk services to customers. Even though the intent of a FAQ based solution is to provide information, the burden of finding relevant information lies with the user. A user has to wade through a large FAQ database to locate pertinent information. Even though a FAQ provides a central repository for publishing support information, its main disadvantage is that it cannot be personalized according to individual customer needs.

Another popular mechanism used for personalizing websites is using HTTP cookies [6]. In this approach, a web server encodes and saves a user's personalization information in an HTTP cookie and stores it with the client browser. Each time the user visits the website from that client browser, the web server uses the information stored in the cookie to personalize and customize the content that is displayed to the user. There are two main limitations of a cookie based approach. One, the approach is machine specific, i.e., the information stored in the cookie can only be used when a user visits the website from the machine where the cookie is stored. Second, the amount of information that a web server can store in a cookie is limited to 4 kilobytes. Though useful, a cookie-based scheme cannot be used to provide a sophisticated personalization experience.

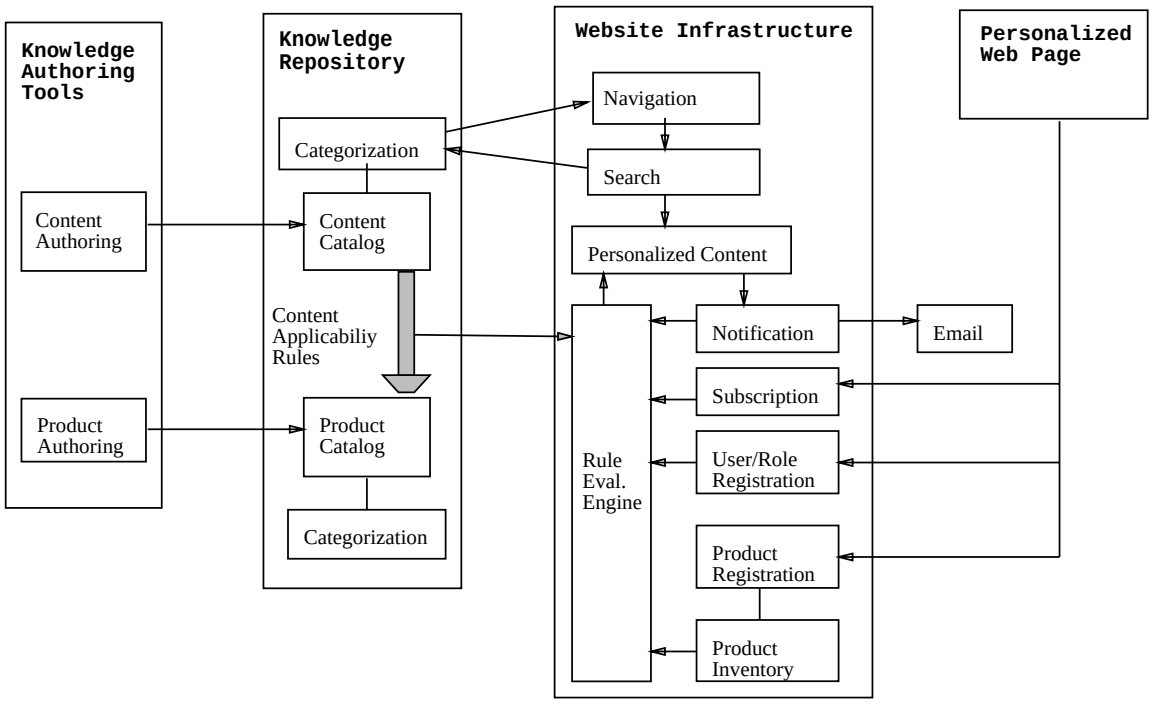

Figure 1: The figure shows the SupportBeam architecture. It consists of four logical parts: A personalized web page, Website infrastructure, a knowledge repository, and knowledge authoring tools.

3 Vision and Architecture

The vision of SupportBeam is to provide a complete end-to-end infrastructure for reducing both the customer total cost of ownership and the support costs while delivering the best service possible to customers. The basic premise upon which SupportBeam is built is – If a business knows what it offers then it can effectively fulfill its customer's needs by knowing who the customer is and what the customer has. Figure 1 shows the architecture of SupportBeam. It consists of four logical parts described below.

3.1 Personalized web page

Providing a personalized web experience is the main "mantra" in SupportBeam. When a user visits the support website, she has the option of either browsing the website as an anonymous user or logging onto the website. When logged in, the user receives a personalized website experience based upon information available in her user profile. The user profile consists of the contact information and a list of products along with installed applications the user owns or manages. For example, a user could register that she owns a ThinkPad T22 with

Microsoft Windows 2000 operating system and Microsoft Office 2000 suite installed. The user profile also contains subscriptions to newsletters and discussion newsgroups, and an option to be notified via email or other means whenever information matching the profile is published on the website.

3.2 Website infrastructure

The website infrastructure consists of several components. The User/Role Registration component enables users to register their contact information and their role in a user profile database. Roles allow a user to separate products that they own or manage in various administrative capacities. A user selects a role when she logs in. A user logged as an "admin" would see device driver updates, while logged as an "end-user" sees only application updates. After the user profile has been created, a user can add products they own or manage to their product inventory using the product registration component. Information contained in the user's product inventory is used to personalize the website. For example, a user, whose inventory only contains one ThinkPad T22 notebook, sees documents applicable only to that notebook and does not see information related to any other models.

At the heart of SupportBeam is a Rule Evaluation Engine that generates personalized content based upon the information available in the user profile and product inventory. Each time a user visits the website, the rule evaluation engine determines the set of documents relevant to the user and populates the Personalized Content component with these documents for display. To simplify navigation and search, category based Navigation and Search components are available. Each user can also subscribe to newsletters and discussion newsgroups via the Subscriptions component. The Notification component allows the users to be notified whenever information matching their profiles is available on the website.

3.3 Knowledge Repository

The knowledge repository contains both the Content catalog and the Product catalog. The Content catalog contains all the support documents available on the website. Examples of such documents in the Content catalog include documents describing bug fixes, frequently asked questions, troubleshooting guides, tutorials, and product descriptions. The Content catalog also contains software such as device drivers, and BIOS updates published on the website.

The Product catalog contains the description and specification on all the products that the website provides information about. In SupportBeam, the products include different models of PCs along with their constituent parts such as CDROM drives, hard disks, keyboards, mice, monitors, cables etc. It also includes all software packages including operating system, applications, and device drivers. The data in the Content catalog is linked via *applicability*

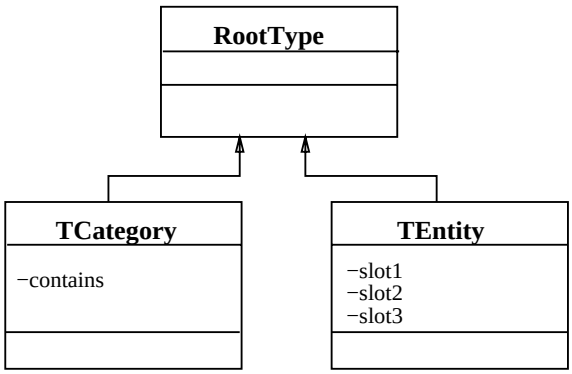

Figure 2: The figure shows the class diagram for the Information Model consisting of an abstract class called RootType, and two concrete classes, TCategory, and TEntity.

rules to other items such as products, roles, discussion newsgroups, and categories. For example, a document describing a device driver update for "Matrox" display adapters is linked via an applicability rule to PCs that contain a "Matrox" display adapter, a category named "Display Adapters," a discussion newsgroup on display adapters, and the role of a "Service Technician". The applicability rules are specified at the time of authoring and publishing a new document in the Content catalog.

3.4 Knowledge Authoring Tools

The effectiveness of personalization provided by the website is as good as the metadata available in the knowledge repository. SupportBeam provides tools to enable authors to easily create and publish information in the Content and the Product catalogs. The tools utilize an information model described in Section 4. The tools require the author to specify the applicability rules for each document in the Content catalog, and the category for each document in the Content and Product catalog. Categorization is used to simplify navigation and presentation of information to users of the support website.

4 Information Model

The main challenge in SupportBeam was to design an information model that was simple, yet flexible and extensible to handle diverse data sets. The primary goal of the information model was to allow for fast and accurate retrieval of information, and to simplify creation and evolution of mass content. The structure of the information model was intentionally kept separate from the presentation logic to allow dynamic publication of content on the website.

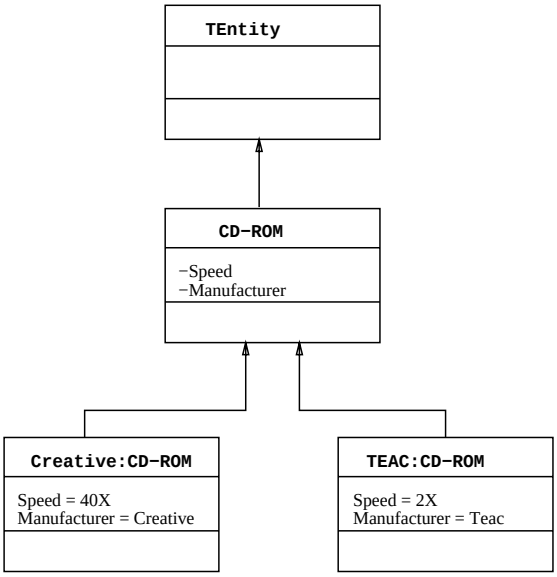

Figure 3: The figure shows an example of a TEntity class called CD-ROM with two slots–Speed and Manufacturer, along with two instances of the CD-ROM entity class called Creative, and TEAC.

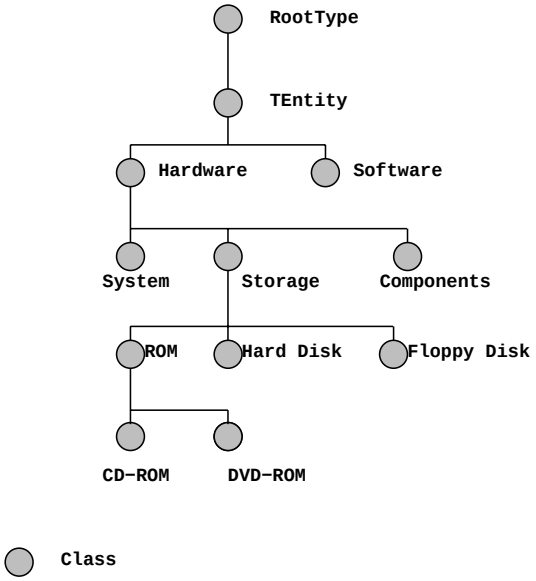

Figure 4: The figure shows an example of the TEntity class hierarchy. The class hierarchy can be arbirarily complex.

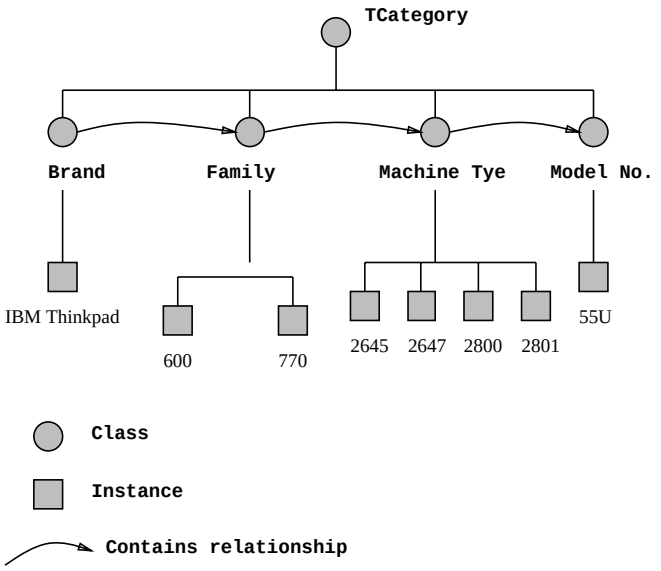

Figure 5: The figure shows an example TCategory class. The figure shows four category types: Brand, Family, Machine Type, and Model Number. For each category type, the *contains* slot has been assigned a value represented by the arrow.

In SupportBeam, we use a frame-based representation [5] for information stored in the knowledge repository. A frame provides a structured representation of an object or a class of objects. A frame can include sets of attribute descriptions called slots. A slot describes attributes of the object or class represented by the frame.

Figure 2 shows the class diagram for the information model. In SupportBeam, an abstract class called `RootType` represents the root frame in the information model. Both the abstract and the concrete frame objects inherit from the `RootType` class. This class cannot be instantiated and provides methods that are common to other objects. The `RootType` class has a property that each instance created from the root class or any of the derived classes is assigned a globally unique identifier. The identifier is used for bookkeeping purposes to keep track of instances at runtime.

The other two frame classes, `TEntity` and `TCategory`, are concrete classes that inherit from the `RootType` class. The `TEntity` class represents concrete object types. It is sub classed to create application specific classes. An instance of a `TEntity` class is called an *entity*. A slot in the `TEntity` class and its derived classes defines a property associated with the class. A slot contains the type of the property, its name, and its value. Classes derived from the `TEntity` class inherit the slots defined in the parent classes. Figure 3 shows an example of a `TEntity` class. In Figure 3, a frame class CD-ROM, representing cd-roms, inherits from the `TEntity` class, and contains two slots – one called Speed, representing the speed of the cd-rom drive, and the second called Manufacturer, representing the manufacturer of the cd-rom

drive. The values for these slots are assigned when instances of the CD-ROM frame class are created. The figure shows two example instances of the CD-ROM class called "Creative" and "TEAC."

Figure 4 shows a more complex class hierarchy. In the figure, the `TEntity` class is the parent for classes `Hardware` and `Software`. Class `Hardware` is further specialized into three classes: `System`, `Storage`, and `Components`. The `Storage` class is further specialized into three classes: `ROM`, `Hard Disk`, and `Floppy Disk`. Class `ROM` is further specialized into two classes: `CD-ROM`, and `DVD-ROM`. Any new slots introduced during specialization are automatically inherited by the derived classes. The reader can see that the class hierarchy can be arbitrarily complex.

The second frame class used in SupportBeam is called `TCategory`. The `TCategory` class defines the type of abstract concepts an entity may be associated with. The `TCategory` class contains slots that represent the *contains* relationship. A class derived from the `TCategory` class is called a *category*. A *category* represents a powerful way of filing and retrieving precise information. A *category* name is not unique as it can be contained in other categories. Figure 5 shows an example of four category types: `Brand`, `Family`, `Machine Type`, and `Model Number`. For each *category*, the *contains* slot has been assigned a value represented by an arrow. For example, the *category* of type `Brand` can only contain *category* of type `Family`. *Category* of type `Family` can only contain a *category* of type `Machine Type`, and a *category* of type `Machine Type` can only contain a *category* of type `Model Number`. In the figure, category `IBM ThinkPad` can only contain categories `600` and `700`. Categories `600` and `700` can only contain categories `2645`, `2647`, `2800`, and `2801`, and so on.

4.1 APIs for the Information Model

To facilitate creation of applications and authoring tools, the information model has been implemented as 11 Java classes that export a programming interface. Table 1 lists the 11 classes along with the associated functionality.

The API classes are all serializable objects and are used to transfer the data from the EJBs to the client application.Use of APIs instead of EJBs provides two major advantages. First, the APIs encapsulate constructs in the information model and its underlying representation. Second, the API classes enable caching of some parts of the data on the client application, improving the performance by preventing redundant queries to the database. To accomplish the latter, SupportBeam uses the database timestamp to ensure that the cached information has not been changed since the time it was cached.

Class Name	Role/Functionality
WAttribute	The class represents the slots in each frame.
WCategory	The class represents the TCategory class.
WEntity	The class represents the TEntity class.
WExpresssion	This is an abstract wrapper class for the WBoolExpr and WRelExpr classes.
WBoolExpr	This class represents boolean expressions.
WRelExpr	This class represents relational expressions.
WTransformer	Helper class.
WTemplate	Helper class
WType	This is an abstract parent class for the `WCategoryType` and `WEntityType` classes
WCategoryType	This class represents Category Types.
WEntityType	This class represents Entity Types.

Table 1: The table shows the 11 Java classes that represent the information model. The APIs exported by the classes are used by the application and authoring tools to access data in the database.

5 SupportBeam Implementation and Performance

The SupportBeam infrastructure is implemented using Java on the IBM WebSphere Application Server 3.5 [3] and DB2 7.2 [2] platforms. Figure 6 shows the multi-tiered architecture of the implementation. All the enterprise data, including the Content catalog, the Product catalog, and user profiles, is stored in DB2. The user registration information is stored in a centralized LDAP [4] based directory. The LDAP repository also performs user authentication. The server-side business logic is implemented using session and container Enterprise Java Beans [7] deployed on the WebSphere Application Server. The server-side presentation layer uses the Java Server Pages and Servlets to construct HTML pages that are sent to the clients. A user uses a standard web browser to view and browse the support website. Java based tools are available for authoring and publishing information to the Content catalog and the Product catalog. These tools use the Enterprise Java Beans to manage information in the enterprise data repositories. By separating the structure of the information model from the presentation logic, SupportBeam enables authors to dynamically publish new content to the website without changing the presentation logic.

5.1 How website personalization takes place?

The main principle behind SupportBeam is to simplify user experience when searching for information on the support website. SupportBeam achieves this through personalization of the information on the website based on user profiles stored at the website. Without

personalization, the SupportBeam infrastructure is no different from a standard website where the user is responsible for navigating and searching through thousands of documents to find relevant information. The heart of SupportBeam is a rule-based engine that performs personalization of content for each user. In this section, we describe the personalization process in SupportBeam. We assume that a user has already registered and created a user profile at the website. The user profile contains contact information, products owned, and interests of the user.

When a user visits a company web site, she selects to "Login to your account" from the web page, *http://www.mycompany.com/*, a Java Server Page *https://www.mycompany.com/LoginPageSSL.jsp* is activated. The web page prompts the user for a *userid* and *password*. The information is processed by a **Login** class where the userid and password is verified using LDAP. If the login information is validated in the LDAP directory then userid is considered valid.

For each company, SupportBeam maintains information about all persons belonging to the company in an ASSOCIATIONS table. A person can belong to only one company. The company ID is stored in the field ID1 and the person ID is stored in the field ID2. The company ID for the person is obtained using the following statement:

```
SELECT A.ID1 FROM ESUPPORT.ASSOCIATIONS A, ESUPPORT.MCOMPANY C WHERE
A.ASSOCIATIONS = 'D' AND A.ID2 = ? AND C.ENTITYID = A.ID1
```

The ASSOCIATIONS table also keeps track of the role of the person within the company. Each role is created as a category and is stored in the category table as CategoryType of "TMUserCat". The category id is stored in field ID1 and the person id is stored in field ID2. A person can be assigned multiple roles in the company. The roles for the person is obtained using the following statement:

```
SELECT A.ID1, C.NAME FROM ESUPPORT.ASSOCIATIONS A, ESUPPORT.CATEGORY C WHERE
A.ASSOCIATIONS = 'C' AND A.ID2 = ? AND C.CATEGORYID = A.ID1
```

The ASSOCIATIONS table also keeps track of all the inventory for a person as well as all the inventory for a company. A person can have multiple inventory associated with him and an inventory can be shared by multiple persons. The person or the company id is stored in field ID1 and the inventory id is stored in the field ID2. The inventories for the person and the company is obtained using the following statement:

```
SELECT A.ID2 FROM ESUPPORT.ASSOCIATIONS A, ESUPPORT.MINVENTORY I WHERE
A.ASSOCIATIONS = 'D' AND A.ID1 = ? AND I.ENTITYID = A.ID2
```

The ASSOCIATIONS table also keeps track of all the products (hardware as well as software) belonging to an inventory. The inventory id is stored in the field ID1 and the product id

is stored in the field ID2. The hardware products belonging to an inventory is obtained using the following statement: SELECT A.ID2 FROM ESUPPORT.ASSOCIATIONS A, ESUPPORT.MHWPRODUCT H WHERE A.ASSOCIATIONS = 'D' AND A.ID1 = ? AND H.ENTITYID = A.ID2

The software products belonging to an inventory is obtained using the following statement:

```
SELECT A.ID2 FROM ESUPPORT.ASSOCIATIONS A, ESUPPORT.MSWPRODUCT H WHERE
A.ASSOCIATIONS = 'D' AND A.ID1 = ? AND H.ENTITYID = A.ID2
```

At the time of user authentication, all the information about the user is retrieved from the database. The information is stored in the class **User**. This class is stored as a session attribute. It is also set to never expire for the current session. The user either has to logout from the site or close the browser session to remove the stored attribute.

```
session.setAttribute("user", user);
session.setMaxInactiveInterval(-1);
```

If the user selects any of the hardware or the software products, the information is passed on to another servlet **SupportServlet** that is used to display the documents associated with the product. The servlet generates the HTML to be displayed on the page. It uses the EJB Session Bean **SServiceBrand** to retrieve the data. The data consists of the categorization of the software or the hardware product as Brand−− >Family−− >Machine Type−− >Model Number. Brand contains many families, each family in turn contains many machine types and each machine type in turn contains many model numbers. For details on the please refer to Section 4 and Figure 5.

All documents are stored as a separate entity in SupportBeam. The ASSOCIATIONS table contains the information about the documents that are associated with a particular Brand, Family, Machine Type or Model Number. The products are also stored as a separate entity and the ASSOCIATIONS table associates a product to a particular model number. To obtain the list of documents associated with a particular Brand, Family, Machine Type or Model Number, the documents associated to all the ancestors as well as all the descendents of the specified node is obtained using recursive query. We describe the inference of applicability rules for product-category association to documents in the next section.

5.2 Inference of applicability for product-category association to documents

An author creates associations between a document and a set of categories. The associations allow the rule engine to infer the applicability between documents and product/categories

through upward and downward propagation. This applicability of documents to multiple categories is saved in a cache called the Bucket Cache. Figure 7 shows an example of inference of applicability for product-category association to documents. The rectangular boxes represent different categories created by instantiating classes: Brand, Family, Machine Type, and Model Number. Numbers from 1-13 represent documents in the Content catalog. Numbers inside the rectangular box represent an association authored between the document represented by the number and the category represented by the box. Numbers inside the ovals attached to the rectangular boxes represent the applicable documents that are saved in the Bucket Cache.

In the example, an association is authored between documents 6 and 7 to category Machine Type "6889". It means that documents 6 and 7 are applicable not only to category Machine Type "6889" but also to any of the ancestor categories of Machine Type "6889" and to any of the descendents (represented by the contains relationship) of Machine Type "6889". The ancestors of Machine Type "6889" are Family "IntelliStation MPro" and Brand "IBM IntelliStation." Therefore, the Bucket Caches of both these ancestors will contain the documents 6 and 7. Since Family "IntelliStation Zpro" and Family "Intellistation Epro" are not ancestors of Machine Type "6889", documents 6 and 7 are not propagated to them. Documents 6 and 7 are applicable to all the descendents of Machine Type "6889" which are Model Number "15K", "15U" and "15V" and all the associated instances of the products. In this example, only multiple descendents are shown. However, a category can have multiple ancestors (can be contained in multiple other categories). Therefore, the propagation will take place upwards through all ancestors.

A product, in turn, is a complex entity and is made up of other entities, for example, display adapters, hard drives, monitors, and processors. If a document is associated with a particular display adapter and that display adapter is a part of another product then the document is propagated upwards to all ancestors of the product containing the display adapter.

5.3 The authoring tool

The authoring tool that has been developed for this system uses extensively the information model API. There is no direct access to the EJBs, all database access is realized through the API. This authoring tool allows the knowledge expert/author to perform all the different actions on the database. This includes creation, removal, and edition of Entity Types/Category Types, and of their attributes, manipulation of templates and the entity creation from these templates, categorization of the entities. The user can link different entities with one another, several different kinds of links being available. For example an entity representing a documentation can be linked to the piece of hardware it refers to with the mention "applies to", and if a new documentation is made for this hardware, this can be linked to the previous one with the mention "replaces". The authoring tool, apart from the creation/edition of database entries, also provides multiple and powerful ways of browsing the information

	Number of items
Categories	41,000
Products	28,000
Entities	2,900,000
User profiles	863,000
Applicability rules	3,200,000

Table 2: This table shows the data distribution in SupportBeam for the performance benchmark

	Time
Cold start	20 seconds
Warm start	$<$ 1 second

Table 3: The table shows the response time to display a personalized web page to a user after she has selected a product from her inventory.

contained in the database. It is possible to retrieve all entities of a certain type that have been categorized the same way, or to retrieve all associations from a particular entity. Based on the rules engine that has been implemented, there is also a filtering system that allows retrieving entities depending on their attributes values. It is for example possible to make queries such as "show me all entities for ibm desktops running Windows NT and with at least 256Mb of memory".

5.4 Performance Measurements

We have performed some preliminary benchmarking to gauge the performance of SupportBeam. The performance was measured on the following configuration. The WebSphere Application Server is running on a dual-processor Netfinity 5600 machine with 1.3 GB memory. The enterprise data server is a DB2 server running on a four- processor Netfinity 5600 machine with 2 GB memory. The total size of support data is 1.87 GB. Table 2 summarizes the distribution of data in the database. The infrastructure contains 28,000 products, classified into 41,000 categories. There are 2.9 million entities in the database, which includes all products, support documents, software updates etc. These entities are linked by 3.2 million applicability rules. The total number of user profiles stored in the system is 863,000.

We measured the response time for a user to be presented with a personalized web page. This operation involves the following steps.

1. A user logs on the website by entering username and password
2. The runtime authenticates the user.

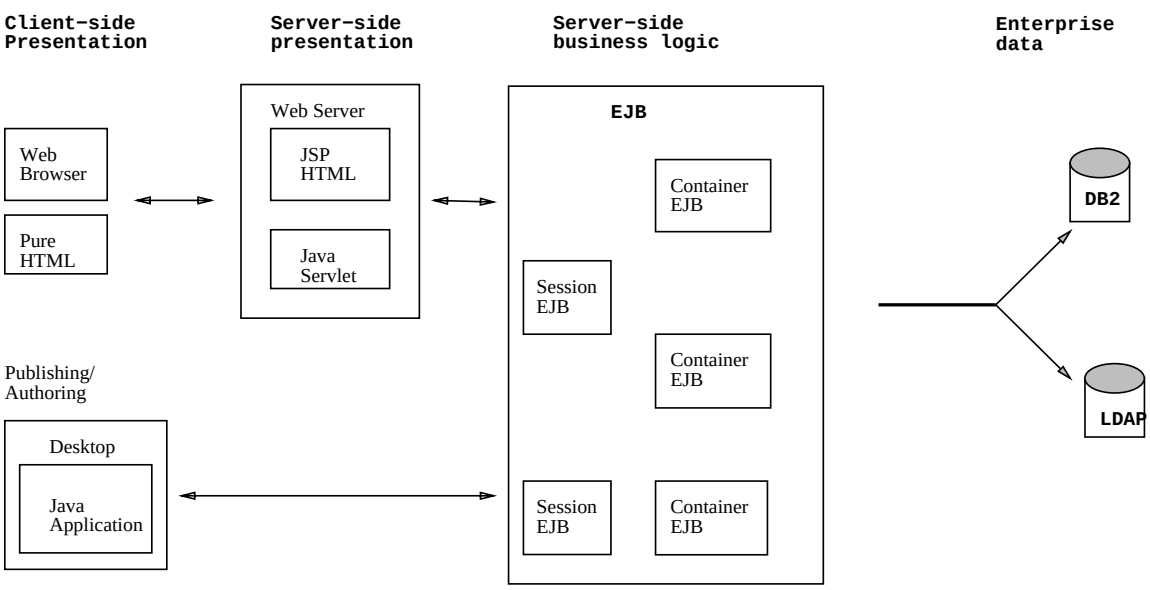

Figure 6: The figure shows the multi-tiered logical diagram for SupportBeam.

3. After authentication, the runtime retrieves the user profile and associated product inventory.

4. The contents of the product inventory are displayed to the user.

5. After the user selects a product from the displayed list, the runtime computes the applicability rules for that product, retrieves relevant documents, organizes the documents into available categories, and presents the results to the user.

We measured the response time to complete Step 5 above. Table 3 shows the results of the performance measurement. The measurement shows the response time to display a personalized web page to a user after she has selected a product from the inventory. For a cold start, the response time is about 20 seconds. This response time is large because the WebSphere Application Server has to load all the Servlets and Enterpsie Java Beans, and the DB2 subsystem has to be primed. Since this cost is incurred only once at system startup time,we chose not to optimize the costs. After the website is operation, in a warm start, the response time is reduced to less than 1 second.

6 Conclusions and Future Work

In this paper, we present the architecture, design and implementation of SupportBeam, an infrastructure to provide web-based customer support. SupportBeam uses a rule-based approach to provide a personalized web experience to users seeking answers to their questions.

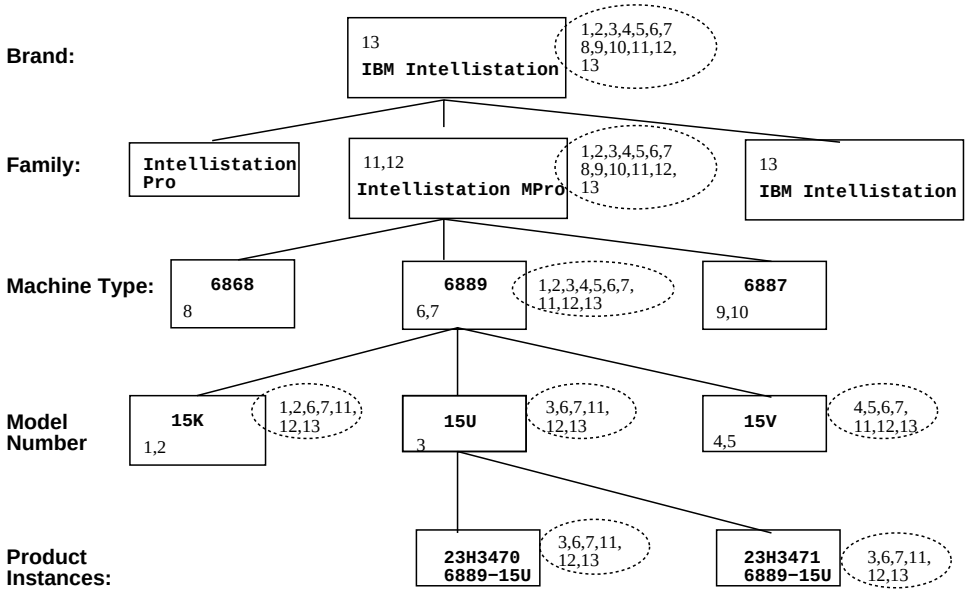

Figure 7: An example of inference of product-category associations for documents. The rectangular boxes represent different categories created by instantiating classes: Brand, Family, Machine Type, and Model Number. Numbers from 1-13 represent documents in the Content catalog. Numbers inside the rectangular box represent an association authored between the document represented by the number and the category represented by the box. Numbers inside the ovals attached to the rectangular boxes represent the applicable documents.

We use applicability rules to specify relationships between documents and products. We separated the structure of the information model from the presentation logic to enable authors to dynamically publish content to the website without changing the presentation logic. Our initial performance results are promising and show that better user experience can be provided via personalization at a reasonable cost.

In the future, we plan to perform a detailed performance tuning of the system, and optimize the evaluation of applicability rules, and the Bucket Cache. We also plan to perform system load tests to measure the response time when multiple concurrent queries are performed.

References

[1] eWeek. Service bots are hotVoyager. *eWeek Magazine*, 2001.

[2] IBM. IBM DB2. *http://www.software.ibm.com*, 2001.

[3] IBM. IBM WebSphere Application Server. *http://www.software.ibm.com*, 2001.

[4] LDAP. Lightweight Directory Access Protocol. *RFC2251*, December 1997.

[5] M. Minsky. A framework for representing knowledge. *The Psychology of Computer Vision*, 1975.

[6] Netscape. Persistent Client State HTTP Cookies. *http://home.netscape.com/newsref/std/cookie_spec.html*, 1999.

[7] Sun. Enterprise Java Beans. *http://java.sun.com/products/ejb*.