Patterns of Web Site Structure in UriGraph

Hrvoje Simic
Department of Telecommunications,
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10 000 Zagreb, Croatia
hrvoje.simic@fer.hr

***Abstract*—This paper introduces a Web site structure model called UriGraph and, using the model, describes several important patterns of site structure. Web site structure is defined as the collective information about the identity, identifier, position and composition of every resource constituting the Web site. UriGraph models the site's resource identifiers and through them the resource identity and composition, and indirectly the resource position. UriGraph is designed specifically for the Web and it is compatible with the current practice. It can be represented graphically and as an XML document.**

Keywords—UriGraph; URI; Web site structure; Web resource; Web application; XML; patterns

I. Introduction

The term "Web site structure" is often used, but seldom defined. This paper gives a novel definition of Web site structure embedded into a wider theory of Web sites and applications. The paper discriminates between the structure of Web site content and the navigation schema on one hand and the Web site structure on the other, although all three concepts are mutually dependent.

The main purpose of this paper is to present a new Web site structure model. The model works by analyzing the Web resource identifier (the URI) to provide clues about the requested resource's identity. This identity is then used by the components of each resource to determine what content they should generate. Creating passages through the structure graph during the identifier analysis allows for inheritability of components, similar to propagating permissions in a file system.

UriGraph can model the structure of any Web site, but it is also a foundation for a specific software engineering approach aimed at reducing inner redundancy, facilitating development and maintenance of larger sites. Specific features of UriGraph need a special Web server to deploy (like Wance [12]), but the basic graphic representations may be used on any platform.

The paper is organized as follows: the next section defines the Web site structure as one of the Web application ingredients. The definition of UriGraph is covered in the third section. The fourth section shows how the model can be used to describe several important patterns of site structure. The paper also includes a section on related work, conclusion and notes on future work.

II. Web Site Structure

A Web application is a composition of two distinct systems: the target system and the Web adaptation system (Fig. 1). The target system contains business logic and data specific to a particular need and independent of the technology used to access it. Its abstract interface allows it to be used through various systems, such as GUI applications, Web sites, Web services, e-mail, or WAP. The Web adaptation system is used to interface between the target system and the Web, allowing the whole system to be perceived as a Web site.

A Web site is a logically coherent collection of Web resources. A Web resource is a persistent source of useful information, used for retrieving information or processing data, and available through the network using the Hypertext Transfer Protocol [11]. A resource that only processes data (changes the server's state) is called an operational resource, as opposed to the presenting resource that handles safe HTTP methods and is used exclusively for information retrieval. Term "Web page" is used for a resource that presents information in the form of hypertext.

A. Web Application Ingredients

There are four "ingredients" of a Web application:

- content – the basic information supplied by the application, created and maintained by site content authors;
- look – means of presenting the content clearly and attractively to the user, created by visual designers and usability experts;
- functionality – means of processing the content, created by programmers;
- structure – the central ingredient catalyzing others, designed by Web information architects.

Content and functionality are defined in the target system, while look and structure are defined in the Web adaptation system.

This paper defines Web site structure as the collective information about the four qualities of every resource constituting the site: its identity, identifier, position, and composition.

B. Web Resource's Structural Qualities

1) Resource Identity

Every Web resource should represent (e.g. provide information about) a concept. Identity of the resource corresponds to the intension of the represented concept, the set of all attributes necessary and sufficient for defining that concept.

As an example of a Web resource we can take a message posted on a Web discussion group called Dylan. Definition of that concept, from the context of a Web site that hosts several discussion groups is "message number 5543 on group Dylan".

2) Resource Identifier

The identifier should convey the information about resource's identity, nothing more and nothing less. Today's common identifier standard for the resources on the Internet is the Uniform Resource Identifier (URI) [2]. There are two parts of URIs relevant for differentiating between the resources on the same site: the path and the query, so most of the examples in this paper will use URIs reduced to these two parts. The path consists of strings called path segments delimited by a slash. Query also has segments: these are delimited by an ampersand. Each query segment includes a name and optionally a value, separated by the equals sign. For example, the URI "/quote/delete?id=3&content-only" is composed of path segments "quote" and "delete" and two query segments: one with name "id" and value "3" and other with just the name "content-only".

The URI is widely exposed to the user and therefore a vital part of the Web user interface [7]. URIs are typed into the address bar of the browser, read in documents and advertising, spoken, remembered and manipulated. The assignment of specific URIs to the resources on the site and thinking up rules for mapping URI-subspaces to classes of resources has become known as the *URI design*. Important requirements for a well-designed URI are: meaningfulness, persistence, good structure, shortness, readability, memorizability, and pronounceability [12].

A good identifier for the above mentioned resource on the Web site would be "/group/dylan/message/5543".

3) Resource Position

Resource's position in the Web site defines its relationships with other resources on that site. The principle of relating resources to concepts ties the resource position tightly to its identity – relationship between resources is similar to relationships between represented concepts.

Our example resource representing a message is subordinate e.g. to the resource "/group/dylan" which represents the discussion group, and also to the home page of the site. The message 5543 resource is closely related to other resources such as those representing replies to message 5543, the author of the message, etc.

4) Resource Composition

It is useful to regard the Web resource as a composition of one or more separate, self-sufficient, encapsulated logical parts. Those parts are called resource components. The information about which components are included in a specific resource as well as the relationships between individual components is called the composition of the resource. Also, a component has its own identity, embedded in the identity context of the resource in which the component is placed.

An example Web page representing a message may contain several components such as: the message text, a list of replies, and author's photo, all receiving the necessary context information from the resource identity. If the page representing a group contains a photo of a "featured" member, that component should have extra clue on who to display, since the group's home page identity doesn't point to any member specifically.

C. Web Application Architecture

A broad architecture for Web applications used in UriGraph is shown in Fig. 1.

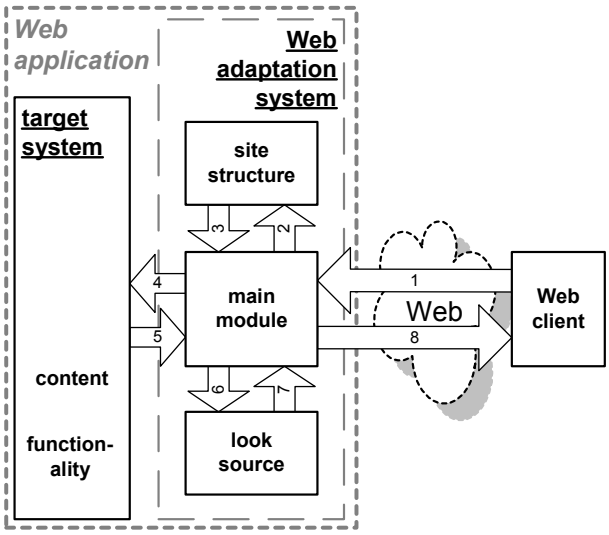

Figure 1. Web application architecture.

A typical request-response cycle proceeds as follows: (1) a client sends an HTTP request; (2) resource identifier is extracted, along with other relevant information from the request; (3) resource identity, composition and position are deduced from the identifier and the site structure; (4) the main module requests content or a transaction for each component of the resource; (5) content data (e.g. Web page content in XML format) or transaction outcome is returned; (6) if appropriate, the content is combined with the "look" of the resource to increase its readability and attractiveness for the user; (7) each component's content is combined with its "component look" (e.g. using XSLT) and all pieces are arranged into the resulting document (e.g. XHTML) using resource layout definition; and (8) an HTTP response is returned to the Web client.

III. URIGRAPH

The model described in this paper starts from the definition of Web site structure. It is named "UriGraph" because it uses URI analysis through a directed graph to construct information about the resource identity, position and composition. For a

formal definition and a more detailed look at UriGraph, see [12].

The model is composed of three layers. The bottom layer is called the *topology layer*, defining the nodes and edges of the graph. The middle layer defines the rules for analyzing the resource identifier and is called the *request analysis layer*. The top, *response synthesis layer*, is used to define how the information is extracted from the request and incorporated in the response.

Site structure in UriGraph can be clearly presented graphically and also in special XML grammar.

A. Topology Layer

UriGraph's topology layer is defined as a directed graph constituting of the set of nodes N and a set of edges E (a binary relation over N). There are two types of nodes: *places* (collected in a set P) and *transitions* (in set T). There is one prominent place called the *root* node (r). Any two nodes may be directly connected via at most one directed edge in each direction, but:

- a node cannot be directly connected to itself (E is irreflexive);
- two places cannot be directly connected;
- a transition can have at most one outgoing edge connecting it to a place.

For each node n there is a set of destination nodes D_n, consisting of all the nodes connected to the node n through its outgoing edges.

Places are depicted as circles, general transitions as wide rectangles, and edges as arrows. The root node is marked with a symbol of a house (alluding to *home* page). An example of the topology layer is shown in Fig. 2.

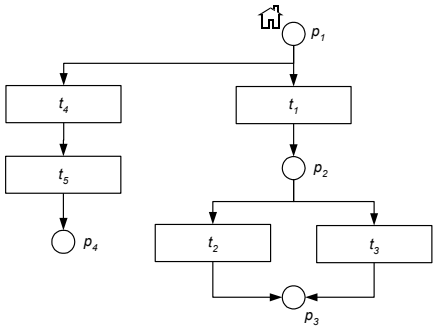

Figure 2. An example of the topology layer in UriGraph.

In XML, the structure graph is represented via a "structure" element containing node-type elements: "place" element for places, and two for transitions ("path-transition" and "query-transition" elements). Each node element has an ID and zero or more "connect-to" elements containing IDs of target nodes.

Places represent classes of resources containing a single resource or several similar resources that differ in content, but not in the way they are represented on the site. Root node is the home page and transitions mark the analysis of pieces of information.

1) Basic Topology Terms

A *walk* is defined as a sequence of nodes $(n_1, n_2, n_3, \ldots, n_z)$ in which any two consecutive nodes in the sequence are connected by an edge in the same direction: $(n_i, n_{i+1}) \in N$. A *passage* through the graph is a walk that begins with the root node ($n_1 = r$) and ends with a place ($n_z \in P$). For example, in Fig. 2 we can find a total of five unique passages: (p_1), (p_1, t_1, p_2), $(p_1, t_1, p_2, t_2, p_3)$, $(p_1, t_1, p_2, t_3, p_3)$ and (p_1, t_4, t_5, p_4).

A *circuit* in the graph is a walk which begins and ends with the same node ($n_1 = n_z$). A graph containing a circuit has an unlimited number of passages.

B. Request Analysis Layer

The UriGraph request container holds two sequences, a sequence of path segments and a sequence of query segments. The analysis starts at the root node with container filled with all the segments from the HTTP request and follows the edges through the nodes, dropping one segment at each transition. Analysis regularly finishes at some place with an empty container, thus constructing a passage in the graph. If the analysis finishes at a transition, the request is said to be *incomplete*, and the analysis is considered unsuccessful.

1) Processing Nodes

Each node in the constructed passage gets *processed*, starting at the root node. To process a node means to: (1) take out one path or query segment from the request container, and (2) find the next node, append it to the walk and continue the analysis by further processing it.

Trimming (step 1) happens only in transitions. There are two types of transitions: *path transitions* which trim path segments and *query transitions* which trim query segments. Path segments are always trimmed in order they appear in the URI; query segments can be trimmed in any order.

Graphical representation of a path transition is a parallel-ogram, side lines resembling the slashes delimiting the path segment. Analogously, a query transition has curved side lines resembling parentheses, as in the (name, value) pair. Both are shown in Fig. 3.

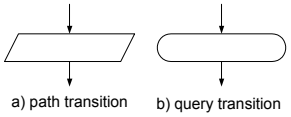

Figure 3. Graphical representations of transitions

2) Traversing Nodes

To find the next node in the passage (step 2) one has to establish which of current node's destination nodes are traversable. If there is only one traversable node, that node is taken to be the next node. If there are no traversable nodes, the request is said to be *unprocessable*. If there is more then one traversable node, the request is said to be *ambiguous* at that node. If the request is unprocessable or ambiguous, the analysis is considered unsuccessful and is halted.

Every place is defined to be open (traversable). To determine if a transition is traversable, a special logical function called a *pass* is introduced. A pass evaluates to true (open) or false (closed) depending on the part of the request being tested (either the top path segment or any query segment) and optionally on the state of the analysis (context information).

Passes are located in transitions. Traversing a transition includes *activating* the open passes in the transition. Each transition directly contains exactly one pass. A pass can be atomic or composite (composed of other passes). An example of an atomic pass is the fixed pass, testing if the segment equals a constant string value. An example of composite pass is a conjunctive pass, returning true (open) only if all its subpasses evaluate to true.

The pass of a transition is graphically represented by a text or other symbols inside the shape representing the transition. The specific representation depends on the definition of the pass. By convention, bold text represents the fixed value of the segment and italic text represents variable value of the segment (a set of permissible values is hinted by the text, e.g. "*productID*").

The XML grammar defines "pass" elements inside the transition elements. Pass is defined as a Java class whose name is listed under the "class" attribute. Other attributes and sub-elements are transferred to the class as parameters. This technique of defining the pass-specific parameters gives great flexibility but specific syntax errors remain undiscovered by the general UriGraph XML scheme. The same technique is used for describing clues and components (see section C).

3) Edge priorities

In some cases it is necessary to establish *priority relation* (denoted with ">") on the set of destination nodes for some node *n*, defining the order of testing traversability and selecting the next node in the analysis.

If the set of traversable nodes has multiple elements, the node with the highest priority is chosen. The set of nodes with the highest priority in D_n is defined as the set of all nodes for which no other node in D_n has higher priority. However, there still remains the possibility that no nodes are traversable (unprocessable request) or that two or more traversable nodes have the highest priority in the set (ambiguous request).

This paper uses the *HNL model of priority markings* when assigning priorities to edges. Each marking is a string of letters 'H' (representing high value of priority), 'N' (normal) and 'L' (low priority). We can define the set of priority letters S = {'H', 'N', 'L'} on which a priority relation ">" is defined so that 'H' > 'N' > 'L'.

A HNL priority marking is an *n*-tuple of priority letters $p = (a_1, a_2, a_3 \ldots a_n)$ – a string. Any marking can be extended by appending an arbitrary number of priority letters 'N' without changing its priority level. For any two non-equivalent markings the priority relation is determined by the difference in the priority level of the first letter at which they differ.

Note the three characteristics of this model: (1) there is a default priority, equivalent to "N"; (2) any priority marking can be extended to create higher or lower priority; and (3) there are always some priority markings whose priority is between any two different markings.

In graphical representation, the priority marking is placed near the arrow end of the edge it refers to in a visually unambiguous way (like on Fig. 7). No marking indicates default (normal) priority. In XML, the HNL priority marking is placed in the node's "connect-to" element as the value of "priority" attribute. The default value is "N" (normal priority).

C. Response Synthesis Layer

The top layer of the UriGraph model provides mechanisms for including the information retrieved from the analysis in the response – the response synthesis. The response contains the identity and the composition of the resource identified by the request.

The information in the response synthesis layer is usually not presented graphically, to avoid cluttering the picture with too much detail. If, for some reason, the clues and components need to be shown, they should be put in a callout, outside the node symbol.

1) Clues

To describe the identity of a resource, UriGraph defines *clues*, elementary pieces of information corresponding to the general attributes of the concept that the resource represents. Clues can be located in passes: each pass has its set of clues. Clues are collected while traversing transitions during the analysis. A clue is only collected if the containing pass was activated. This also applies to activated passes which were part of a composite pass. The set of collected clues at the end of synthesis represents the resource identity.

The XML grammar defines "clue" elements inside "pass" and "component" elements. They too can have a variable list of attributes, depending on the class definition.

2) Components

The composition of a resource is a set of *components*. Components are also collected during the analysis, but they reside in the places, rather than transitions. Each place in the graph has a (possibly empty) set of components.

Each component can be associated with two properties determining its inclusion into the composition: *localness* and *inheritability*. A local component is included in the response if the passage ends in the place it resides. An inheritable component is included if the place where it resides is in the passage, but not as the last node. Note that any component may carry both properties.

Components have their identity which contains the resource's identity. The difference may be in components' extra clues, which are assigned to the components themselves. So, each clue in the graph is located in clue set which is assigned to either a pass or a component in the graph.

The XML element "component" located at "place" elements is used for describing each component. Component element can also contain "clue" elements.

IV. Structure Patterns in UriGraph

To illustrate the definition and use of UriGraph and also to give some insight into Web structuring issues, this section defines and discusses some Web site structure patterns.

A. Simple Topology Patterns

These patterns are all defined in the topology layer, i.e. in terms of places, transitions and edges. Most of them are schematically shown in Fig. 4.

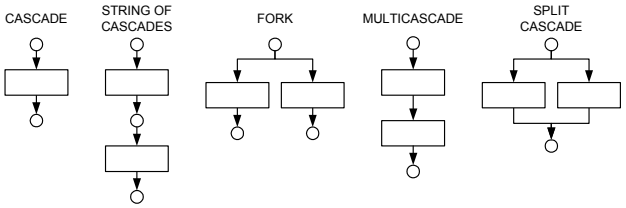

Figure 4. Some simple topology patterns.

1) Cascade

The simplest pattern of connecting nodes in UriGraph is called a *cascade* and it involves two distinct places p_1 and p_2 connected via a transition t with two edges: (p_1, t) and (t, p_2). Such cascade is marked C(p_1, t, p_2).

A simple XML description limited to the topology layer of the graph containing a single cascade:

```
<structure root-id="p1">
  <place id="p1">
    <connect-to id="t" />
  </place>
  <path-transition id="t">
    <connect-to id="p2" />
  </path-transition>
  <place id="p2" />
</structure>
```

A *string of cascades* is a tuple of two or more cascades in which the next cascade starts with the same place where the previous ended. For example, C(p_1, t_1, p_2) and C(p_2, t_2, p_3) make a string of two cascades.

A *fork* is a tuple of two or more cascades, each starting with the same place and ending in different places, such as C(p_1, t_1, p_2) and C(p_1, t_2, p_3). It is a straightforward way to implement a hierarchy between resources.

2) Multicascade

A multicascade is a variation of a cascade containing a chain of two or more transitions between the two places.

The main difference between a string of cascades and a multicascade is that in multicascade each transition in turn must be traversed in order to construct a passage. Failure to do so results in labeling the request incomplete or unprocessable and application may find it hard to recover from these conditions.

Sometimes, the functionality of a multicascade should be implemented with a single cascade containing the combined information of all cascades, eliminating the problem of creating broken URIs by hacking off segments of a good URI. Example of such a transformation is shown in Fig. 5. A branch containing images of the site has a simple structure, but no general resource "/image". A better design is shown on the right, replacing URIs in form "/image/345" to "/image-345".

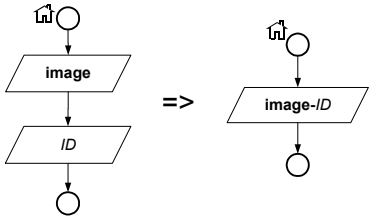

Figure 5. Replacing image multicascade with a single cascade.

An XML description of multipart pass used to combine the fixed and variable values into a single path transition on the right of Fig. 5 is shown here:

```
<pass class="MultipartPass" delimiter="-">
  <pass class="FixedPass" value="image"/>
  <pass class="ImageIdPass">
    <clue class="ImageIdClue"/>
  </pass>
</pass>
```

3) Split Cascade

A split cascade is a set of cascades sharing both places, e.g. C(p_1, t_1, p_2) and C(p_1, t_2, p_2).

A split cascade with several fixed value transitions can replace a single transition with variable value. This is a simple example of an expanded site structure being defined either in the site structure definition or in the site content (see also section B.3)). Example in Fig. 6 illustrates the alternatives: one can either code each value as a separate transition in the split, or make one pass examine the allowable values.

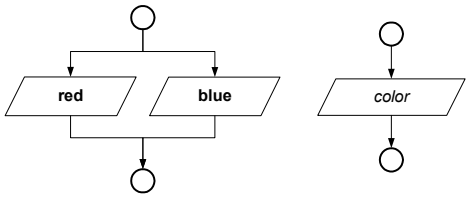

Figure 6. Split cascade and cascade with variable value transition.

4) Loop

Since no node can be connected to itself, the smallest circuit in the graph is a simple pattern involving a node double connected to a transition, i.e. E contains both (n, t) and (t, n) pairs. It is called a *loop*.

A loop is often used with the query transition. If a query segment is optional, the analysis should proceed from the same place with or without that segment. Query transition has higher priority than the path transitions between the destination nodes because it needs to be processed earlier. Fig. 7 shows an example of loop use – a clue modeled by a query segment "help" added to the URI to include detailed explanations of options on some Web page (on-screen help).

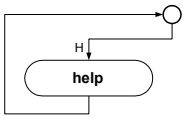

Figure 7. A simple loop example.

B. Simple Analysis Patterns

Analysis patterns in UriGraph focus on the implementations of various useful types of passes and their constructions.

1) Multilingualism

An important aspect of Web site design is multilingualism, the property of the site to provide parallel content in multiple languages, while keeping the structure and any other part of site definition that does not depend on the chosen language defined only once. Sites with multilingual content only rarely have multilingual URIs. However, the language in which the URI is written is crucial to its meaningfulness, readability, memorizability, and pronounceability.

UriGraph's implementation of multilingual URIs consists of a composite *language pass* and a built-in *languages clue*. A languages clue containing the set of all the languages that the site uses is added to the response container at the start of analysis. Whenever traversing a transition depends on the language of the segment, a language pass is used to filter through only the language(s) that apply. If the current languages clue does not contain any of the specified languages, that pass evaluates to false (closed).

Take for example graph on Fig. 8. It consists of a string of cascades with multilingual passes in Croatian and English. Path "/city/Vienna" is allowed, but "/city/Bec" is not. The reason is that first transition is traversed with segment "city" and the languages clue is reduced to contain only the English language. The second transition is not traversable because there is no city called "Bec" in English in the content of the site, although there is one in Croatian.

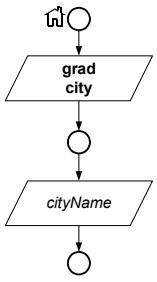

Figure 8. An example of multilingual passes.

As you can see from the Fig. 8, a multilingual clue is depicted as text in multiple lines, in the globally defined language order. Language pass is implemented in the class called "LangPass". XML element defines subelements called "translation". Each translation element contains a language-specific pass. Description of the pass in the first transition is shown here:

```
<pass class="LangPass">
  <translation lang="HR">
    <pass class="FixedPass" value="grad"/>
  </translation>
  <translation lang="EN">
    <pass class="FixedPass" value="city"/>
  </translation>
</pass>
```

2) Limiting the Number of Traversals

Each circuit in the graph provides a way to model an unlimited number of passages through the graph. The number of passages may be limited using a pass called the *traverse limit pass*. Each such pass has a limit (a positive integer) and an assigned counter. Each time a transition is processed and traversed, a counter is increased by one. When the counter reaches the defined limit, it closes.

Traversal limit pass is depicted with a formula $T_c \leq L$, where c is an integer identifying the counter used and L is the limit. It is implemented as the "TraverseLimitPass" class with XML attributes "counter" and "limit".

Notice that several traverse limit passes may share the same counter. When the counter is increased in one pass, another may close. This can be useful in cases when several transitions are alternatives. Graph in Fig. 9 shows an example of two alternative query segments that serve as switches for the entire site. The "content" transition includes a clue that signals the Web server not to transform the content using a stylesheet, skipping steps 6 and 7 from the section II.C. The other, "printable" transition signals the use of a special stylesheet for creating "printer-friendly" page renderings. These two switches are obviously incompatible and therefore modeled as alternatives.

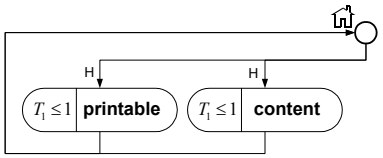

Figure 9. An example of alternative query segments.

The XML definition of the "printable" query transition is shown below. The "ControlClue" clue communicates to the Web server not to transform the content using the resource look (to skip steps 6 and 7 in Fig. 1).

```
<query-transition id="printable">
  <connect-to id="root" />
  <pass class="TraverseLimitPass"
        counter="printable-and-content"
        limit="1" />
  <pass class="FixedPass" value="printable">
    <clue class="ControlClue" key="no-transform" />
  </pass>
</query-transition>
```

3) Semi-structured Site

Sometimes, a function of a pass depending on content data can replace a part of a graph, as in section A.3). Therefore, that part of the Web site structure is more defined by its content than by the fixed structure definition in UriGraph. This phenomenon could be called a *semi-structured site*, and the content of such a site is often itself semi-structured data.

As an extreme form of structure located in the content, Fig. 10 shows a loop with a path transition and a loop with a query transition. Transitions contain complex passes that determine the URI interface from the content. Clues in the passes carry over the string values of the segments. The place has a single

component that creates the whole composition for each resource based on the values of the segments.

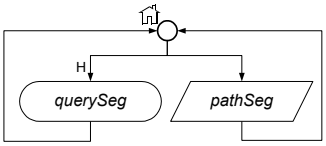

Figure 10. An extreme example

4) Parallel Walks

Parallel walks through the graph are the ones that have the same starting and finishing nodes, but different nodes in-between.

For example, the original site of a daily newspaper had a single cascade denoting the date of the issue, with a pass allowing dates formatted as "*mm-dd-yyyy*". Later was decided to provide a string of cascades "*year*", "*month*", and "*day*" to group the issues by year (like "/2002") and month ("/2002/11"). The expansion was simple: a string of three cascades with special passes as shown on Fig. 11.

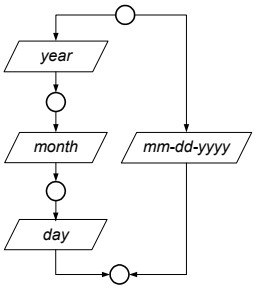

Figure 11. Two parallel walks

Note that the old URI "/11-01-2002" still works, and also its equivalent "/2002/11/1". There is no canonical URI and there is no redirection.

5) Shortcuts

It is essential that URIs exposed through the "non-hyper" media are kept short [11]. On the other hand, well-structured URIs are sometimes relatively long, so a "shortcut URI" may be useful.

A typical example is a shortcut to a popular product. Some company's ice cream IceKing is located at "/iceking", which is a mere shortcut to a more verbose "/product/icecream/iceking". A shorter URI may be publicized (e.g. on TV commercials) and then those URIs would be redirected to their respective "full" identifiers.

UriGraph implementation may consist of a single path cascade attached to the root node, with a pass that reads a list of shortcuts from the content. Alternatively, the cascade may be replaced by a split containing the shortcut segment values in the graph, as in section A.3). Each transition in the split carries a clue with the "full" identifier, and the place contains a component issuing a "301 Moved Permanently" response status code.

C. Web Transactions

Altering the data in Web site content is referred here as a *Web transaction*. The site structure needed to facilitate a Web transaction should in general conform to this pattern:

- a Web page with HTML form which can generate an HTTP POST request;
- an operational resource which performs the transition but doesn't return an HTTP entity, but a "303 See Other" response code redirecting to a post-transaction page;
- a post-transaction Web page displaying a short description of session status on top of the page followed by "normal" page contents.

A detailed structure for displaying and processing messages from the discussion group is shown on Fig. 12. Resource 1 is representing all messages, resource 4 a specific message (e.g. "12"), and resource 5 a selection of messages, either a range (e.g. "?from=1&to=9") or a enumeration (e.g. "?id=2&id=5&id=3"). Resources 2 and 6 contain a form component used to enter or alter the message data. Resources 3, 7, and 8 are operational.

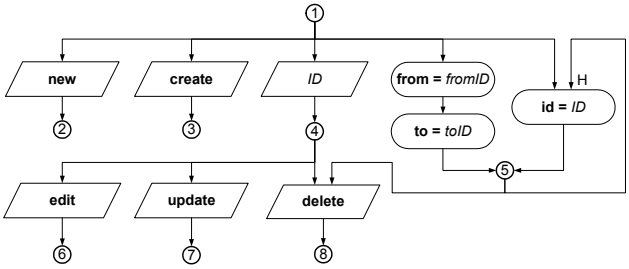

Figure 12. Structure for displaying and processing messages.

Most operations (creating, updating and deleting a specific message), as well as displaying the message data, are done by a "message" component located at places 3 (local) and 4 (local and inheritable). A "message selection" component is located at place 5 (local and inheritable). It displays a list of messages, and can also delete several messages at once, using the same "delete" clue. The "message form" component is located at places 2 and 6, and displays the empty (for new messages) or pre-filled form (for editing the existing message).

Session info component indicating the status of the last transaction is inheritable and located in the resource 1, so every resource will have it. Often this component will display the same information regardless of the resource it is a part of, because it doesn't have to depend on any of the resource identity.

An example of a Web transaction is the creation of a new message on a discussion board. Data is entered using a form on a Web page ("new"), an operational resource ("create") receives the data and redirects back to the form if the data is incorrect. The form will now display additional session information instructing the user how to correct the input data. If the data is correct, a new message is created and the browser is redirected to the resource representing that message (e.g.

"33"). The page will display "You successfully created a new message shown here" and the message itself.

V. Related Work

The term "Web site structure" has often very different meaning than the one used in this paper. Sometimes it is synonymous to content data structure (such as entity-relationship diagram) or the site navigation scheme. There are numerous models and methodologies directly or indirectly designed for the modeling Web site structure in a broad sense, raging from the ones used in commercial applications to experimental and purely academic. Some inspiring but rather unrelated approaches to UriGraph include OOHDM [10] and Strudel [4].

The classic model for structuring Web sites is used in all of today's popular Web servers like Apache and Microsoft Internet Information Server. It is mainly based on the file system and partially on simple configuration data (e.g. redirection URIs). Resources are implemented as files, either passive (returning the content of the file) or active (returning the output of the program stored in the file). A similar approach may be found in Oracle's Internet Application Server (*i*AS) [9], where the underlying system is the relational database and the active resources are stored procedures. These two models depend on the underlying system (file system or database server) and project its identifiers (directory and file names or procedure names).

There are some approaches that concentrate on resource position modeling. The Structured Graph Format (SGF) [6] integrates the strict hierarchical (tree) organization with the free network model. A similar approach can be seen in the SiteBrain model [13].

The resource composition is efficiently modeled with WebML [3]. The specification of a site in WebML consists of four orthogonal perspectives that include the structural model (expressing the structure of the data content of the site) and the hypertext model. The hypertext model is further divided into the composition model that provides a number of built-in components (called "units") and the navigation model that expresses the links between pages and components. Especially important are the "contextual links", transferring identity information from one component to the other.

A different composition model is included in Oracle Portal (a part of *i*AS) [8]. Its components are called portlets, and they are modeled as programming components conforming to a specified API.

Most models mentioned here are incomplete. They usually limit themselves to Web pages and leave other types of presenting and all operational resources to be handled by the classic model. Many are also detached from the Web server leading to discrepancies between the design and deployment. Most models also fail to cover some important aspects of Web site design such as Web transactions or multilingualism. However, those and other issues [5] are currently being identified and addressed.

VI. Conclusion and Future Work

It may be noted that the above models model only one or two aspects of Web site structure, and cannot provide the complete coverage. This is mainly because those models are effectively forced on, and not natural to the Web. Most of them do not appreciate the importance of URI design, nor resource identity.

UriGraph can be used to describe the structure of any Web site, but it is especially intended to be used as a blueprint for larger, even enterprise-sized Web applications. It is a tool for software engineers and some of its features can only be exploited through programming.

Future work on UriGraph may include a better deployment platform (a UriGraph-enabled Web server) for further testing and experimenting. It would be useful to have a graphical tool for designing and maintaining structure graphs, especially a feature for securing the backwards-compatibility of structure (identifier persistence [1]). An adequate position model needs to be developed, since the current resource position may only be deduced indirectly, via resource identity.

VII. Acknowledgments

Thanks to my coworkers at the University of Zagreb which worked with me on several projects leading to UriGraph, especially Maja Matijasevic and Marko Topolnik for detailed reviews and suggestions.

VIII. References

[1] Berners-Lee, T. Universal Resource Identifiers – Axioms of Web Architecture. W3C Design Issues, December 1996, http://www.w3.org/DesignIssues/Axioms.html

[2] Berners-Lee, T., Fielding, R., and Masinter, L. Uniform Resource Identifiers (URI): Generic Syntax and Semantics. RFC 2396, August 1998, http://www.ietf.org/rfc/rfc2396.txt

[3] Ceri, S., Fraternali, P., and Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites, 9th International World Wide Web Conference, May 2000, Amsterdam, The Netherlands, http://www9.org/w9cdrom/177/177.html

[4] Fernandez. M. F. *et al*. Overview of Strudel - A Web-Site Management System. Networking and Information Systems 1(1), pp. 115-140, 1998.

[5] Gu, A., Henderson-Sellers, B., Lowe, D. Web Modelling Languages: the gap between requirements and current exemplars. AusWeb02, July 2002, http://ausweb.scu.edu.au/aw02/papers/refereed/lowe/paper.html

[6] Liechti, O., Sifer, M. J., and Ichikawaielsen, T. Structured graph format: XML metadata for describing Web site structure. 7th International World Wide Web Conference, August 1998, Brisbane, Australia, http://www7.scu.edu.au/programme/fullpapers/1853/com1853.htm

[7] Nielsen, J. Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Indianapolis, 2000, http://useit.com/jakob/webusability/

[8] Oracle Portal Center, http://portalstudio.oracle.com

[9] Oracle9i Application Server, http://www.oracle.com/ip/deploy/ias

[10] Schwabe, D., Rossi, G. The Object-Oriented Hypermedia Design Model. Communication of ACM, 38(8), 1995, pp. 45-46.

[11] Simic, H. Application of UriGraph to Uniform Resource Identifier Design. CARNet User Conference 2002 Proceedings, Zagreb, Croatia, September 2002, http://www.tel.fer.hr/users/hsimic/cuc2002

[12] Simic, H. Modelling of WWW site structure. Master's Thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, May 2002 (In Croatian), http://www.tel.fer.hr/users/hsimic/magisterij.pdf

[13] The Brain, http://www.thebrain.com