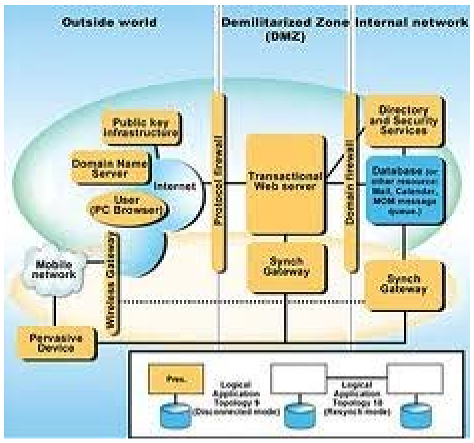
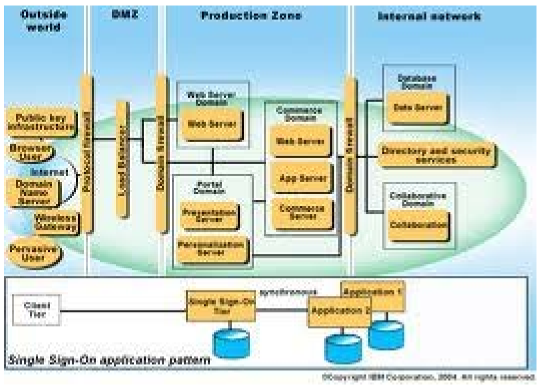
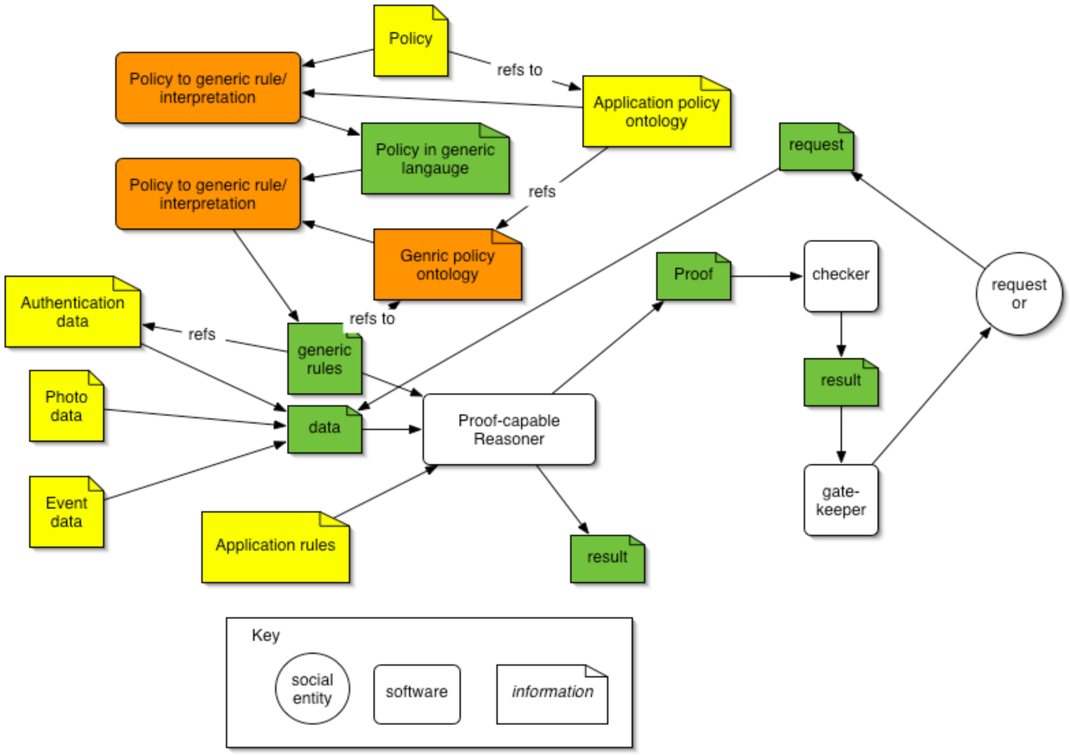
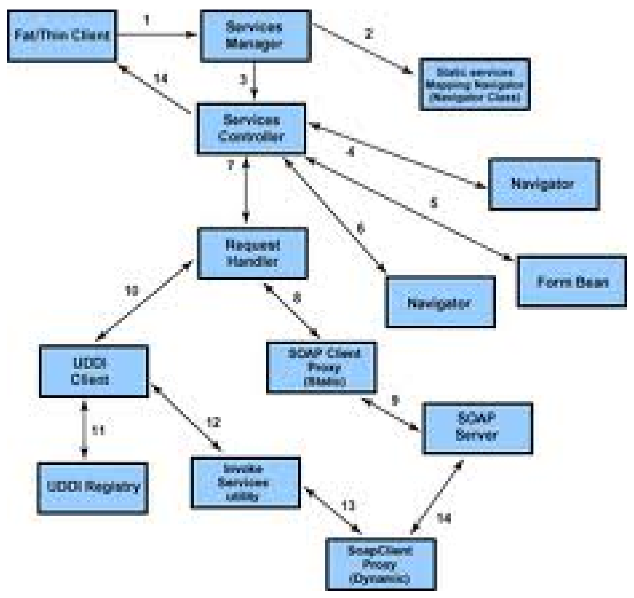
PERVASIVE WEB APPLICATION ARCHITECTURE









A web application is an application that is accessed over a network such as the Internet or an intranet [1]. The term may also mean a computer software application that is coded in a browser-supported language (such as JavaScript, combined with a browser-rendered markup language like HTML) and reliant on a common web browser to render the application executable.

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.

Through Java, JavaScript, DHTML, Flash, Silverlight and other technologies, application-specific methods such as drawing on the screen, playing audio, and access to the keyboard and mouse are all possible. Many services have worked to combine all of these into a more familiar interface that adopts the appearance of an operating system. General purpose techniques such as drag and drop are also supported by these technologies. Web developers often use client-side scripting to add functionality, especially to create an interactive experience that does not require page reloading. Recently, technologies have been developed to coordinate client-side scripting with server-side technologies such as PHP. Ajax, a web development technique using a combination of various technologies, is an example of technology which creates a more interactive experience.

Structure

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to an n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails or Struts2) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

For more complex applications, a 3-tier solution may fall short, and it may be beneficial to use an n-tiered approach, where the greatest benefit is breaking the business logic, which resides on the application tier, into a more fine-grained model. Another benefit may be adding an integration tier that separates the data tier from the rest of tiers by providing an easy-to-use interface to access the data. For example, the client data would be accessed by calling a "list_clients()" function instead of making an SQL query directly against the client table on the database. This allows the underlying database to be replaced without making any change to the other tiers.

There are some who view a web application as two-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server. The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both.[6] While this increases the scalability of the applications and separates the display and the database, it still doesn't allow for true specialization of layers, so most applications will outgrow this model.

APPLICATION SCENARIOS

Over the course of this work we have found that the most efficient and effective way to present the functionality of our system is through application scenarios. In this section we will do exactly that, that is present our system’s functionality through three usage scenarios: first we discuss the case of the user on the supermarket floor, then the home scenario and finally the so-called “on-the-move” scenario.

Supermarket Scenario. The consumer enters the supermarket and selects a “smart” shopping cart (a smart shopping cart is equipped with radio frequency identification (RFID) readers and an on board personal digital assistant (PDA). She identifies to the system using her RFID enabled loyalty card (her user ID is read into the PDA and transmitted to the authentication server) on the cart and gains access by entering her personal identification number (PIN). The system logs her in, responds with a welcome message and then proceeds to present a “suggested” shopping list, based on monitored home inventory and actual consumption data.

The consumer walks in the supermarket aisles and picks up products from the shelves. For example, she may decide to buy a shampoo, which she picks up and places inside her shopping cart. By doing so, the RFID readers on the cart identify the entry of the shampoo bottle and trigger the following sequence: the ID is sent to the back end systems, the systems retrieve information about the specific product and update the display of the shopping list and of the total cost of the shopping cart content. Next, the consumer decides to buy a hair conditioner where the retailer has placed a discount promotion for customers with her profile. When the consumer places the product in the cart, the system displays the promotion on the PDA screen as well as instructions on the shortest path to the aisle and self where the product is held. Later, the consumer decides to remove one can of orange juice from her cart and replace it on the supermarket shelves. The system updates the shopping list with the new total amount and the new contents of the cart.

When the items on the shopping list are exhausted, the consumer proceeds to the check out. When she approaches the till the system rescans all the items in her shopping cart, calculates the total value of the products, displays that information on the till display and prints out a receipt. The consumer pays at the till or charges everything to her account.

Home Scenario. The consumer returns home and places her shopping in her RFID enabled storage (fridge, cupboard etc). New product information is recorded by her home server and consolidated to the home inventory data. The home maintains data on inventory levels as well as consumption. Periodically, the consumer gives permission to her home server to upload her new shopping list to the system.

On-the-move Scenario. While on her way to work, the consumer uses her mobile phone to check which products she needs to replenish before the weekend. After logging in, the list. The consumer decides to add new items to her shopping list for the dinner party she gives on Saturday night. The consumer is happy with her new shopping list. The system displays the total cost of her shopping list at her usual supermarket. The consumer is unhappy with the price and she decides to look for a better price, thus initiating a reverse auction. The system forwards her list to participating retailers and prompts the consumer to define the duration of the auction, which she does. The system sends a confirmation message that the process has been initiated. A short while later the consumer receives offers by different retailers and selects the best. The consumer selects “home delivery” and confirms the order. Later in the day, the system notifies the consumer via SMS to her mobile, that baby diapers are going to run out in the following hours and request confirmation of instant replenishment order. The consumer confirms and the order is placed.

In contrast to traditional web interaction, Web services incorporate some additional overhead. In particular, due to usage of XML, requests and replies are larger compared to traditional web interactions and the need for parsing the XML code in the requests adds additional server overhead. We present a typical web application that requires the transmission of four to five times more bytes if implemented as a Web service compared to the same service implemented as a traditional dynamic program (in our case as an Active Server Page application).

Therefore, compression of Web service interactions is attractive. It is easy to imagine that in the future clients using mobile devices will generate a large percentage of all Web service requests. Although the computing power of handheld devices is increasing rapidly the CPU time required for decompression might eliminate the benefits of compression for these types of devices. We present experiments that quantify the decompression overhead on a handheld computing device with constraint processing capabilities. As expected, mobile clients benefit from compression when the available bandwidth is scarce, for example when the client is connected via GPRS.

But even when resource-constrained devices have better connectivity, the performance loss caused by decompression is almost negligible. Note that mobile clients also might prefer compressed responses since they are often charged by volume rather than by connection time, e.g. in the case of GPRS.

Source : http://nprcet.org/e%20content/Misc/e-Learning/IT/VIII%20Sem/IT1452%20-%20Fundamentals%20of%20Pervasive%20Computing.pdf