SYSTEM AND METHOD FOR MANAGING DYNAMIC CONTENT ASSEMBLY

Inventors: John N. Dreystadt, Pickney, MI (US);
Timothy P. Allen, Saline, MI (US);
John A. Koenig, Ann Arbor, MI (US);
Curt M. Malouin, Northville, MI (US);
Ying-Che Fang, Ann Arbor, MI (US);
Andrew E. Dobrowski, Saline, MI (US)

Correspondence Address:
DORSEY & WHITNEY LLP
INTELLECTUAL PROPERTY DEPARTMENT
50 SOUTH SIXTH STREET
MINNEAPOLIS, MN 55402-1498 (US)

Appl. No.: 10/839,109
Filed: May 5, 2004

Abstract

A dynamic content assembly system, including an application and underlying database, with methods to support the creation, transformation and management of relationship information between resources and to enable dynamic assembly of content based on these relationships.
Dynamic Content Process

Start → 10 → Parse Document

12 → Find Link → Resolve Link → 14 → Traversal → 16 → Embed → 20 → Assembly

18 → Insert Resolved Link → 22 → Insert Content

24 → Run Query and Insert Content

End

FIG. 1
FIG. 2
FIG. 3
FIG. 4
FIG. 6

User Interface Code

DCAM Client Extensions

Scripting API

Application API

Service Interface

SOAP Transport Layer
FIG. 8
FIG. 9
FIG. 10
Apply Profiles 250

Select the Profiles to apply to the resource pair by choosing from the options below:

- Operating System
  - Windows
    - Windows XP
    - Windows 2000
    - Windows Server 2003
  - Unix
- User Level
- Output Type
- Security Level

Select Target(s) From:

- Apply Individual Profiles
- Apply Profile Group

FIG. 12
Update Element Profile

Select the Profiles to apply to the Title element by choosing from the options below.

☐ ☐ Advance XP Users

Mode:

☐ Apply Individual Profiles ☐ Apply Profile Group

[Buttons: OK, Cancel, Help]

FIG. 13
FIG. 15
Profile Filter

Select the applicable Profiles by checking the check boxes:

- **Document**
  - TM1456
  - TM1457
  - TM1458
  - TM1459
  - TM1460

- **Output Media**
  - Screen
  - PDF
  - Web
  - HTML
  - HTMLHelp
  - Wireless
  - PDA

- **Language**

Buttons:
- **OK**
- **Cancel**
- **Filter Group**
- **Help**

**FIG. 16**
Profile filter Group

Name: OtherNamedProfile

FIG. 17
Administrator UI Client Code Transport Server Code

Document registered Flag set

no

yes

Register all resources in the document

Activities: Add Resource Pair for Existing Link

Final State

Set "autoregister" parameter

Autoregister flag: true

yes

Set "autoregister" parameter

Create Link definitions in repository for every link type ethernet

no

Encapsulate link markup within the "elink" tag

Determine Link names through the DCF configuration file

Server Code

Transport

Client Code

Activities: Add Resource Pair for Existing Link

Commit all links and resource pairs to repository

Patch create links

Create all links and resource pairs in the database. Return IDs for created links

FIG. 19
Use Case Model: Create Link Definition

for each resource pair

Name Link

Reference Starting Resources
Reference Ending Resource
Set Profiles
Set Other Attribute Values

Record Link in Repository

Create Link Definition Complete

FIG. 20
Create ID/IDREF Link

Link Internally

Capture Resource Information

Create Link Definition

Insert Link Reference

Create ID/IDREF Link Completed

FIG. 21
FIG. 22
Insert Link Reference

Register Starting Resource with Link

Insert Link Reference in Object

Insert Link Reference Complete

FIG. 24
Create Outbound Link

Browse/Select Ending Resource

[Ignore 1]

Create Link Definition

Insert Link Reference

Create Outbound Link Complete

FIG. 25
FIG. 27
Insert Link Repository Reference

Browse Link Repository

Select Link

Insert Link Reference

Activities: Insert Link Reference

Insert Link Repository Reference Complete

FIG. 29
DCA Resolution Filter

SAX events from include content link

SAX events from object processing

DCA Resolver Filter

Convert reference link to markup SAX events

SAX events from link processing

FIG. 31
SYSTEM AND METHOD FOR MANAGING DYNAMIC CONTENT ASSEMBLY

FIELD OF THE INVENTION

[0001] This invention relates generally to a system for creating, linking, and assembling electronic content. More specifically, it relates to a system and method for dynamically assembling content from different sources.

BACKGROUND OF THE INVENTION

[0002] In today's content publishing environment, content may be generated and edited using a variety of editors, such as Microsoft Word, Web editors (e.g., Arbortext's Contributor), and Extensible Markup Language (XML) editors (e.g., Arbortext's Epic Editor). Similarly, this content may be published to a variety of output media formats, including print, Portable Document Format (PDF), various forms of Hypertext Markup Language (HTML), Wireless Markup Language (WML), and PostScript. Content may also be published to compiled formats such as HTML-Help, MS Reader, formats for personal digital assistants (PDAs), and formats for mobile phones.

[0003] Assembling documents from various formats can be challenging. The documents must be assembled from many different pieces with many different cross-document links. While the task of storing documents and their components in a repository is currently being handled by multiple vendors, there is a need to automate the dynamic assembly of document components and their related links to other document components. This assembly is currently being done in a laborious way requiring extensive special case programming.

[0004] The complexity of dynamic content assembly across multiple media formats, audiences, compound documents, and versions can be costly when done via manual processes such as creating multiple documents, cutting and pasting content, or completely recreating information with additional review required for all newly created information.

[0005] Accordingly, there is a need in the art for a system or method to manage the dynamic creation, linking, and assembly of content.

BRIEF SUMMARY OF THE INVENTION

[0006] The present invention is designed to support the defining, creating, modifying, storing, reusing, validating, resolving, and exchanging multiple link types. Each link or link collection may have multiple audiences defined. Each link or link collection may have multiple media-appropriate, output-resolution filters. Each link or link collection may be validated for the given contexts of media format, audience, compound document usage, and document or sub-document component version.

[0007] While multiple embodiments are disclosed, still other embodiments of the present invention will become apparent to those skilled in the art from the following detailed description. As will be apparent, the invention is capable of modifications in various obvious aspects, all without departing from the spirit and scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not restrictive.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] FIG. 1 illustrates an overview of the DCAM steps of the Arbortext DCAM system in accordance with one embodiment of the present invention.

[0009] FIG. 2 illustrates a first embodiment of the architecture of a DCAM system.

[0010] FIG. 3 illustrates a second embodiment of the architecture of a DCAM system.

[0011] FIG. 4 illustrates a diagram of a server diagram in accordance with one embodiment of the present invention.

[0012] FIG. 5 illustrates schema implemented by a database in accordance with one embodiment of the present invention.

[0013] FIG. 6 illustrates a diagram of the client extensions in accordance with one embodiment of the present invention.

[0014] FIG. 7 illustrates a screen shot of the Link to Current Document version of an Insert Link dialog in accordance with one embodiment of the present invention.

[0015] FIG. 8 illustrates a screen shot of the Link to Repository version of an Insert Link dialog in accordance with one embodiment of the present invention.

[0016] FIG. 9 illustrates a screen shot of the Link to Web Object version of an Insert Link dialog in accordance with one embodiment of the present invention.

[0017] FIG. 10 illustrates a screen shot of an Modify Link Properties dialog in accordance with one embodiment of the present invention.

[0018] FIG. 11 illustrates a screen shot of a Link Explorer dialog in accordance with one embodiment of the present invention.

[0019] FIG. 12 illustrates a screen shot of an Apply Profiles dialog in accordance with one embodiment of the present invention.

[0020] FIG. 13 illustrates a screen shot of an Apply Profile Group dialog in accordance with one embodiment of the present invention.

[0021] FIG. 14 illustrates a screen shot of a Search dialog in accordance with one embodiment of the present invention.

[0022] FIG. 15 illustrates a screen shot of an Export Linkbase dialog in accordance with one embodiment of the present invention.

[0023] FIG. 16 illustrates a screen shot of a Profile Filter dialog in accordance with one embodiment of the present invention.

[0024] FIG. 17 illustrates a screen shot of a Profile Filter Group dialog in accordance with one embodiment of the present invention.

[0025] FIG. 18 illustrates a diagram of creating a link definition in accordance with one embodiment of the present invention.

[0026] FIG. 19 illustrates a diagram of creating a link definition in accordance with one embodiment of the present invention.
FIG. 20 illustrates a diagram of creating a link definition in accordance with one embodiment of the present invention.

FIG. 21 illustrates a diagram of creating an ID/IDREF link in accordance with one embodiment of the present invention.

FIG. 22 illustrates a diagram of linking internally in accordance with one embodiment of the present invention.

FIG. 23 illustrates a diagram of IDing an element in accordance with one embodiment of the present invention.

FIG. 24 illustrates a diagram of inserting a link reference in accordance with one embodiment of the present invention.

FIG. 25 illustrates a diagram of creating an outbound link in accordance with one embodiment of the present invention.

FIG. 26 illustrates a diagram of browsing/selecting a starting resource in accordance with one embodiment of the present invention.

FIG. 27 illustrates a diagram of browsing/selecting an ending resource in accordance with one embodiment of the present invention.

FIG. 28 illustrates a diagram of inserting a link repository reference in accordance with one embodiment of the present invention.

FIG. 29 illustrates a diagram of inserting a link repository reference in accordance with one embodiment of the present invention.

FIG. 30 illustrates a diagram of composing with a DCAM system in accordance with one embodiment of the present invention.

FIG. 31 illustrates the functioning of a link resolution filter in accordance with one embodiment of the present invention.

FIG. 32 illustrates the configuration components for linking, data merging, and profiling in accordance with one embodiment of the present invention.

FIG. 33 illustrates a diagram of the configuration components in accordance with one embodiment of the present invention.

FIG. 34 illustrates the overall flow of inserting data in a document using data merge in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

The present invention is capable of dynamically managing content linking and assembly. Dynamic Content Assembly Management, or "DCAM," is generally the process of supporting and enabling the creation, transformation and management of relationship information between content resources and dynamic assembly of content based on these relationships. Thus, generally, a DCAM system provides a means for defining, creating, modifying, linking, reusing, validating, resolving, and assembling dynamic content using an efficient and automated process.

Dynamic Content Assembly (DCA) can be divided into many functional areas such as: embedding traversal points such as hyperlinks; embedding content from one resource into another resource; transforming content to a representation in the desired data format; filtering content based on meta-information such as the target audience for a specific portion of the content; and transforming the content to a representation with the desired style.

The DCAM System (or Link Management System) of the present invention is an application and underlying database, which supports the creation, transformation and management of relationship information between resources. Links can serve multiple purposes such as:

- indicating a point of traversal to a related piece of content, for example, Hyperlink in a web document.
- embedding textual content of one resource at a particular location in another resource, for example, a company biography paragraph may be stored separately and used by reference in multiple pages.
- embedding graphical content of one resource at a particular location in another resource, for example, a logo or graphic in a Hypertext Markup Language (HTML) page which is stored separately from the HTML page and could be used by reference in multiple pages.
- embedding content from an external application, for example, including product specifications in a product data management system in a marketing data sheet about that product.
- indicating a related collection of information by associating multiple pieces of related content, for example, a retailer providing a related collection to a purchaser of a certain product.
- indicating the audience for the content, which allows creation of a single source of content that supports multiple views, for example, the views could be driven by the person reading the document deciding between the expert view and the beginner’s view, by the parent document needing the repair description rather than the build description, or by the publishing process needing the Spanish version rather than the English version.

The present invention enables dynamic assembly of content based on these relationships or links.

The term link, as used herein, is meant generally and includes not only hyperlinks but also "include", "filter"; or any other means for embedding text or content within a document. Thus, the concept of link management in the DCAM system supports all types of dynamic content assembly management.

Generally, Extensible Markup Language (XML) provides a useful, human and machine readable, standardized syntax for representing content, metadata, links, queries, transforms, and configuration information, which permits simplified processing of dynamic content. Document content (prose) can be created as XML natively, or transformed from proprietary applications to XML. Data from a variety of software applications, including directly from many databases is accessible as XML. W3C Standards such
as XML Linking Language (XLink), XML Path Language (XPath), and Extensible Stylesheet Language (XSL) can be leveraged to enable application programmers and users with a standard, application interchangeable syntax.

[0053] In today's content publishing environment, content may need to be published to print, Portable Document Format (PDF), multiple forms of HTML including different web sites, locations, and languages, compiled formats such as HTML- Help and MS Reader, formats for palm devices, formats for cell phones, and/or exchanged with partners and customers in a source format such as XML. Each media type has its own functional capabilities for representing dynamic content. For example a print document represents a navigation link such as "See also "The Definitive Guide," chapter 13, paragraph 12" while an HTML file might have a hyper-link embedded in the phrase "Get More Information" to a specific paragraph in another HTML file.

[0054] For each media format the "link" needs to be represented natively for that format. For example, in HTML a link could be defined by a simple anchor tag such as: <a href="http://www.arbortext.com">. However, more complex links may require dynamic HTML using a scripting language such as JavaScript. For printed documents, the same cross reference may appear as "Contact Arbortext, Inc." or alternatively "See Arbortext, Incorporated's web site at www.arbortext.com"

[0055] As links become more complex—one to many, many to many, rings, etc.—the ability to create, verify, and resolve links for multiple publish media becomes exceedingly difficult to do manually. A DCAM system enables resolving links appropriately for each media format.

[0056] Several types of links may be managed using the DCAM system of the present invention: simple links (also called outbound or navigation links), inbound links, graphic links, extended links, taxonomy links, object to object links, links for use with other applications, links wherein the application decides the traversal means, third party links, related information links, and media object links.

[0057] A simple link associates exactly two resources, one local and one remote, with an arc going from the former to the latter. Thus, a simple link is always an outbound link (going away from the linking element). An outbound link is used when an explicit location within an object references outside the object. With this type of link, the profiling of the link determines the resolution. Conversely, if the arc of the link were to start at a remote resource and end at a local resource, the link is inbound. An inbound link relationship is maintained entirely within link manager. A graphic link is used when an explicit location within an object references to different graphics based on profiling. With this type of link, the profiling of the link determines the resolution. An extended link associates an arbitrary number of resources. The participating resources may be any combination of remote and local. A taxonomy link associates information with a subject taxonomy. Taxonomy links are used to associate topics with subjects and categories of subjects. This allows improved searching for appropriate topics both for specific subjects such as cardiac infarction and general subjects such as heart disease. A third party link is a link wherein neither the starting resource nor the ending resource is local. Related information links are used when an object does not explicitly reference another object but a relationship exists. This type of link is a third party arc entirely maintained in the link repository. The links provided are in relation to a given resource. Media object links are frequently context sensitive links. Resolution may be document dependent, output dependent or language dependent.

[0058] XML further simplifies managing a document as a hierarchical collection of sub-document level components. These collections are often referred to as "compound documents." For example, a cookbook is made of 20 recipe chapters, but each recipe chapter is stored individually. A given recipe chapter may be reused in numerous books. Creating and managing a link in a sub-document component which may be reused in multiple contexts is difficult. If these documents and components are managed in a content management system, they may also be versioned. The present invention provides a DCAM system that may be used to enable the managing and resolving of links for different document contexts and different document versions.

[0059] Additionally, the DCAM system of the present invention may be used to publish alternate views of a document targeting different audiences. For example, a catalog may contain different prices for customers, partners, and employees. The catalog may be different for North America, Europe, and Asia, and may also differ when in print versus web format. Making links work in all of these contexts is often handled by manual processes and redundant copies of the document content. The present invention enables the assembly of content dynamically to suit the needs of a given audience.

[0060] Documents may be a combination of textual content created by authors and database content derived from queries performed against a database application. The invention, in providing a DCAM system, provides mechanisms for managing the database derived content the same way that the textual content is managed. This allows the database derived content to be profiled for different audiences and to be published appropriately for different output media formats.

[0061] The present invention supports reuse, repurposing and dynamic assembly. This includes multichannel publishing such as output sensitive linking, information reuse such as context sensitive linking, dynamic content such as inclusion and data merge, and content supply chain such as web services. Performing link management using the present invention has several benefits. A single link may be managed with different output resolutions—thus providing simplified support for multichannel publishing. Further, a single link may be managed with different resolutions based on different contexts. This enables template-based authoring and reusing rather than duplicating business data with context sensitive links. Linked content may be dynamically included based on context and can include or reference the content. A web services interface supports information exchange across the enterprise and a Simple Object Access Protocol (SOAP) Application Program Interface (API) (discussed more fully below) is provided for manipulating a link repository.

[0062] The link manager manages component or linking relationships internal to an object, across objects, across versions, across publishing cycles, and across supply chains.

[0063] FIG. 1 illustrates an overview of the DCA steps of one embodiment of a DCAM (or link management) system.
In order to run a dynamic content process, the DCAM system must manage and interpret configuration information for each process. As seen in FIG. 1, the dynamic content process is initiated at block 10 with a request to parse a document. The request may be initiated by determining an access point, such as an end point of a relationship arc (described above). The end point of the relationship arc may be either the start or the end of the relationship arc. As the document is parsed, links are found at block 12. Additional data regarding the link is fetched at block 14 during link resolution. For example, link profiling (explained more fully below) may be performed. The link type is checked at block 16. If the link is a navigation link, updated markup is inserted at block 18 based on the additional data. If the link is not a navigation link, determination is made, at block 20, of whether the link is a simple embedding of content or a graphic. If the link is a simple embedding of content or a graphic, the actual embed operation happens at block 22. If the link is not a simple embedding of content or a graphic, the link type is an assembly operation at block 24 and the actual complex assembly operation happens at block 26. Thus, the contents of the link is assembled dynamically based on rules. Once the assembly operation is performed, the content of the link is available from the access point. The DCAM system of the present invention also permits incorporating references to external data sources using data merge (described more fully below) wherein the references are then periodically resolved.

As explained above, managing the dynamic assembly of content involves determining an access point or end point of a relationship arc or link. The arc leads from the access point to contents for dynamic assembly. Thus, for example, a hyperlink leads to a url. The content to which the arc leads may be verified based upon rules. Once the appropriate relationship is verified, the content is made available from the access point.

FIG. 2 illustrates the architecture between an example DCAM client application and a DCAM server application. Using the architecture shown, communication is enabled between the DCAM client application and the DCAM server application. The example DCAM client application shown is Epic Editor 40. Of course, alternate DCAM client applications may be used. A set of DCAM Client Extensions 30 provide additional user interfaces and programming logic to the client application. A DCAM Repository 48 is shown including an Epic E-Content Engine 64, a Relational Database 65, DCAM Server Extensions 63, and a SOAP Transport Layer 34. Operations requiring interactions with the DCAM Repository 48 first call into the client SOAP Transport Layer 32. This communicates with the corresponding SOAP Transport Layer 34 inside the DCAM Repository 48. The interactions are then processed by the DCAM Server Extensions 63 which extend the Epic E-content Engine 64. These same extensions 63 use a Relational Database 65 for long-term storage and retrieval of information.

A link management application generally performs certain functions. These functions typically require the application to include means to create, modify, remove, store, configure the behavior of, validate, resolve and exchange link information with other applications. The architecture of a DCAM System in accordance with a first embodiment of the invention is shown in FIG. 2. A diagram of an alternate architecture of a DCAM System of the present invention is shown in FIG. 3. As shown, to effectively perform the functions of a link management application, the DCAM System includes: an Epic Editor Authoring User Interface 40, a DCA Repository Authoring User Interface 42, a DCA Configuration 44, a Configuration Interface 46, a DCA Repository Authoring User Interface 50, a DCA Repository SDK 52, a Linkbase 54, a DCA Resolver 56, a DCA Resolver SDK 58, and a DCA Validator 60. Following is a description of each of these components in the embodiment shown. These descriptions are intended to be illustrative only, not limiting.

The Epic Editor Authoring User Interface 40 is a user interface used within the Epic Editor environment. It is an interface for creating, modifying, removing and managing linking information. Generally, operations performed by an author within the editing environment are exposed through this interface.

The DCA Repository Authoring User Interface 42 is a user interface for creating, modifying, removing and managing linking information directly in the link repository. Generally, operations performed by an author within this environment are exposed through this interface.

The DCA Configuration 44 is the configuration information for the DCA system based on a specific data model. The focus of this application is configuration rather than customization. To this end, anything which can be abstracted to this configuration generally should be.

The Configuration Interface 46 is a simple user interface for inputting and modifying configuration information for the DCA system based on a data model.

The DCA Repository 48 is a database application for storage, search and retrieval of the link information. The DCA Repository Administrator User Interface 50 is the user interface for performing IT administration functions on the DCA Repository 48. The DCA Repository SDK 52 is documented Application Program Interface (API) for programmatically communicating with the DCA Repository 48.

The Linkbase 54 is an XLink compliant linkbase XML document. These documents may be used to export portions of the DCA Repository 48 or to import modifications into the DCA Repository 48.

The DCA Resolver 56 is a filter in the content pipeline for resolving linking information. The DCA Resolver SDK 58 is documented API of the Java methods (or other) used in the DCA Resolver 56.

The DCA Validator 60 is a validation program to verify the validity of link information.

The DCA server is a system designed to manage linking information externally from documents, while still maintaining actual links as if they solely exist within documents. The DCA server provides late-binding linking, in a variety of formats. Using this, a link may be formatted into a variety of link syntaxes at publish time. The DCA system comprises three major components: a server tier, a transport transport tier, and a client tier.

The design of the DCA system is flexible, enabling bindings to be made to any database. In a particular embodiment the overall language used in the design of the DCA
system is Java. The system may be built for an Oracle Platform. Alternately, any other relational database platform may be used.

[0077] Explained differently, the DCAM System includes a Server Tier, a Transport Tier, and a Client Tier.

[0078] The server tier of the DCAM System comprises a database bound to a server application. The server application contains logic necessary to interact with the database, and exposes a full API. All database access is performed through the server API. The server component is central to the DCA System. The server component is responsible for storing and managing all linking information within the system. The server component comprises a database binding, classes representing database objects, and an API that connects the components, in addition to providing a simple way to interface with the server.

[0079] A server diagram is provided at FIG. 4. As shown, XML text 61 is input into a Composition Pipeline 62. The server component may be customized by writing custom applications that use the Server Extensions 63 to add functionality. Additionally, the Server Extensions 63 provides multiple classes that are programmatic representations of database objects and their corresponding collection classes. The Server Extensions 63 may be tied to the Epic E-content Engine 64 and Relational Database 65. Information can then be output as print output 66, web output 67, or other outputs 69.

[0080] In one embodiment, the server component is built on top of a SQL compliant database. FIG. 5 illustrates schema implemented by the database to properly store the linking information. The database schema contains the tables and relationships necessary to store and manage links, resources, resource pairs, folders, and their associated properties and metadata.

[0081] The links table 70, contains all links in the system. Each link includes one or more titles (as it has multilingual support) and may have metadata properties associated with it. The link_metadata table 72 contains a list of metadata properties related to the links. The link_title table 74 contains one or more names for each link. Again, links may have multiple titles defined, each in a different language. The link_folders table 76 contains a hierarchical listing of folders used to categorize and classify links. The link_folder_metadata table 78 contains metadata properties that are related to link folders. The link_folder_title table 80 contains titles that are directly related to each link folder. Multiple titles may be specified, each in a different language.

[0082] The resources table 82 contains a listing of resources within a repository. Each addressable object (documents, element, etc.) has a resource definition specified in this table. Resources may be addressed by URL and when URIs are stored, they are broken down into their component atoms. All resources that are considered top-level documents have an is Document flag set. The resource_metadata table 84 contains metadata properties that are related to the resources. The resource_title table 86 contains titles that are directly related to each resource. Multiple titles may be specified, each in a different language. The resource_folders table 88 contains metadata properties that are related to resource folders. The resource_folder_title table 90 contains titles that are directly related to each resource folder. Multiple titles may be specified, each in a different language. The resource_pairs table 92 contains a listing of all resource pairs in the system. Resource pairs are components of a link that comprise a starting and ending resource (both of which are relationships to the resource table), as well as information about the role and traversal constraints. Resource pairs may be related to profiles to scope their usage. Resource pairs are bound to links, and upon deletion of a link, that deletion will be cascaded to it’s resource pairs. The properties table 94 maintains a listing of XML attributes pertinent to each resource pair (such as graphic size, etc.) that are placed in markup at resolution time.

[0083] The profiles table 96 contains profiles defined within the system. Profiles are defined on a per doctype basis and may span multiple doctypes. Profiles are applicable values that, when references, set the usage and scope of a resource pair. The resource_pair_profile_xref table 98 contains relationships of resource pairs to profiles. A resource pair may be related with as many profiles as desired. The named_profiles table 100 contains a mapping of names to profiles. This may be used to create a grouping or categorization of profiles. Further shown are a link_folder_metadata table 106, a link_folders table 108, a link_title table 110, and a link_folder_title table 112.

[0084] A configuration table 104 may be provided containing system configuration information relevant to the DCAM server.

[0085] The transport tier of the DCAM System provides the communication link between the client tier and the server tier. The transport tier facilitates client/server communications.

[0086] The format for transport requests is SOAP (Simple Object Access Protocol). Thus, in one embodiment, the transport layer may be referred to as a SOAP transport layer. Each SOAP request is considered a transaction boundary. Alternately, transport-oriented languages (such as EJB, RMI, and CORBA) may be used to perform object marshalling. FIG. 6 expands upon FIG. 2 and illustrates the relationship between the transport layer 34 and DCAM Client Extensions 30. In the embodiment shown, the transport tier 34 is a SOAP aware implementation, used to enable client/server communications. Due to the simple nature of SOAP, all transactions occur through HTTP (hypertext transfer protocol). User Interface Code 115 goes through Scripting API 116 and/or Application API 117. DCAM Client Extensions 30 are applied before the code 115 is input to the Scripting API 116 and Application API 117. The Scripting API 116 and Application API 117 then output the Code 115 through the Service Interface 118 and finally to the transport layer 34.

[0087] The transport tier manages user sessions and transactions, by interpreting requests, executing transactions atomically, and returning those results. Each operation is performed by creating one or more data transfer objects along with a command, sending the objects and command wrapped in a SOAP request, processing the command which returns as results zero or more data transfer objects, and returning the results of the operation wrapped in a SOAP response. The client tier contains everything needed by users to create, delete and manage links. One embodiment of the client side implementation includes Epic customizations.
(menu items, hooks, etc.), user interfaces, and a client API that provides direct access to the server. The client API facilitates communications to the transport tier. Calls to the server are done within a transaction. The client component includes all classes necessary to create/submit a transaction and receive those results in the form of classes that represent server side objects. As shown in FIG. 6, the client API comprises two APIs, the scripting API 116 suitable for use from scripting languages that are not object aware and the application API 117 suitable for use from languages that are object aware.

[0088] FIGS. 7-16 illustrate specific embodiments of user interface details. These figures are intended as illustrative only and are not intended to limit the present invention.

[0089] FIG. 7 illustrates a screen shot of an Insert Link dialog 120 for inserting a link from the current document in accordance with one embodiment of the present invention. The Select Target(s) from box 122 enables the user to select targets from the DCAM Repository, the Current Document, or the Web. The Current Document Selection is displayed in FIG. 7. The Current Document Targets table 126 displays available targets in the current document. The user may select any of the targets. The Selected Targets table 128 lists the targets selected. The Display Button 130 allows the user to select the fields to display in the Selected Targets table 128. The Insert button 134 allows the user to insert a link reference in the current document at the cursor location. The Modify button 136 launches a Modify Link Properties dialog (see FIG. 10) allowing the user to update the attributes of a link.

[0090] FIG. 8 illustrates a screen shot of an Insert Link dialog 120 for inserting a link from the DCAM Repository in accordance with one embodiment of the present invention. The Link Name field 124 allows the user to assign a human readable name to the link. The Select Target(s) from box 122 enables the user to select targets from the DCAM Repository, the Current Document, or the Web. The DCAM Repository selection is displayed in FIG. 8. The DCAM Targets table 144 is a tree control of the available targets in the DCAM Repository and allows the user to select an existing link from the Repository. The Selected Targets Table 128 lists the targets selected. The Display Button 130 allows the user to select the fields to display in the Selected Targets table 128. The Modify button 136 launches a Modify Link Properties dialog (see FIG. 10) allowing the user to update the attributes of a link.

[0091] FIG. 9 illustrates a screen shot of an Insert Link dialog 151 for inserting a link from the Web in accordance with one embodiment of the present invention. The Link Name field 124 allows the user to assign a human readable name to the link. The Select Target(s) from box 122 enables the user to select targets from the DCAM Repository, the Current Document, or the Web. The Web selection is illustrated in FIG. 9. The Web Target Name field 154 allows the user to assign a human readable name to the target. The Web Target URL field 155 allows the user to assign a URL to locate the target resource. If the Browse button 156 is used, the URL field 155 will be populated with the location selected. The Browse button 156 launches a Web browser and records the URL of the currently selected page in the browser in the URL field 155. The Selected Targets table 128 lists the targets selected. The Display button 130 allows the user to select the fields to display in the Selected Targets table 128. The Modify button 19 launches a Modify Link Properties dialog (see FIG. 10) allowing the user to update the attributes of a link.

[0092] FIG. 10 illustrates a screen shot of a Modify Link Properties dialog 210. This dialog allows the user to modify any of the properties associated with the link. The Source Label field 212 allows the user to assign a label to the source resource. The Target Name field 214 displays the name of the target resource. The Target Location field 216 displays the locator of the target resource. The Target XML ID field 218 displays the XML ID of the target resource. The Target Label field 220 allows the user to set the label of the target resource. The Refer to Target or Include Content radio buttons 222 and 224 respectively, specify how the link should be handled when resolved. If Refer to Target 222 is selected, the target’s markup template is used to create a link to the content when the link is resolved. If Include Content 224 is selected, the target resource is included at the location of the link when the link is resolved. The Markup Combo Box 226 allows the user to select the type of link markup represented by the link. The list may be prepopulated with link tags specified in the document types DCF file. The Profiles button 228 launches the Apply Profiles dialog (see FIG. 12), allowing the user to apply profiles to the link.

[0093] FIG. 11 illustrates a screen shot of a DCAM Explorer dialog 240. The DCAM Explorer permits automatic target registration and user defined folder creation. The user may choose links, resources, and data merge components. Thus, the dialog allows the user to manage links in the same manner they would manage files on a file system.

[0094] The user may be provided with the ability to profile groups. Thus, the user may save and name a choice of profiles, which may later be applied to an object by selecting the named profile group, rather than requiring each choice to be selected each time. The user may also save and name a choice of profiles, which may later be used to designate the profiles to use for resolution purposes (such as at block 14 of FIG. 1) by selecting the named profile filter, rather than requiring each choice to be selected each time. Profile values may be added dynamically during the editing process. The user may define particular elements to restrict individual profiles to or from. The user may define element markup for profiling rather than simply attribute markup. Further, the user may define the specific order in which profile values appear.

[0095] Hierarchical profiles may be provided to allow the user to apply a group of profiles simultaneously to an object based on selecting a containment node. Typically, the profiles are applied at the leaf level. Radio profiles may be provided. Radio profiles are mutually exclusive profiles where only one choice is allowed. Profiling may be done via containment. Containment is the concept where a single profile value represents the inclusion of other lower level profiles (for example, “top secret” including “secret, classified, and unclassified”). Further, named profiles may be used (profile groups and profile filters), profiles may be restricted to or from particular elements, and logical expressions (AND, OR, NOT, EQUAL, XOR) may be included in profile filters.

[0096] A screen shot of an Apply Profiles dialog 250 is illustrated in FIG. 12. The dialog 250 permits adding new
profile values, naming and saving profile selections, and supporting hierarchical and radio choice profiles. The Apply Profile tree control 252 shows the profile values represented as a tree control. The checkboxes or radio buttons allow the user to choose the profiles to apply. When the Apply Individual Profiles radio control 254 is selected, individual profiles are displayed. When the Apply Profile Group radio control 256 is selected, profile groups are displayed. The OK button 258 applies the profile values selected in the tree control to the applicable markup.

[0097] FIG. 13 illustrates a screen shot of an Update Element Profile dialog 272. The dialog allows a user to individually select profiles or to select a named profile group for updating the element profile. The tree control 274 shows the profile values represented by the profile groups. The user may choose the profile group to simultaneously apply all the listed values. The Apply Individual Profiles and Apply Profile Group radio buttons 275 and 276 respectively, allow the user to choose the profiles to update. When the Apply Individual Profiles radio button 275 is selected, individual profiles are displayed. When the Apply Profile Group radio button 276 is selected, profile groups are displayed. The OK button 276 applies the profile values represented by the profile group to the applicable markup.

[0098] A screen shot of a Search DCAM Links dialog 280 is illustrated at FIG. 14. This dialog 280 allows the user to search for links based on the properties of the link itself, its resource pairs, and the starting and ending resources of those resource pairs. The link and resource pair properties are listed on the Link Properties tab 282. The source resource properties are listed on the Source Properties tab 284. The target resource properties are listed on the Target Properties tab 286. The Advanced Parameters tab 288 allows users to specify additional search criteria, which can either be user defined or attributed on the resource pairs or less visible properties of links, resource pairs and resources (such as DCAM key). Most parameters, except the Referenced/ Included field 290, allow the user to specify an operator that discriminates the way to test the entered values with those in the database. These operators include: like (the value in the object contains the given value), greater than (the value in the object is greater than the given value), less than (the value in the object is less than the given value) and equals (the value in the object is equal to the given value). When search results are returned, they are displayed in the Search Results folder in the DCAM Explorer dialog (see FIG. 11).

[0099] A screen shot of an Export Linkbase dialog 300 is illustrated at FIG. 15. This dialog allows the user to export the data for all of the DCAM links in the current document, including resource pairs, source, and target resources. The XML file containing the link data may be referred to as a linkbase. A linkbase typically contains lay names, custom metadata, XML IDs, and URLs. It may be sent to a downstream process or organization so that the link information may be used outside of DCAM. Downstream applications may require link data in specific formats, and multiple linkbase templates may be installed for different formats. The Save As control 301 and associated Browse button 302 specify the output linkbase filename. The Stylesheet drop down list 303 and associated Browse button 304 select which XSL stylesheet is used to create the linkbase. The stylesheet determines the format of the linkbase. In a specific embodiment, the stylesheet drop down list 303 is populated with a list of XSL for the document type that can be used for linkbase export. All xsl files in the doctype directory are examined and the ones that include “linkbase” in their “CompositionType” list are displayed. “CompositionType” is generally specified in a profile near the beginning of the XSL file. The Browse button 304 allows the user to select any XSL stylesheet, regardless of whether “linkbase” is in its “CompositionType” list. Further, again in a specific embodiment, Epic sets up a composition pipeline that includes a link filter (filters out non-DCAM markups), a link resolution filter (retrieves link data from database), and a filter for the selected stylesheet. A linkbase export stylesheet may be used to transform canonical DCAM link markup into desired linkbase markup.

[0100] A screen shot of a Profile Filter dialog 308 is shown at FIG. 16. The dialog 308 is used to set the profiles during resolution. The default assumption is all values are logically ANDed together. To use more complex logical expressions, the user selects the Filter Group button 310. The Filter Group button 310 launches the Profile Filter Group dialog (see FIG. 29).

[0101] The Profile Filter Group dialog 312 is shown at FIG. 17. This dialog 312 allows the user to choose a predefined name profile filter group for setting the profiles. These profile filter groups support complex, logical expressions such as logical conjunction, logical disjunction, logical inequivalence, logical equivalence, and logical negation.

[0102] As shown in FIGS. 18-29, a link definition may either be created manually or automatically. When an object is registered with the link repository, any elements within the object which are IDed can automatically create a special type of link definition, called a Target, in the user interface. The DCF can define what elements to automatically ID and what attribute or element content to use as the name of the link target. The attribute to use as the ID is registered with the DCF file. If configured in this way, the application automatically registers the link target with the repository for use later by other authors.

[0103] FIG. 18 provides a basic diagram of creating a link definition. FIG. 19 expands on the creation of a link. As shown, this may be done by using “autoregister.” If the document registered flat is not set, the resources in the document must be set at the user interface level. At that point, or if the document registered flag is set, it is determined whether the Autoregister flag equals true. If no, final state is reached. If yes, the next step at the user interface level is to walk the document. Next, link definitions are created in the repository for ever link type element. Within the client code, this involves committing all links and resource pairs to the repository. Within the transport layer, batch create links is performed. Within the server code, all links and resource pairs are created in the database and IDs are returned for created links. At the user interface level, the next step is to encapsulate the link markup with a tag such as “<atidcam:m:link>. Link names are then determined through the DCF configuration file and final state is reached.

[0104] FIG. 20 illustrates a diagram of creating a link definition. This can involve simply naming the link and recording the link in the repository. Alternatively, creating a link definition for a resource pair involves setting the reference starting resource, setting the reference ending resource, setting profiles, and setting any other attribute values. The link can then be recorded in the repository.
FIG. 21 illustrates a diagram of creating an ID/IDREF link. The first step is to link internally. Next, resource information is captured. The link definition is then created and a link reference is inserted. FIG. 22 diagrams linking internally. If the element is already IDed, the ID Label is selected from the list. If the element is not IDed, the user ID is selected. FIG. 23 diagrams IDing an element. The first step is opening the object. Next, the user views selects the element. An ID is autogenerated. The user then sets the element’s ID attribute to the generated ID. The ID is then labeled. FIG. 24 diagrams inserting a link reference. The first step is to register a starting resource with the link. The next step is to insert the link reference in the object.

FIG. 25 is a diagram illustrating creating an outward link. The first step is to browse/select an ending resource. The next step is to create a link definition. Finally, the link reference is inserted.

FIG. 26 diagrams browsing/selecting a starting resource. The first step is to browse/select an object. If the link is internal to the object, the next step is to link internally. After linking internally, or if the link is external to the object, the next step is to capture resource information. FIG. 27 diagrams browsing/selecting an ending resource. The first step is to browse/select an object. If the link is internal to the object, the next step is to link internally. After linking internally, or if the link is external to the object, the next step is to capture resource information.

FIG. 28 is a diagram illustrating inserting a link repository reference. A file link reference is inserted to the link repository. A CMS link reference is inserted into the link repository. A web link reference is inserted into the link repository. A link repository reference is inserted into the link repository. Diagrammatically, the link repository reference is between the Link Author and the Object Author. FIG. 29 further diagrams inserting a link repository reference. The first step is to browse the link repository. Next, a link is selected. Finally, a link reference is inserted.

FIG. 30 is an alternative to FIG. 1 illustrating composing with the DCAM system in more detail. Composition starts with XML Documents 400 which are sent to an E3 Server 402 at block 410. The E3 Server 402 parses the documents (block 10 of FIG. 1) and sends them to the Content Pipeline 404 at block 412. As links are found in the content (block 12 of FIG. 1), the links are sent to the DCAM Resolver 406 at block 414. The links are resolved against the DCAM Repository (block 14 of FIG. 1) and the resolved links are sent back to the Content Pipeline 404 at block 416. Data merge queries are also processed against the Data Merge Data Store 408. The Content Pipeline continues resulting in composed documents such as HTML in the embodiment shown.

The functioning of the link resolution filter 340 is illustrated in FIG. 31. SAX events from object processing are input into the Link Resolver Filter 340. The reference link is converted to markup the SAX events. The SAX events are then output from link processing.

To use the client component, an application calls the establishClientSession() method in the Application API. This call initiates a connection to the transport tier, which passes on the create session request to the server. If the user authenticates, a session is created. Once a Client Session object is created, user actions may generate calls to the Service interface. That interface generates a SOAP request and sends it to the transport tier which processes the transaction through the server. Upon completion, the server returns a SOAP document containing all of the results.

FIG. 32 depicts the configuration components for linking, data merging, and profiling. Because customers have many different needs around linking and data merge, these functions are highly configurable. FIG. 33 shows the general nature of the configuration components. The markup resolution is configured using XSL stylesheets. XML files are used to configure how targets are automatically identified (Resource Definition), both the database queries and the parameter handling for data merge, and the configuration of profiling.

Data merge, in accordance with the present invention, permits including content from a separate data store. Data merge is the incorporation of references to external data sources in a document, and the periodic resolution of those references. A reference to some external data is called a query. Three authoring stages may be identified: query declaration; reference to a query declaration; and update of one or more query results. Named declarations are reusable, that is, they may be reference many times. They are given a name when created and their location defines their scope. The name must be unique for the scope. Query declarations with no name are not reusable. The point of declaration is the only reference. Query results appear at the location of a query reference. Results are inserted at the time a reference is inserted or whenever a reference to a query is updated.

A query declaration must refer to an external data source, called the query definition. A query definition comprises of a UI component and a formal definition. The UI component includes: the name of the query definition, the parameters that must be passed to the query, whether the query returns document content of name/value pairs, and if the document content is returned, a representative top level tag for quick context verification. The name of the query definition links a document’s query declaration to a query definition. The formal definition includes: a source stage, one or more transformation stages, a description of the order in which the stages are to be applied, and a mapping of UI parameters to actual parameters for each stage. The source stage may be any program that generates a Document Object Module (DOM) node. The actual source may be a database, a file, a URL, or some external process. The program is responsible for presenting the result as a node, perhaps using some simple markup to represent value pairs. The transformation stages take a DOM node as input and generate a new DOM node as output.

The data merge framework is designed for flexibility and extensibility. It adapts the Model-View-Controller (MVC) design paradigm and uses a pipeline structure for handling data acquisition and processing. The MVC paradigm separates the business logic from the user interface, allowing both sides to be modified independently. A pipeline allows easy reuse of components. FIG. 34 illustrates the overall flow of inserting data into a document. Three major components are shown in the diagram. DOM Server 350, Data Merge Controller 352 and Query 354. DOM Server 350 takes input from a user and displays the result. Data Merge Controller 352 interprets user inputs and passes the
information to and from a Query 354. Query 354 processes the information provided by the Controller 352. When a user issues the insert_query or update_query command, the Data Merge Controller 352 constructs a set of parameters based on the user inputs and current document instance and then selects a Query 354 for execution. Upon returning of the query, the node is inserted by the Data Merge Controller 352 into the DOM Server 350.

DOM Server 350 refers to the program which is capable of manipulating a DOM instance and displaying a DOM instance when in the interactive mode. Data Merge Controller 352 is a program that consists of helper functions that are assembled from DOM API. It receives requests from the user, distributes the requests to queries and then inserts the result into a document. It is also responsible for updating changes and retrieving data fields. A Query 354 comprises one Source and zero or more Transformers. A source is a component that can generate a DOM node. A Transformer takes an existing node as input and returns a modified node as the output. A Query process generates a DOM node (which may be an element, document fragment, or document) and returns it to the Data Merge Controller. The components in a Query communicate with each other via a DOM node. A Query generally comprises three components: SQL, Query, Transformation and XPath. The Query process retrieves data from a database and transforms the result using some XLT Transformation. Before returning the result, an XPath filter is applied to retrieve only interested components. The query process involves a sequence of transformations.

Although the present invention has been described with reference to preferred embodiments, persons skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

We claim:
1. A method for managing dynamic assembly of content in a document, comprising:
   parsing a document to find at least one link;
   resolving the at least one link against a DCAM Repository resulting in a resolved link to content;
   assembling the resolved link to content into the document.
2. The method of claim 1, further including profiling the link when the link is resolved.
3. The method of claim 1, further including processing data merge queries.
4. The method of claim 1, further including determining link type of the at least one link.
5. The method of claim 4, wherein if the link is a navigation link, updated markup is inserted into the document.
6. The method of claim 4, wherein if the link is an embedding of content, the method further includes embedding the content into the document.
7. A method for managing dynamic assembly of content, comprising:
   determining an access point, the access point invoking a dynamic content assembly request;
   selecting one of a plurality of contents for assembly from the access point; and
   assembling said contents dynamically based on rules; and
   making said dynamically assembled content available from said access point.
8. The method of claim 7, wherein the access point is the start point of a relationship arc.
9. A method for managing creation of links between and within content modules, comprising:
   determining first and second end points, the first end point being a start of a relationship arc and the second end point being an end of a relationship arc;
   creating a relationship arc;
   indicating an appropriate audience for the relationship arc; and
   dynamically resolving the relationship arc based on the audience.
10. The method of claim 9, wherein determining end points and creating a relationship arc further comprise:
    determining possible end points based on rules;
    capturing metadata about each possible end point based on rules; and
    validating the end points used within a document or system.
11. The method of claim 10, wherein possible end points are found using a navigation mechanism for finding end points.
12. A method for dynamically assembling documents, comprising:
    entering a target with assembly rules;
    entering a link that points to the target; and
    resolving the link into content.
13. A system for managing dynamic content assembly, comprising:
    a server tier comprising a database bound to a server application, the database including linking information;
    a client tier enabling a user to create, delete and manage links; and
    a transport tier for communicating between the server tier and the client tier.
14. The system of claim 13, wherein the server application includes logic for interacting with the database.
15. The system of claim 13, wherein the transport tier is a SOAP transport layer.

* * * * *