PROPHET: A CONTEXTUAL INFORMATION SYSTEM FRAMEWORK

Vipul Singhvi[1] and Michael Terk[2]
[1]*Graduate Student, Department of Civil Engineering, Rice University, Houston TX.*
vipul@rice.edu
[2]*Assistant Professor, Department of Civil Engineering, Rice University, Houston TX.*
terk@rice.edu

SUMMARY

Construction sites are information intensive environments. The timely and satisfactory completion of any project is dependent on the accessibility of project information to the team members. Accessing this information at the point of work, where and when it will be most effective, has been difficult or impossible. This paper describes Prophet, an information system architecture that integrates context-aware web services to provide construction workers with on-site contextual data and services. The system is designed to run on off-the-shelf handheld computing devices and uses wireless network for communication between the clients and the servers hosting services and data. This allows construction personnel to maintain continuous access to data and services as they move around the job site. The system uses the knowledge of user's context to help in managing the complexity of the construction data by proactively tracking current resource requirements and proactively obtaining access to context-relevant information and services. This paper outlines the architecture of this Prophet framework and provides a detailed description of the local and the network caches that supports context-based queries.

INTRODUCTION

Construction sites are information intensive environments. On-site decisions require large amount of design and archival data and knowledge to make rapid on-site decisions, and the lack of access to these resources at the job site often causes these decisions to be deferred. Traditionally, accessing this information at the point of work, where and when it will be most effective, has been difficult or impossible. In addition, the complex nature of the construction process and the dynamic nature of the construction sites can lead construction personnel to overlook important issues that may require quick response. The increasing power and decreasing cost of handheld computing devices makes them appealing for construction applications but their limitations in storage capacity and user interface presents a challenge in accessing, synthesizing and modifying the large amounts of data associated with modern construction projects. To address these issues, we have built Prophet, a contextual information system designed to deliver up-to-date project information at the construction site. This system helps the users to manage the complexity of the construction data by proactively tracking current resource requirements and proactively obtaining access to context-relevant information and services.

One approach that could be used in overcoming the limitations of the handheld devices is to use information about the user's context to anticipate the information and processing needs of the user. This paper describes the software architecture for Prophet, a context-aware system utilizing a combination of handheld computers and web services designed for on-site delivery of knowledge and data. Prophet's architecture has been designed to intelligently link on-site construction personnel to the entire project information repository. The challenge for such a system lies in the complexity of defining, capturing, representing and processing contextual data. The next section describes various steps in handling of context, starting from defining the concept of context for construction sites to final processing of the context.

CONTEXT PROCESSING

Context is an intuitive and subjective concept, difficult to articulate in the form of a definition. However there have been various attempts to give a specific definition of context by enumeration of examples.

For example, Schilit [Schilit et al. 1994] refers to context as location, identities of nearby people and objects, and changes to those objects. Brown [Brown et al. 1997] defines context as location, identities of people around the user, the time of the day, season, temperature, etc. On the other hand, Dey [Dey et al. 2000] argues that definitions that define context by example are difficult to apply in cases when we want to determine whether a type of information not listed in definition is context or not. As a result, they define context as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. In building the Prophet system, we follow a more specific and operational approach to define context. Context is defined as any information that helps in the dynamic characterization of the construction environment. The basic elements of context are: identity of the user, temporal information, location and community. The basic elements of context are referred to as the *raw* context, and are defined as the context that can be obtained from the hardware sensors in or attached to our handheld hardware. In addition to raw context we define *derived* context as context that can be used to refine the characterization of the environment and is generated through interaction with project information.

Establishing context is the first step in context-aware computing. Some elements of context are easier to establish than others. For example, the identity of a user can be established through a login mechanism, or every user could be provided individual equipment with a unique identification code. The temporal information is easily obtained from the operating system installed on the portable computers. Finally, the handheld computer can use a range of communication technologies like Bluetooth and Rendezvous to identify other members of the community by identifying and recognizing neighboring equipments in a given space.

On the other hand, location information is the most challenging and difficult aspects of the context to establish, especially in a changing environment, but is one of the most useful elements for establishing context. While the Global Positioning System (GPS) has been the localization technology of choice for outdoor environments, indoor localization is a much more difficult problem. Indoor localization is an important issue for building construction because a large portion of construction activities occurs in areas that are unable to receive GPS signals. The topic of indoor localization is a subject of active research in computer science, robotics and surveying industries. In our research, we are using the RADAR system [Bahl et al., 2000] developed at Microsoft to provide indoor localization. RADAR is an indoor tracking system and it is built using an IEEE 802.11 standard-based RF wireless local area network (WLAN) deployed at the construction site. The fundamental idea used for localization in RADAR is that in an RF network (wireless LAN) the energy level or the signal strength of a packet is a function of the user's location. Based on the signal strength database or a RF propagation model the system is able to determine the user's location. The main advantage of the system is that it requires few base stations and it uses the same wireless network infrastructure that could be used at the construction site for communication. The following section describes the software architecture of the system.

PROPHET SOFTWARE ARCHITECTURE

Most of the contextual information system architectures, like context toolkit [Dey et al., 2001], infrastructure-centered architecture [Hong & Landay, 2001] and blackboard architecture [Winograd, 2001] are designed to support applications that are part of fixed environments, where the only dynamic component is the position of the user. These environments are well defined, have a fixed set of resources and a well-defined work process. Construction applications, on the other hand, have limited and changing infrastructure, must deal with changes to the physical layout of the site and must be able to adapt to each new trade and discipline as they become involved in the project. To enable context-aware applications to function in this environment, we have designed the Prophet software architecture to address the requirements of the construction site. This architecture is designed to be flexible, robust and extensible in order to cope gracefully with the dynamic nature of the construction process.

The hardware infrastructure used in Prophet consists of a wireless network, a collection of server machines and a "client" application running on the handheld device. The outdoor localization is handled using a GPS receiver attached to the handheld device and indoor localization is implemented through a modified RADAR approach. The schematic diagram of the architecture is shown in the

following Figure. Both the server and the client host software modules that interact with each other to ensure context-aware delivery of project information from the servers to the handheld device.

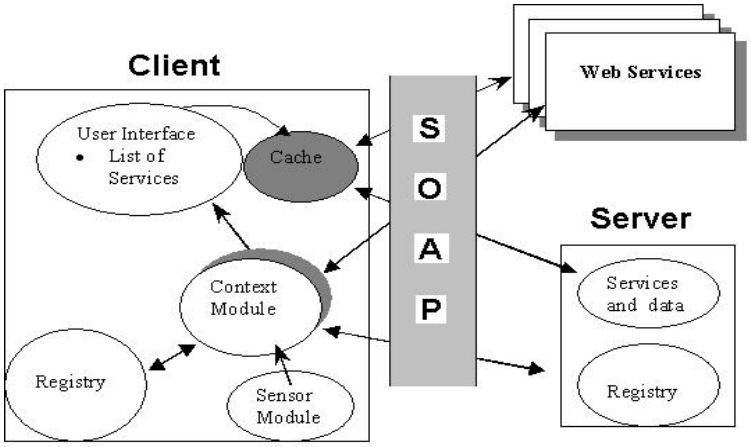

Figure 1 Schematic Diagram of Prophet Architecture

The modules exchange information in form of XML (eXtensible Markup Language) tuples. A tuple is defined as a structured list of values and an XML-tuple is a list of data fields expressed in XML. The list can have any number of fields. A data field or field is simply XML-tagged data. XML provides several benefits as an encoding mechanism for sharing information and functionality between disparate systems. Because XML is essentially text based, it can integrate easily with existing application. In addition, the syntax can be easily read, understood and shared by developers. The communication between the clients, the network services and the main server is implemented using Simple Object Access Protocol (SOAP). SOAP is the new industry standard for communication protocol.

The following paragraphs describe how each module functions in the whole system.

Web Services

Figure 2 Schematics of a Web Service

The Prophet information architecture uses local and network services to provide information to construction personnel. The network services are implemented in form of web services. A Web Service is a programmable application accessible using standard Internet protocols, in this case SOAP. Web Services combine the aspects of component-based development and the HTTP protocol. Like software components architecture, Web Services model the underlying functionality of the services as a black-box that can be assessed and reused without worrying about how the underlying service is implemented. Unlike current component technologies, Web Services are not accessed via object-model-specific protocols, such as Distributed COM (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP). Instead, Web Services are accessed through HTTP and use XML to represent data exchanged between services. The capabilities and inputs of web services are described using Web Service Description Language (WSDL). WSDL is an XML format for describing network services as a set of endpoints operating on SOAP messages containing either document-oriented or procedure-oriented information. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, we have used WSDL in conjunction with SOAP. The above figure gives a schematic for web service architecture.

The Prophet architecture classifies all web services into three categories based on the type of the information they provide. The categories are:

- Context services
- Data services

- Processing services

A context service is defined as a service whose results modify the current context for an application. An example of a context service would be one that takes the current location of the user and determines the floor of a building that the user is currently on. The context services can add a new element to the context, remove and modify elements of the context. The service that identifies the building floor the user is on would add context element *floor-id* when the user enters the building, modifies it as the user moves from floor to floor and removes it once the user has left the building.

A data service is defined as a service that can search and retrieve a specific set of documents from the server. An example of a data service would be a service that uploads design drawings of the construction site based on the user's position on that site. The output of each service would be a set of documents and a range defining what changes to context will render these documents non-relevant. For example, each CAD drawing retrieved based on a position will have a range that defines that, as long as position remains within the specified limits, this service does not need to re-run.

A processing service is defined as a service that does some computation on behalf of the user. These are intended to be services that allow the user to perform quick, back-of-the-envelope calculations, though the architecture does not place any restrictions of the type of calculations produced by the services.

The description of all the services available in the system is stored in the registry module and is represented using a standard descriptive language (SDL). Each service uses the SDL to define its requirement for inputs, its location and the output generated as the result of its processing. The SDL description for a context service includes service name, location of the service, the elements of the context that are needed to execute the service and the elements of the context that are modified by the result of this service. The SDL for a data service includes the name of the service, location of the service, the elements of the context that are needed to execute the service and a set of intervals for the element of the context for which the resulting document is valid. The SDL for a processing service includes the name of the service, the location of the service, the context in which the service is applicable and a set of input and output fields that define the values needed and generated by the service. The next section describes the structure of the registry that stores and processes the SDL descriptions of web services.

Registry Module

The registry module implements a catalogue service listing all web services available to the users of the Prophet architecture. The capabilities and requirements of each service are described using the service description language (SDL). To reduce the communication requirement and to help deal with the interruptions in the wireless connectivity, copies of registry are located both on the handheld device and on the main server. The registry lists all services available to the user regardless of their location. The registry allows the services to be added and removed at any time and provides a mechanism for context-based search of data, context and processing services. The following is an example of a typical registry entry for a data service:

```
- <ServiceList>
    -<service>
        <id>service1</id>
        <type>data generation</type>
        <availability>network</availability>
        <location>http://128.42.42.42/apps/apps.wsdl</location>
        -<context>
            <latitude>latitude</latitude>
            <longitude>longitude</longitude>
            <time>time</time>
        </context>
        -<output>maps</output>
    </service>
  - </ServiceList>
```

Sensor Module

The sensor module provides the mechanism for obtaining the raw context. The sensor module includes a set of interfaces that are used to query the sensors on or connected to the handheld device. The sensor module provides an interface that allows it to populate the context module with the information that defines the raw context. Some of the elements of raw context that could be obtained from widely available sensors include: time, position, orientation and availability and quality of the Internet connection. It is important to note that the notion of sensors, as considered by the sensor module, includes not only sensors able to monitor the environment external to the handheld device but also the sensors that are able to monitor the state of the handheld device itself. The raw context elements that could be obtained by monitoring the handheld device include the identity of the user, the battery level of the device, the current focus of the user and the existence or lack of additional hardware such as digital cameras.

The sensor module interface provides a mechanism for querying the sensors for raw context. The interface provides a mechanism for controlling both what sensors will be polled and the rate at which they will be queried. The sensor module models the raw context produced by the sensors in form of XML tuples. For example, the sensor module polls the GPS to provide the location information of the handheld device. GPS provides the information according to NMEA standard and the sensor module parses the stream of data to get values of latitude, longitude and altitude. It formats the information in form of XML tuple as shown below:

```
- <context>
        <latitude>29.45</latitude>
        <longitude>90.29</longitude>
        <altitude>5</altitude>
        <time>12.00</time>
        <user>vipul</user>
  </context>
```

This information is shared with rest of the modules, which poll the sensor module for contextual information. Normally this information is required by the context module to process for derived context.

Context module

The context module is responsible for building the derived context. After it receives the raw context from the sensor module its objective is to refine the context using the available context services. Based on the network availability, the context module scans the registry, either locally or remotely on server, to find all available context services. Context services are local or network services (web services), which are able to further refine the current state of the context. For example a context service would take in the values of latitude and longitude to provide the project name associated with that location. A detailed description of these services is given in service module section. The context module then iterates invoking available context services until it detects that the iteration did not produce any changes to the context. This process is iterative because of the dependencies between context services. For example, a service that finds the floor-id of the user in a building is dependent on the service that generates building id based on the location of the user. As a result, each iteration is expected to add information to the current context.

To reduce the processing load on the handheld device and the server, the output of each context service is qualified with a set of fields that define the range during which this output will remain valid. The output of each service and the qualifying range are stored in form of a tuple in a context cache, which is located locally on the handheld device.

To simplify the operations of the context module, this module is implemented as tuple space [Ahuja et.al. 1986] with each tuple represented using XML. This approach to the concept of tuple spaces is referred to as XML-spaces. XML-tuples can be written to, and read from, XML-spaces. Writing is done simply by sending an XML-tuple to the XML-space. Reading is done by presenting the XML-space with a "template", which is an XML-tuple, whose tags and values are to be matched against XML-tuples already in the XML-space. If the template matches a stored tuple, a copy of that tuple is returned. The normal read operation returns null if the template does not match anything that is stored in the space. Another read operation, called a scan, returns copies of all the tuples that match a template, instead of

returning just the first one to match, as the normal reads do. In addition, XML-spaces can be used for communications buffers. For example, a client application can register with an XML-space, giving it a template to be matched, and an operation, such as write. If any other client application writes an XML-tuple that matches the template, the XML-space notifies the registered application, sending it a copy of the new XML-tuple. This approach has several significant advantages: the simple nature of the representation ensures the ease of implementation, and, since the senders and the receivers do not interact, XML-spaces are able to gracefully deal with client failure and evolution of the client set [Carrieroet.al. 1994].

Because of the requirement of the flexibility in implementation of our proposed architecture, XML-spaces provide significant advantages in representing and manipulating context. Equally important for use of this approach in handheld devices is the ease of implementation that ensures that the task of context matching will not overburden the processing and storage capabilities of the handheld computer. As a result, the context module is implemented as a XML-space and uses the XML-space operations to manage the context tuples generated by the context services running within Prophet.

User Interface Module

The user interface module is responsible for listing and invoking of all the relevant services that are available to the user based on its context. The user interface module uses the current context to scan the registry and to identify all services relevant to the current context. As a result, the user can be presented with the list of services that are relevant to the current context. The user interface can also identify all services available to the user and highlight the ones that are relevant to the current context. To further reduce the computing burden, the Prophet system uses a data cache to store recently executed requests. As a result, before contacting the data or processing service, this module checks the local cache manager to determine if the results of this request have been cached there. If managed properly, the cache module can produce significant reduction in the processing and communication requirements of web services.

Cache Module

The cache module on the client is divided into two parts: cache and the cache manager. The cache is a repository for files retrieved from data services and a database storing the results of processing services. The cache manager indexes the information about the data in the cache. The cache manager is implemented as a tuple space with each tuple defining the input fields and the service used to generate each piece of data in the cache. Just as in the case of the context module, each tuple has a qualifying range defining the range for which this data remains valid. For example the AutoCAD file for a plan of a building would be retrieved by a single position but would be stored in the cache as long as the user's position does not extend outside the four corners of the drawing. Cache manager is also implemented as a XML tuple space with each tuple in the tuple space serving as a XML representation for data stored in the cache and the period during which it is valid.

IMPLEMENTATION

We have implemented the components of the Prophet architecture and built a client application that uses this architecture to pro-actively deliver project information. The implementation uses an iPAQ 3850 handheld computer with a GPS receiver and a wireless Ethernet card as the client hardware platform. The client software is built using a combination of Embedded Visual Basic and Embedded Visual C++ and uses the Pocket SOAP library to support remote service interaction. The client software uses AutoCAD OnSite View to view CAD drawings, Pocket Word to view Word Documents and custom viewers are

Figure 3 Client: iPAQ 3850 , GPS and Wireless Card

provided for the remaining types of data received from the services. The choice of using an iPAQ was based on the availability of software development kits and APIs, but all implemented code will run on any handheld device that run the Pocket PC operating system.

The server implementation of Prophet uses Microsoft Internet Information Service (IIS) as the server platform. The services are implemented using Microsoft .NET platform and Microsoft Soap Toolkit 3. The project repository, at this time, contains a set of CAD drawings, a set of Microsoft Word documents and a construction schedule modeled using Primavera Project Planner. The current set of services allows the user to query and modify the project repository.

CONCLUSIONS AND FUTURE DIRECTIONS

This paper describes the Prophet software architecture for context-aware information systems. The main contribution of this architecture is the use of context to proactively identify and retrieve information and services that would benefit the user as they move around a construction site. The architecture is flexible, robust and extensible in order to cope gracefully with the dynamic nature of the construction process**.** The Prophet architecture was designed to deal with dynamic changes to the set of services, the context of the user, the communication infrastructure and changes to the data from the sensors that are available to the user. The architecture allows services to be distributed across network and allows graceful degradation in situations where the network connectivity is temporarily unavailable. Finally, the architecture has been designed to run off low-cost, off-the-shelf hardware and uses commonly available technology that reduces the cost of building and deploying this architecture. The system design also reduces the level of computer sophistication required from the users of this architecture.

The Prophet architecture provides a framework for building an information system that aids construction workers in managing their resources intelligently and efficiently. While the initial implementation shows significant benefits of this approach, we have also identified a number of issues for future research. One of the major issues is the ability to adapt to variations in the bandwidth of the available communication. The ability to filter data files based on the bandwidth and retrieving and communicating only a sub-set of information available in those files would make this system more responsive. There are also a number of issues that need to be addressed in improving the ability of the RADAR localization method to handle the specifics of the construction site. We are currently looking at how knowledge bout the construction site, and its evolution, and domain-specific characteristics can be applied to RADAR to improve its accuracy and reduce the effort involved in setting up its network infrastructure. Finally, there are a number of approaches that can be implemented to support the building and distribution of context-aware services such as sending location based services. All of these issues are being considered in our ongoing work.

REFERENCES

Ahuja, S., Carriero, N., and Gelernter, D., [1986] Linda and Friends, *IEEE Computer*, August, 1986.

Bahl, P., and Padmanabhan, V. N., [2000]. RADAR: An RF-Based In-Building User Location and Tracking System. Proc. IEEE Infocom, March 2000.

Brown, P.J., Bovey, J.D. Chen, X., [1997] Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications, 4(5) (1997) 58-64

Carriero, N., Gelernter, D., Mattson, T., and Sherman, A., (1994) “The Linda alternative to message-passing systems”, *Parallel Computing*, 20, 633-655, 1994

Dey, A.K. Abowd, G.D., [200]. *Towards a Better Understanding of Context and Context-Awareness*. CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness (2000)

Dey, A.K. Abowd, G.D., Salber, D., [2001]. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. HUMAN COMPUTER INTERACTION 2001, Volume 6, pp. 287-303

Hong, J.I., Landay, J.A., [2001]. An Infrastructure Approach to Context- Aware Computing. HUMAN-COMPUTER INTERACTION, Volume 6, pp. 287-303

Schilit, B., Theimer, M., [1994]. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5) (1994) 22-32

Winograd Terry, [2001]. Architectures for Context. HUMAN-COMPUTER INTERACTION, Volume 16, pp. 401-419.