Uniform Web Presence Architecture for People, Places, and Things

Philippe Debaty, Debbie Caswell
Internet and Mobile Systems Laboratory
HP Laboratories Palo Alto
HPL-2000-67
June, 2000

Cooltown, Web presence, location, user context, pervasive computing

The Cooltown vision is that people, places, and things have a web representation and that many useful services can be offered by creating a tighter link between the real world entity and its virtual representation.

We defined a horizontal and uniform software architecture for building a Web presence for people, places and things. This architecture enables the dynamic generation of Web contents based on the user context (location, identity, device capabilities), on his security permission, and on the relationships with other Web presences.

Our implementation of this architecture is portable enough to be embedded in the entity that the Web presence describe but also scalable enough to support multiple Web presences hosting. We focused on making the creation of a Web presence easy for non-programmers.

Internal Accession Date Only

Uniform Web Presence Architecture for People, Places, and Things

Philippe Debaty and Debbie Caswell

Internet & Mobile Systems Lab

Abstract

The Cooltown vision is that people, places, and things have a web representation and that many useful services can be offered by creating a tighter link between the real world entity and its virtual representation.
We defined a horizontal and uniform software architecture for building a Web presence for people, places and things. This architecture enables the dynamic generation of Web contents based on the user context (location, identity, device capabilities), on his security permissions, and on the relationships with other Web presences.
Our implementation of this architecture is portable enough to be embedded in the entity that the Web presence describe but also scalable enough to support multiple Web presences hosting. We focused on making the creation of a Web presence easy for non-programmers.

1 Introduction

In Cooltown [1], we believe that the future consists of nomadic people carrying personal communication and web browsing devices interacting with services that are location specific and customized to the user. We believe that unlike other attempts to make computing ubiquitous and pervasive, the web-based approach to communication will be adopted more readily.

The Cooltown vision is that people, places, and things have a web representation and that many useful services can be offered by creating a tighter link between the real world entity and its virtual representation. One of the challenges to making this vision real is how to enable non-programmers to build a web presence for real world entities without requiring programming expertise. Many of the systems that have come before us have been handcrafted by expert programmers. These systems would not easily be created by real people. We are attempting to meet this challenge by creating a general Web presence architecture together with web authoring tools to enable the easy creation of web-present entities.

This paper describes the kinds of useful services that can be provided by bridging the physical and virtual worlds. Armed with this motivation, we identify requirements for the general infrastructure. Next we present the architecture and implementation for the infrastructure under development. We conclude with a discussion of related work, our future goals, and conclusions.

2 Definitions

Here is a short definition of the terms used in this paper:

- **Entity:** an entity is a person a place or a thing in the physical world. For instance, a book is a thing and therefore an entity. A conference room is a place and therefore an entity.
- **Web presence:** a Web presence is a Web representation of one entity. It mainly consists of a set of Web pages describing the entity and accessible through a URL. This URL can be obtained when in the physical presence of the entity. A more detailed architecture of a Web presence is presented in section 4. One entity can have several web presences. An entity that has at least one web presence is said to be ‘Web present’.

3 Motivation and requirements

In this section, we describe the goals we want to achieve with our Web presence architecture and our motivations.

As we said in the introduction, we believe in providing web presence for people, places and things. We imagine the following scenario to motivate the need for these web presences:

A patron enters into an art museum carrying a handheld personal communication device with embedded web browser. The museum has web pages corresponding to each room of the museum which becomes available automatically upon entering the room. Individual paintings also have a web presence. By approaching a painting, the web page for that painting becomes available automatically. The patron visits her favorite painting, and selects the poster-making service available through the web page. A life-sized poster is created for her in the gift-shop while she continues to browse the collection, available for her to pick up upon leaving.

There are many scenarios we can construct of the convenient new world we envision, but there are aspects to this scenario that should be noticed:

3.1 User context consideration

Services such as printing, shopping, and choosing one's next activity can be made more relevant by connecting the services to the user's location context. Shopping can be made more convenient by having an automatic representation of the user's identity, and yet other services such as learning the length of the line at the post office could be offered to an anonymous user.

Today, the majority of web pages that represent real or physical entities simply describe the entity. For example, many retail stores have a web page that describes the merchandise they offer, directions to the store, and store hours. Others might also provide easy email access for asking questions, and still others might offer on-line ordering.

But the user context is rarely taken into account. For instance, currently, there is no system support to benefit from being physically present in a place and on-line at the same time. There is no content adaptation for a user carrying a device with poor browsing capabilities. We believe that there is great value in providing a dynamic, interactive, and custom web representation for a physical entity. It is the bridging of the virtual and physical worlds that makes this vision compelling.

A requirement for our architecture is therefore to support the generation of dynamic contents based on the user context including location, identity and device capability.

3.2 Security

Let's come back to our scenario:

While the patron is visiting the gallery, the manager of the museum wants to know how many customers are currently in the place. Using a Web browser on his PC, he accesses the Web presence of the museum. The service he wants to access is restricted and only accessible to him. After authorizing his identity, he can see a map of the museum with the repartition of customers in each room updated in real time as the customers move around. Next, he wants to know how many paintings from Van Gogh are in the museum and where they are.

The patron's husband is at home and he wants to reach his wife. Using the Web browser on his Web TV, he accesses the Web presence of his wife. She has granted him special access on her Web presence and he can therefore see that she is currently in the museum. He can also see that she is carrying her cell phone but she switched it off. He can now choose between leaving a

message on her cell phone or follow the link to the museum and call there to reach her if it is an emergency.

In this part of the scenario, we can see that some of the contents of a web presence are only accessible to authorized users. For instance, only the husband of the patron can see where she currently is and only the manager of the museum can access the repartition map of the museum. These contents and services are hidden for other users.

A web presence must enable the definition and the enforcement of security policies and must be able to dynamically generate contents based on the user permissions.

3.3 Dynamic relationships between Web presence

Another element of the previous scenario that should be noticed is that there is a great value in enabling relationships between web presences. These relationships can be:

- If the entity is a Place, its related entities are the ones located inside the place. For instance, the devices and persons currently staying inside the physical place. Places can contain places.
- If the entity is a Person, its related entities are the things carried by the person or close to the person, the place where the person is currently located and possibly the directly surrounding persons.
- If the entity is a Thing, its related entities are the person who carries it and the place where it is located.

A Web presence needs to record a link to its related Web presences to provide information about them when needed.

For instance, when a patron enters into the museum, the Web presence of the museum is automatically updated and linked to the Web presence of the patron. This supports the ability to get the count of the number of patrons. In the same way, the Web presence of the patron is updated and shows where the patron currently is. This supports the patron's husband finding out where she is. The patron Web presence could as well be automatically linked to the Web presence of the devices (cell phone, Handheld device…) she is currently carrying. And analogously, the museum Web presence could be linked to the web present devices or things (printers, computers, paintings…) located inside the museum.

Once those links are created, the Web presence can use them to present real time information on the related entities like the map displaying the customers in the museum. Moreover, it is very useful to enable arbitrary queries to the database of related entities. For instance, the manager of the museum wanted to learn about the Van Gogh painting of the collection.

Our architecture must enable the automatic recording and updating of these relationships. It must also support arbitrary queries on these relationships and enable the generation of dynamic contents based on these relationships.

3.4 Additional requirements

- A web presence must be accessible by any kind of HTTP client including but not limited to a Web browser.
- A web presence infrastructure must support the easy creation of a web presence for non-programmers. Graphic designers or web masters should be able to create a web presence using off-the-shelf web authoring tools.

- A web presence infrastructure must support interactions between heterogeneous web presences. A lot of Web pages already existing on the Web describe an entity. Interactions between the dynamic Web presence that we describe in section 4 and these already existing Web pages must be supported.

4 Architecture

In this section, we present a proposed architecture and mechanisms for a web presence for a person, a place or a thing.

4.1 Diagram

Figure 1 shows the architectural diagram that we will describe in this section.

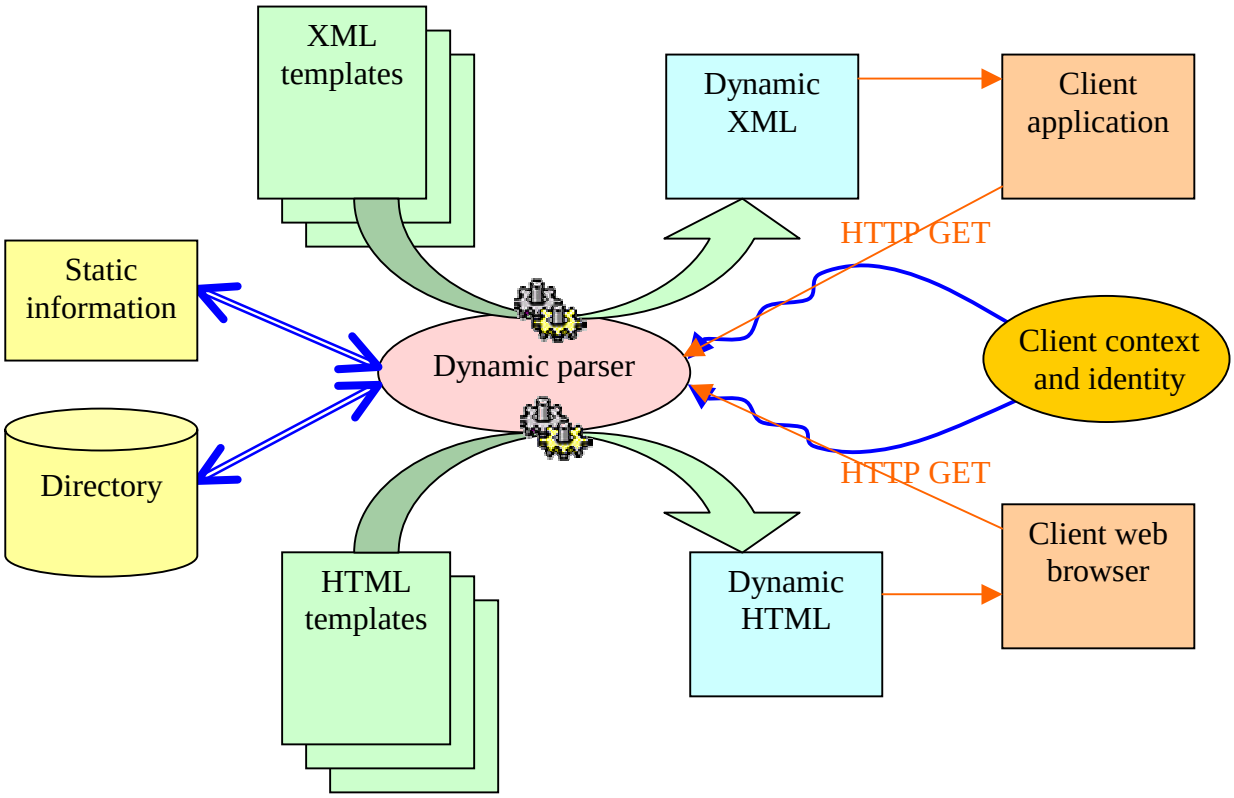

Figure 1: Component architecture for a Web presence.

4.2 Overview

As presented in section 3, we want to generate dynamic contents based on the user context, security policy, and the relationships with other web presences. We also want the creation of the dynamic contents as easy as possible. These requirements lead us to the use of dynamic generation technologies based on the Microsoft Active Server Page(ASP) or Java Server Page (JSP) model. These technologies consist in combining static HTML code with script commands that are processed on the server side and that can generate dynamic HTML. The scripting language used in these technologies is designed to be simple and therefore the dynamic content generation is accessible to Web designers and non-programmers.

To simplify even more the task of the Web presence creator, we chose to create our own dynamic generation system, called the Dynamic Parser, based on the ASP or JSP model.

The Dynamic Parser is specialized for our needs (generation based on the user context, access control, and relationships with other web presences). It is even simpler than ASP and JSP because we don't need to provide all the functionalities that those technologies provide. The set of script commands that we provide is simpler and smaller but sufficient for our needs.

Moreover, it can also be used for dynamic XML generation by combing static XML code with the same script commands used for HTML generation. The XML format is particularly useful for sending raw data to a client application that doesn't need the presentation features of HTML. XML is actually the format that we chose to represent relationships between Web presences (see section 4.3.4).

XML descriptions and HTML pages are treated the same way in our architecture. To avoid the repetition of information in both formats, common information can be stored in a component called *Static Information* (see section 4.3.5). Eventually, when XML and XSL will be commonly supported by both Web browsers and authoring tools, HTML pages will disappear and only the XML description generation will remain.

The Dynamic Parser (see section 4.3.6) is the central component of our architecture. It generates the contents on the fly based on the HTML or XML templates (combination of HTML or XML with script commands. See section 4.3.2 and 4.3.3) and on information coming from the following other components:

- The Directory, which is the component that handles the automatic storage and updating of the relationships with other Web presences. See section 4.3.4 for more details
- The Client context and identity, which is sent within the HTTP get request of the client. See section 4.3.1 for more details.
- The Static information component, which handles security policies configuration and static information about the entity common to both HTML and XML pages. See section 0 for more details.

4.3 Components of the architecture

4.3.1 Client context and identity

Information about the client's context must be presented with each URL accessed using cookie mechanisms. This process is more described in [2]. By context we mean the user's identity, location, and the capability of the browsing device. This information is used by the dynamic parser (cf. 4.3.6) to create a customized view of the entity for this client.

4.3.2 HTML templates

An HTML template contains HTML code combined with script commands that are processed by the Dynamic parser when a query is made. A dynamic and custom HTML page is then generated.

These HTML pages are accessible by a client using simple URLs. Actually, the URLs do not reference directly a static HTML page but a method of the Dynamic parser that will dynamically generate this HTML page. Only the URL referencing the home page actually needs to be provided. The client can then access the other web pages by following links in this home page.

4.3.3 XML templates

The XML descriptions of an entity are used by a client application or another entity's web presence to get raw information about an entity without the presentation features of the HTML format. An XML template contains XML code combined with the same script commands used for the HTML templates.

The XML descriptions are also accessed by a client using URLs. The URLs reference the method of the dynamic parser that will dynamically generate them.

If there are several XML descriptions for a same entity, one of them must be the default XML description. The others can be more or less detailed ones depending on the needs of the client. The client can choose among those different XML descriptions by electing the desired URL in the interface of the Web presence. For instance, there could be one XML description called 'default', another one called 'basic' and another one called 'full'.

The URL referencing the default XML description of the Web presence is essential in our architecture because it is used as the unique key to reference this Web presence in interactions with other Web presence (see section 4.3.4).

The XML descriptions must contain some well-known attributes that will be commonly recognized by every client applications and especially in interactions with other Web presences. It can also contain specific attributes that will be used in arbitrary queries (see section 4.3.4.2) or specific client applications.

4.3.4 Directory

The Directory is the component that records relationships with other Web presences. Entities related to the current entity are called 'resources' of the current entity.

The directory of an entity is just a list of keys referencing each of its resources (the key of an entity is the URL referencing its default XML description as seen in 4.3.3). Using the key of a resource, the Web presence can retrieve all the useful information concerning this resource. This concept of a unique key referencing the rest of the data is analog to the primary key concept in a relational database.

For efficiency purposes, the Directory also caches the XML description of all the resources. The queries made on the resources (see section 4.3.4.2) are then much faster because the data doesn't need to be retrieved from the network. This caching is analog to the indexing in relational databases in the sense that it increases the efficiency of queries.

4.3.4.1 Registration

The directory component also provides web accessible methods to register and unregister an entity as a resource. An entity needs to register as a resource of another entity when enters into a relationship with the other entity. For instance, let us consider the scenario of a Web present person entering into a Web present room:

The person may just want to access information about the place or they may also want to become a resource for the place. In the first case, the person just needs to get the URL for the Web pages of the place and access it.

In the second case, the person's Web presence needs to be registered in the directory of the place. This is done using the URL for the register method of the place's directory. This method adds the person's Web presence in the place's directory and also adds the place's Web presence in the person's directory. At the end of this process, the person is in the place's directory and the place is in the person's directory.

The calling of the registration method can be manual but there is a great value in enabling automatic registration by using external automatic discovery mechanisms. This is more described in [2].

4.3.4.2 Queries

Another functionality of the directory is to provide web accessible methods to make arbitrary queries into the resources of the entity. For example, an administrator of a place may want to have a list of all the devices of a specified type that are present in the place, accomplished by querying the place's directory. Those methods could also be used to get the place where a person is currently located by quering the person's directory.

4.3.5 Static Information

Some information about an entity is common for both XML descriptions and HTML pages. For instance it can be the name of the entity, its general description, its type (people, place or thing)... This information is stored in the Static information component. The script commands integrated in the XML or HTML templates can then retrieve this information and generates dynamic contents based on it.

The Static information component also handles the security policy configuration. See section 4.4 for more details.

4.3.6 Dynamic parser

This component enables to dynamically generates the HTML pages and the XML descriptions of the entity. This gives the ability to integrate the contents of the directory of the entity in the Web pages or in its XML descriptions. It also permits application of security policies on what is output in the Web pages or in the XML descriptions.

Here is how it works:

- A client web browser or a client application makes a HTTP request to the Dynamic parser. In the request, the client specifies as a parameter, the Web page or the XML description he is interested in. The request also contains the client context and identity.
- The Dynamic Parser picks the corresponding Web page template or XML template.
- The Dynamic parser processes the template and generates a resulting page. The script commands in the template can for instance query the Directory or static information on the entity. They can also specify a security policy depending on the user context or identity.
- The resulting page is sent back to the client browser or to the client application.

4.4 Security

There are many security checkpoints needed in our infrastructure. The security component is not represented in our diagram because it is closely integrated with it at different points. This section describes the need for access control, privacy, integrity, non-reputability.

There are several roles that users play with respect to a Web presence. There is an administrator role that is allowed to create HTML and XML template, establish security policy, and manually enter resources into the entity's directory. There are users who may choose to register as a resource with the entity's directory (Resource role), and there are users who are viewers of an entity's web presence. These different roles require different security mechanisms.

The access control policy specified by an administrator determines which consumers may view an entity's web page or XML description, and in addition may specify which parts of the web page or of the XML description may be viewed.

To accomplish this access control policy, the administrator can define different groups of users that will have different permissions. For instance, one group can be 'Family member', another can be 'HP employee', another can be 'Administrator'... Then the administrator can associate user credentials to each group.

Then, in the HTML or XML templates, there can be script commands specifying which part of the HTML or XML page is shown to which group of users.

The configuration of the access control policy is stored in the *Static Information* component.

Another policy decision the administrator makes is whether anonymous access is permitted by viewers. If anonymous access is allowed, then the anonymous view can differ from the authenticated view using the dynamic parsing mechanisms. The anonymous view can present just the information that anonymous viewers are permitted to see. In that case, it is recommended that the Web page provide a *log in* button to allow viewers to authenticate themselves and thus gain access to the additional information in the web page. Typically, if anonymous access is allowed, the URL that references the anonymous content is considered to be the URL of the entity. For example, it is the anonymous URL that will be obtained in the physical presence of the entity.

Additional security can be obtained by requiring the Web presence to only accept HTTPS requests (Secure Sockets Layer [12]). By using at least server authentication, a secret key is negotiated and all communication is encrypted in this key. Encrypting the communication ensures privacy which is especially important in a wireless environment. Furthermore, it ensures that any tampering with the communication will be detected. If mutual client/server authentication is used, then we achieve non-repudiation as well. In a highly secured environment where trust is of paramount importance, a user's identity implies both privilege and responsibility. Special privileges may be granted certain user identities, and those users are also liable for damage done as a result of receiving those privileges. For example, a school place might grant the teacher the ability to change students' grades. Now, imagine teacher gives away her credentials to her students who can now masquerade as teacher and change their own or other students' grades. The security breach would be traceable to the teacher, and the teacher would be liable for any damages.

In addition, there is another kind of privacy that we need to consider: the privacy of relationships among entities. In other words, which entities are allowed to know about a relationship among other entities? For example, are other users allowed to know the phone number of the cell phone I carry? Are they allowed to contact me on that cell phone even if the system hides the phone number? Does a person entering an adult bookstore with web presence want other people to know he/she is there?

The generic infrastructure provides the ability of a Web presence to permit anonymous viewers and resources. An anonymous viewer is one that can view the entity's web content without the viewer's identity being disclosed. A Web presence, however, will choose to use this feature or not as a matter of policy.

We also provide the ability of resources to disguise their real identity. A resource is registered into another entity's directory for the purpose of allowing interaction between the entities. As such, the resource must provide a means of contact. However, the name associated with the resource need not be authenticated, and thus any name can be chosen. An example of this is how most chat rooms work today. Participants in the chat room must register their interest in

participating in the discussion so that others can contact them. However, the participants are free to choose whatever name and description of themselves they want whether it's true or not. Whereas the administrator for a science fiction convention place supporting an on-line chat service might elect to allow role playing among participants, the administrator of a company's conference room might not elect to provide anonymity (or disguised identity).

The ability to view an entity's content anonymously protects the privacy of the viewer. This can be especially important to provide in a place Web presence. People entering a place might not want others to know they are there. They need to be able to view the place's web contents without having to authenticate or register their presence.

Of course, the Web presence must ensure the data integrity of the directory and static information as well.

4.5 Integration of pre-existing Web pages

One requirement for our architecture is to support the integration of already existing Web pages describing a physical entity and providing services related to this entity. For instance, a printer might already be described by some Web pages provided by the manufacturer of the printer. A person might already have his personal Web pages served up by his ISP or by his company Web server. In this case, what if this person enters in a room equipped with our Web presence architecture and wants to become a resource of this room?

There are two solutions to this specific case that are applicable to others:

- The pre-existing personal Web page of this person can be easily moved to a server hosting Web presences with our architecture. These Web pages will be considered by the server as HTML templates (see 4.3.2) without script commands. A short XML template describing the person needs to be manually created to enable interaction with other Web presences. Later, the person can benefit from the dynamicity of our architecture by adding script commands enabling security policies or contents generation based on user context and relationships with other entities.
- A simple and static XML description of the person referencing his Web pages can be manually created and served up by any Web server. When the person wants to register in the directory of a place, he just needs to send the URL of the static XML description as the key for his Web presence. The XML description will contain the URL of the static HTML page.

5 Our implementation

An important point to keep in mind is that the Web presence implementation of an entity does not need to be embedded in this physical entity. For instance, the web presence implementation of a person does not need to be embedded in his PDA or personal computer but it could be anywhere in the network. However, there might be advantages of embedding the Web presence implementation in the entity itself. For instance, if the entity is a device like a printer, a Web presence implementation embedded in the printer can have direct control over the printer.

Therefore, there are two different approaches to implement this architecture. A first solution is to have a big server hosting and serving up a lot of Web. The second solution is to have an implementation running only one Web presence and designed to be embedded in the physical entity itself and therefore small, portable and easy to install. This second implementation could then run on a simple PC or on appliances such as a printer. Both solutions can coexist. So far,

we have developed a first version of the second solution and we are in the process of developing the first one. It is planned to be ready before Fall 2000.

We use HP's Chaiserver platform as the basis for our implementation. There are several advantages to using this platform. The Chaiserver platform implements the web-appliance connectivity architecture [8]. Chaiserver provides basic distributed systems services such as security, loading, and notification. We depend heavily upon the Chai gatekeeper [3]. Developers create objects called Chailets that are invoked when they implement the object type specified in the HTTP request. The base chailet from which all chailets are derived, provides HTML helper functions that make it easy to create dynamic html. Finally, Chaiserver was designed to run in web-appliances; it is portable and small. However, Chaiserver also runs on large HPUX platforms. This means that our implementation can scale to meet the requirements of the application.

5.1 Authoring Environment

An HTML template is authored in Front Page or any HTML authoring tool and designed to create the content look and feel. Several alternative representations can be authored by creating distinct sections of the web page demarked by the tag directives described below. The dynamic parser is passed a vector of argument/value pairs that it uses to instantiate the template. If the *Authorize* argument appears in the list, then the user has been authenticated. If the *UserInSpace* argument appears, the user's presence in the place has been authenticated. The *UserName* argument is another built-in argument that contains the name of the credential of the user if it is not anonymous. Other than those two built-in arguments, the rest of the arguments are those that are passed in the query string when fetching the URL.

In addition to the argument vector, the dynamic parser can also query the directory for resources that match certain criteria. For matching resources, the value of specified tags may be substituted for the tag name.

The script commands in the templates are wrapped in a pair of <% and %>. Here are some examples of script commands that can be incorporated in the HTML or XML code:

`<%=host_url%>`	The host_url directive is replaced with the URL of the host on which the dynamic parser runs: http://<hostname>:4242
`<%=login_url%>`	This directive is replaced by the URL of the authorized access point to the Dynamic Parser.
`<%=logout_url%>`	This directive is replaced by the URL of the anonymous access point to the Dynamic Parser.
`<%=args.ArgName%>`	This directive is replaced by the value of the argument ArgName.

`<%if ARGUMENT then%>`
`body1…`
`<%else%>`
`body2…`
`<%/if%>`
	The 'if' statement determines which of two alternative sections will be displayed. If the ARGUMENT specified is present in the argument list, then body1 is displayed; otherwise, body 2 is displayed.

`<%loop getResources([type])%>`
`body…`
`<%/loop%>`
	This directive starts a loop. It gets the resources corresponding to the specified type. The type parameter is optional. If no type is specified, all resources are returned. The body of the directive is displayed for each resource. Within the body, the tags <%=current.TagName%> are replaced by the corresponding tag value of the XML description of the current resource.

`<%loop getArgs([ArgName])%>`
`body…`
`<%/loop%>`
	This directive starts a loop. It gets the arguments received by the Dynamic parser. If the ArgName parameter is specified, only the arguments starting with ArgName will be

	returned. The body of the directive is displayed for each argument. Within the body, the tag <%=current.name%> is replaced by the name of the current argument and <%=current.value%> is replaced by the value of the current argument.

5.2 Security

To accomplish access control security, the administrator must first set up a security realm which is the collection of authenticated identities that will be used to specify access control policy. This is done by using the Chai gatekeeper [3]. Next, the administrator fills out a Web presence configuration form for configuring security policy. The administrator configures the name of the realm created for the entity. Then, the administrator can define tag names to denote groups of user identities that can be used in defining information hiding rules on sections of the entity's web page. For example, a family home Web presence might want to specify that only family members may view the family calendar and photo album. The administrator might create a *USERISFAMILY* tag and associate with it the identities of each of the family members. Then, in the HTML template, this tag can be used to define a region of the web page that only family members may view:

<%if USERISFAMILY then %>
link to family calendar
link to family photo album
<%/if%>

A similar mechanism for selectively disclosing sections of an XML description are possible as well. The idea is that certain attributes of an XML description may be selectively disclosed or hidden based on some access control policy.

6 Related Work

Because our solution is web-based, our implementation looks very much like any other web application: it receives HTTP requests containing control information and data, processes HTML content (sometimes with embedded commands), retrieves information from other back-end services, and returns a dynamically generated web page. As we said earlier, we based our solution on the ASP [4] or JSP [5] model. Other technologies like Java Servlets [6], Common Gateway Interface [7], PHP [9] or SSI [10] enable on the fly Web page generation but these solutions are less accessible for non-programmers.

Others have built context managers that were specific to creating Web presence for places, but not appropriate for people or thing Web presences [11]. Spreitzer and Theimer [13] describe an architecture to provide location based services in pervasive computing environment but their architecture is not Web based and therefore less easily deployable and accessible to non-programmers.

A lot of work has been done on virtual places such as the mushroom project [14]. Our work differs from this since we focus on a tighter link between the physical reality and its Web representation.

Rather than focusing on creating the best solution for a particular application, we have concentrated on building general-purpose mechanisms that are common to providing web presence for people, place, *and* things. In addition, we are trying to lower the barrier to adoption of Cooltown technology by making it very easy for non-programmers to create web presence for people, places, and things they care about. We have favored techniques supported by off-the-shelf authoring tools.

7 Conclusion and future work

We have defined both an architecture for a Web presence and a model for the interaction mechanisms between Web presences. These elements enable us to create a Web representation of the physical world. We also described an implementation solution for this architecture. The work on this project is still in progress. Some security requirements described in our architecture have not yet been implemented. An XML template authoring solution easy to use and specific to our needs has to be found.

8 Acknowledgements

Our Dynamic Parser evolved from work originally done by Jeff Morgan. We wish to recognize and thank our reviewers Tim Kindberg, John Barton and Gary Herman who gave us invaluable suggestions for improvement and support. Tim Kindberg also helped us to refine this architecture.

9 References

[1] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra. *People, Places, Things: Web Presence for the Real World,* http://www.cooltown.hp.com HPLabs Technical Report HPL-2000-16

[2] D. Caswell and P. Debaty, *Creating a Web Representation for Places.* Submitted to HUC2000.

[3] Hewlett-Packard Chai, http://www.internetsolutions.enterprise.hp.com/chai/

[4] Microsoft Active Server Page, http://msdn.microsoft.com/workshop/server/asp/ASPover.asp

[5] JavaServer Page, http://www.javasoft.com/products/jsp/index.html.

[6] Java Servlet, http://www.javasoft.com/products/servlet/index.html

[7] Common Gateway Interface (CGI), http://web.golux.com/coar/cgi/

[8] Morgan, J. http://cooltown.hp.com/papers/jam/WebDeviceAccess.htm

[9] PHP, http://www.php.net

[10] SSI, http://www.sonic.net/~nbs/unix/www/ssi/

[11] A.K Dey, G.D. Abowd, and D. Salber. *A Context-Based Infrasructure for Smart Environments*. Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments, Dublin, Ireland; Dec. 13-14, 1999.

[12] Secure Sockets Layer (SSL), http://home.netscape.com/eng/ssl3/index.html

[13] M. Spreitzer and M. Theimer . *Providing location information in a ubiquitous computing environment*. In Proceedings of the 14th International Conference on Distributed Computing Systems, pages 29-38, Poznan, Poland, June 1994.

[14] T. Kindberg, *A framework for collaboration and interaction across the Internet*, the International Workshop on CSCW and the Web, Sankt Augustin, Germany, February 1996, http://www.dcs.qmw.ac.uk/research/distrib/Mushroom/publications/CSCWWeb.html.