RDF Models for Dynamic Syndication and Wireless Applications

Leon Shklar
Information Architects
70 Hudson St.
Hoboken, NJ 07030, USA
shklar@cs.rutgers.edu

ABSTRACT

Machine-understandable metadata is providing the foundation for next-generation frameworks that enable automated construction of server-side Java applications. Such applications are composed of metadata objects implementing RDF models and Java classes that use metadata objects as processing context. Using RDF to support dynamic transformation of content to different channels and devices opens up opportunities for multi-purpose content services that target different audiences and a wide variety of desktop and wireless devices. Such services do not have to depend on costly maintenance to keep up with new, evolving, and personalized devices. Properly designed applications that are built in the era of cell phones and palm devices should still work for interactive TV and futuristic Star Trek-like personal communicators.

Keywords

Metadata, RDF, Transformation, Wireless.

1. INTRODUCTION

The Resource Description Framework (RDF) is a metadata standard that was designed by the World Wide Web Consortium (W3C) to enable Web applications that depend on *machine-understandable* metadata and to support interoperability between such applications. It targets a number of important areas that include dynamic syndication and personalization, mobile devices, resource discovery, intelligent agents, content rating, intellectual property rights, and privacy preferences. RDF uses XML to encode and transport RDF models but may also use alternative mechanisms in the future.

Applying RDF to redesigning the syndication process makes it possible to model content subscriptions. Instead of having to receive scheduled distributions of content, subscribers can direct their customers to syndicators' sites, while syndicators use the subscription models to recognize subscriptions (e.g., based on `User-Agent` or `HTTP-Referer` HTTP headers) and dynamically tailor content to subscribers' profiles. RDF-based syndication models can be naturally extended from targeting different content channels to enabling the expanding variety of desktop and wireless devices.

Emerging commercial products support multiple devices by building libraries of device-specific XSLT stylesheets to transcode XML content. Such stylesheets may be fairly efficient when compiled into Java bytecode (Sun distributes the XSLT compiler). The problem is in maintaining stylesheet libraries for the rapidly growing variety of new and evolving devices, let alone device personalization. The proposed solution is to change the level of granularity of transformations and design them for individual features rather than devices. Using RDF models to implement device and user agent profiles, it is possible to dynamically compose transformations by adapting and combining feature-based stylesheets, making it unnecessary to build and maintain ever-expanding libraries of complex device-specific components.

W3C, in coordination with the Wireless Access Protocol (WAP) Forum, is developing the *Composite Capabilities/Preference Profiles* (CC/PP) specification as the standard for setting device and user agent preferences. The upcoming RDF-based specification would allow defining a device by its features (e.g., screen size, keyboard (if any), display characteristics, etc). Next-generation servers that target multiple devices would combine device and user agent profile information with connection bandwidth and use it to construct stylesheet transformations. An efficient server would optimize stylesheet construction by caching components, as well as intermediate composites. For example, caching device-specific stylesheets that are constructed based on individual device profiles and combining them with stylesheet components that are determined by the operating system, user agent software, and connection bandwidth.

Currently, the main practical limitation is the lack of CC/PP support on the part of device vendors and service providers; it is likely to be remedied in the near future. In the prototype, we implement our own version of this service based on the local database of CC/PP-based device profiles. This implies that administering the server involves maintaining device and user profiles, which is orders of magnitude less work than maintaining device-specific XSLT transformations. Even this overhead will not be necessary when CC/PP becomes the recommendation and vendors start providing CC/PP services.

2. DEVICE SPECIFICATIONS

We begin by discussing sample RDF specifications for devices and user agents. These specifications, when represented as

Semantic Web Workshop 2001 Hong Kong, China.

addressable metadata graphs, will provide processing context for HTTP requests.

Consider the XML-encoded RDF specification describing a handheld device "xyz":

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2000/07/04-
ccpp#"

xmlns:uaprof="http://www.wapforum.org/UAPROF/ccpps
chema-19991014#">
  <rdf:Description
about="http://www.mozilla.org/wap/profiles/Mozilla
">
    <type resource="http://www.xyz-
mobile.com/profiles/Schema#UserAgent" />

<uaprof:BrowserName>Mozilla</uaprof:BrowserName>

<uaprof:BrowserVersion>Symbian</uaprof:BrowserVers
ion>
    <uaprof:CcppAccept>
      <rdf:Bag>
        <li>text/plain</li>
        <li>text/vnd.wap.wml</li>
      </rdf:Bag>
    </uaprof:CcppAccept>
  </rdf:Description>
</rdf:RDF>
```

Here, the subject of the description is `http://www.mozilla.org/wap/profiles/Mozilla`. The `rdf:type` element identifies the subject resource, the `uaprof:BrowserName` identifies the browser name as *Mozilla*, `uaprof:BrowserVersion` identifies the browser version as *Symbian*, and `uaprof:CcppAccept` defines MIME types supported by the browser.

Individual devices and user agents are likely to differ from default configurations. For example, my *xyz* device may have an optional screen, and I may have configured my browser to support HTML in addition to plain text and *WML*. However, it is possible to incorporate default configurations by reference. My profile would have the following form:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2000/07/04-
ccpp#"
xmlns:uaprof="http://www.wapforum.org/UAPROF/ccpps
chema-19991014#">
  <rdf:Description
about="http://www.ia.com/leon/profile">
    <ccpp>
      <rdf:Description
about="http://www.ia.com/mobile/Hardware/device112
3">
        <rdf:type resource="http://www.xyz-
mobile.com/profiles/Schema#Hardware" />
        <ccpp:Defaults
rdf:resource="http://www.ia.com/xyzProfile" />
    <uaprof:ScreenSize>320x200</uaprof:ScreenSize>
      </rdf:Description>
    </ccpp:component>
     <ccpp:component>
      <rdf:Description
about="http://www.ia.com/mobile/UserAgent/Mozilla-
beta">
        <rdf:type resource="http://www.xyz-
mobile.com/profiles/Schema#UserAgent" />
        <ccpp:Defaults
rdf:resource="http://www.mozilla.org/wap/profiles/
Mozilla" />
        <uaprof:BrowserVersion>Symbian-
beta</uaprof:BrowserVersion>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

Individual Here, `ccpp:Defaults` elements reference default profiles while `uaprof:ScreenSize` and `uaprof:BrowserVersion` override default properties of the device and user agent correspondingly. The resulting profile, when interpreted by the server-side transformation agent, would control automated assembly of an XSLT stylesheet that get compiled into Java bytecode and cached on the server.

A feature-based approach to building device profiles helps to automate targeted transformations of XML content. A new and unknown device may be analyzed and mapped to a known device with the closest set of features. The resulting specification gets stored in a repository and serves as the basis for combining transformations into a single XSLT stylesheet.

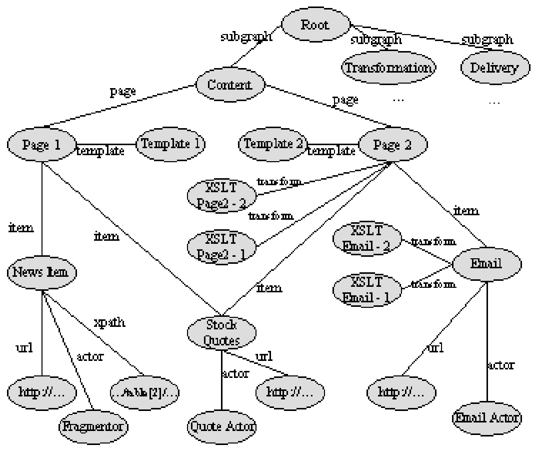

Figure 1. Sample Model - Content Subgraph.

3. CC/PP SERVER

Since CC/PP is defined as an RDF application, our approach is to build a specialized RDF server that lends itself to constructing CC/PP-based applications. Our RDF server is designed to construct RDF models from XML specifications and to use these models as processing context for programmable plug-ins. In this prototype, we are not attempting to build an RDF-based inference engine. Instead, we are taking a pragmatic approach by supporting a Java API for pluggable *Actor* modules that get invoked in the context of the nodes in the RDF graph.

The system graph is composed of three sub-graphs - the *content graph* that models content and encapsulates content retrieval functionality, the *delivery graph* that models end-user devices and browsers, and the *transformation graph* that models feature-based transformations (see Figures 1 and 2). The distinction between the sub-graphs is semantic; structure-wise they are all RDF graphs and are connected to the same root node. For convenience, we will refer to *content nodes*, *delivery nodes*, and *transformation nodes* depending on the context of the respective graph. Note that the same node may belong to more than one graph (e.g., the node associated with the "screen size" feature belongs to both the delivery graph and the transformation graph).

The idea of our system is to promote the design of content transformations for features and not devices. Adding support for a new device will only require creating a new feature profile and even this will become unnecessary in the future. The further idea is to separate out content-independent components of feature-based transformations and to minimize the amount of design work needed to define transformations when adding new content.

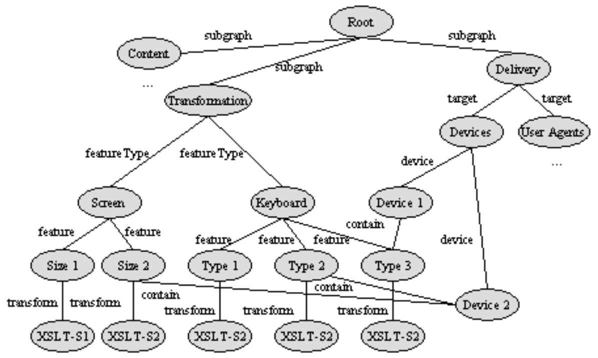

Figure 2. Sample Model - Transformation and Delivery Subgraphs.

3.1 Server Operation

Details of the server operation are illustrated in fig. 1. User requests are always directed at a content node. When the Controller receives the first request in a session, it performs the following steps:

1. Sets references to content retrieval and transformation Actors according to the following priorities (in descending order):
   - 1.1 information in the request;
   - 1.2 properties of the content node;
   - 1.3 global defaults.
2. Invokes the transformation Actor to perform the following steps:
   - 2.1 Establish a new session.
   - 2.2 Traverse the delivery graph to compute device and user agent profiles and store them in the session.
   - 2.3 Check for the availability of cached content-independent transformations for computed profiles; if such transformations are not available, traverse the transformation graph, compute the transformations, and store them for future use.
3. Invokes the content retrieval Actor to perform the following steps:
   - 3.1 Recursively refer to other content nodes and invoke their associated content retrieval Actors (if required).
   - 3.2 Compute and apply content-specific transformations based on device and user agent profiles.
   - 3.3 Apply profile-based content-independent transformations.

The architecture allows for multiple content retrieval and transformation actors that implement different policies for content aggregation, feature profiling, and the composition of stylesheet transformations. Transformation actors are responsible for compiling feature profiles that they use to compose XSLT-based transformations. Feature profiles are also used by content retrieval actors for the purpose of selecting and composing content-specific transformations. Content retrieval actors get invoked recursively and may or may not apply transformations to their content components.

3.2 Content Transformation Actors

Content transformation actors are responsible for figuring out device and user agent features based on device identification and user preference profiles that may include information about custom extensions (e.g., additional memory) or preferences regarding not making use of certain features that are available with the device (e.g., keyboard).

In the future, with the support for the CC/PP standard, we will expect browsers to submit either CC/PP specifications or references to such specifications. The specification may be further refined based on user preferences but it will be the primary source for building device feature profiles. Our implementation is designed to function in the absence of CC/PP support but benefit from such support when it becomes available. To this end, the server maintains a local repository of device specifications. Device identities may be inferred by the transformation actor based on information in the request (e.g., User Agent). They may also be identified by authenticating users and retrieving device preferences from their profiles.

Having inferred information about the type of the device the transformation actor checks for the availability of user preferences and creates the device feature profile based on combining this information. For example, the User Agent header in the request indicates the version of the browser that runs on a Motorola cell phone of a particular model. The transformation actor queries the local repository for device features and creates the initial profile. This profile may get modified based user preferences.

For example, I may have upgraded memory on my Motorola cell phone. The transformation actor can at best infer the make and model of my device based on information in the request but it can not learn about and take advantage of my upgraded memory. However, I can list my devices and their upgrades in my profile. The transformation actor can use this information to update the initial feature profile that gets retrieved from the local repository.

Once the transformation actor compiles the feature profile, it uses it to combine feature-based XSLT transformations (if the target transformation is not already available). It passes the feature profile to the content retrieval actor that may use it to select and/or compose content-specific transformations. Once the transformation actor receives partially transformed content from the content retrieval actor, it applies feature-based transformations and returns the result.

3.3 Content Retrieval Actors

Content retrieval actors are responsible for interpreting request context and using it to control retrieval and transformation of content. A content retrieval request always targets a metadata node that provides the processing context, along with request headers and session information which includes the feature profile of the target device. Node metadata may reference content, applications, or recursively refer to other metadata nodes. It may also reference content-specific XSLT stylesheets that get selected based on feature profiles for target devices.

For example, a metadata node may reference a fragment of an XML file, and two other metadata nodes that, in turn, are associated with a database query and an LDAP query correspondingly. The content retrieval actor that gets invoked in the context of the first node retrieves the XML fragment and recursively invokes content retrieval actors for the referenced nodes. Content retrieval actors for the query nodes may select and apply their own transformations to query results prior to returning them to the original content retrieval actor. Having received all responses, the original content retrieval actor may apply different transformations depending on whether the browser is running on a cell phone or a desktop computer.

3.4 Summary

Our RDF server is designed to provide metadata context for functional actors. Transformation actors are responsible for building feature representations of end user devices and for using the feature models to assemble content-independent XSLT transformations. Content retrieval actors are responsible for retrieving and partially transforming content. Content design that is conducive to applying content-specific transformations may have dramatic affect on presentation quality.

4. CONCLUSIONS

The future direction of RDF, CC/PP, and related standards has immediate implications on design and development of wireless applications. This includes building separate classes responsible for the assembly and transformation of content, and implementing feature-based XSLT stylesheets that may be assembled into device-specific transformations. The way is open for commercial wireless application development products that do not make use of proprietary formats and protocols and that establish a clear path for supporting W3C and WAP Forum standards.

RDF is emerging as the foundation for next-generation frameworks that enable automated construction of Java applications. Such applications are composed of networks of metadata objects implementing RDF models and Java classes that use metadata objects as processing context. Using RDF for wireless applications opens up opportunities for building next-generation mobile services that don't depend on costly maintenance to keep up with new, evolving, and personalized devices. In other words, properly designed applications that are built in the era of cell phones and palm devices would still work for future microwave ovens that connect to the Internet to download cooking instructions.

5. REFERENCES

[1] Composite Capabilities/Preference Profiles, Work in Progress, W3C.

[2] Resource Description Framework Model and Syntax Specification, W3C, 1999.

[3] Resource Description Framework Schema Specification 1.0, W3C, 2000.

[4] Didier Martin, How would you like that served?, xml.com, 2001.