# Publishing Semantic Web Content as Semantically Linked HTML Pages

Eero Hyvönen*
University of Helsinki
Helsinki Institute for Information Technology
(HIIT)

Arttu Valo
Helsinki Institute for Information Technology
(HIIT)
University of Helsinki

Kim Viljanen
Helsinki Institute for Information Technology
(HIIT)
University of Helsinki

Markus Holi
Helsinki Institute for Information Technology
(HIIT)
University of Helsinki

## ABSTRACT

The Resource Description Framework RDF is used to describe content, such as HTML pages and other documents, for the machines to interpret on the Semantic Web. In contrast, we consider the problem of rendering RDF content for the human interpreter by transforming RDF descriptions into semantically linked HTML pages. In our approach, the layout of the pages is described by HTML templates and the semantic linkage structure of the page repository is defined in terms of logical rules based on the RDF repository. The rules are mapped on the HTML level using template tags. This approach provides a simple method for creating and publishing a human readable version of RDF content on the web. We present a tool called *SWeHG* for generating a static, semantically linked site of HTML pages from an RDF repository. As a case application, a web exhibition is generated from a museum photo collection.

## 1. SCOPE AND MOTIVATION

A key idea of the Semantic Web[1][1, 3] is to enrich the web with metadata describing resources, such as web pages, documents, photos, and real world objects in a machine understandable format. For representing the metadata, ontologies and the resource description framework, consisting of RDF[8] and RDF Schema [2]) specifications, are commonly used. RDF is intended for the machine to interpret but the ultimate goal is usually to present the semantic content in one way or another also in a human readable form. This is typically done by dynamic HTML pages provided by a semantic portal[2].

The semantic portal approach creates a hindrance for publishing semantic content on the web from the individual content provider's view point. Firstly, only content of certain type conforming to the portal's application ontologies can be published. Secondly, the publication process is dependent on the organization maintaining the portal application. Most content providers are able to publish only static web pages with the help of their Internet Service Provider since they do not have the infrastructure or capabilities for maintaining a portal of their own. The notion of semantic portal is in this respect in contrast with the democratic idea of the current web, where everybody can publish content easily and independently by just maintaining HTML files in a public directory. Third, the dynamic content in semantic portals cannot usually be found by search engines as easily as content represented as static HTML pages.

More and more content will be available in the RDF(S) format in the future. An important question for the success of the Semantic Web is how easily everybody can make use of it and how easily it can be published it in a human readable form on the web. To address this problem, we propose the idea to transform RDF(S) content into static HTML pages which can then be published easily and independently by using the traditional "HTML in a public directory" -approach.

In the following, we first set constraints for the RDF(S) and HTML repositories to be used in our scheme. After this the transformation from an RDF(S) repository to an HTML site is discussed and our experimental *SWeHG* tool is presented. As an application experiment, the RDF(S) repository of photographs [7] of the Helsinki University Museum[3] is transformed into a semantically linked site of HTML pages. In conclusion, experiences of our research and experimentation are summarized and directions for further research outlined.

## 2. TRANSFORMING RDF(S) TO HTML

Representing content in RDF(S) is becoming more and more popular. Examples of potential application areas of RDF(S) include photo archives, artifact collections in museums [5], MP3 music repositories, paintings in a gallery, books in a library, entries in a thesaurus, articles of an encyclopedia, and web pages of a portal or on the WWW[4].

In our approach, an RDF(S) repository is considered on the following conceptual levels. First, on the data level the repository contains the actual data. For example, in a photo repository the photos

*Address to all authors: University of Helsinki, P.O. Box 26, 00014 UNIV. OF HELSINKI, FINLAND
Email to all authors: firstname.lastname@cs.helsinki.fi
Research group home page: http://cs.helsinki.fi/group/seco/

[1]http://www.w3.org/2001/sw/

[2]Cf., e.g., http://www.ontoweb.org, http://www.mindswap.org.


Reference: *Proceedings of XML Finland 2003*, Oct 30–31, 2003, Kuopio, Finland.

[3]http://www.hwelsinki.fi/museo/

[4]See e.g. http://dmoz.org for an RDF(S) repository of metadata about web resources on the WWW.

are data. Second, on the metadata level information about the data is represented as RDF statements. For example, the metadata of a photo may indicate the photographer, what the photo depicts etc. Thirdly, on the ontology level, the vocabulary used at the metadata level is defined by an RDF Schema. For example, the schema may tell that the photographer is an instance of the class "Human" and that the Dublin Core property "cd:creator" is used as the statement predicate. Fourthly, on the logic rule level, semantic relations between the resources in the repository are considered. For example, a binary relation between two photographs may be that they are taken at the same location, at the same time, but by different photographers. The data model of an RDF(S) repository simple: a set of triplets which can be interpreted as a directed graph.

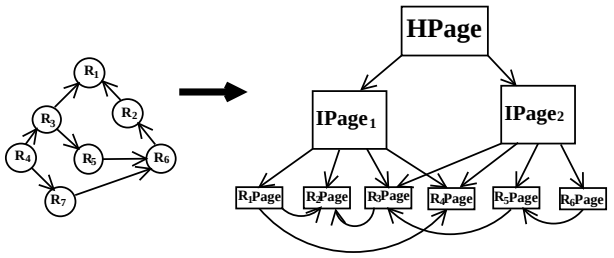


**Figure 1: Transforming HTML pages from an RDF(S) repository.**

Our goal is to generate a semantically linked static HTML page repository (SHR) from an RDF(S) repository. The SHR consists the following kind of pages:

1. A *home page (HPage)* defines the entrance page to the repository. It is typically defined by a HTML page that contains frames that are used to show other pages in the SHR.

2. *Resource pages (RPage)* depict one resource such as an ontological concept (e.g., a class) or a piece of data with its metadata. Each resource that is intended to be shown to the end-user has a RPage of its own. For example, a photo with its metadata can be rendered as an RPage.

3. *Index pages (IPage)* classify RPages along conceptual hierarchical dimensions. We call such classifications *views* as customary in the field of information retrieval [10, 4].

Figure 1 illustrates this RDF2HTML transformation. The RDF(S) graph is on the left. On the right, the HPage has links to various IPages classifying the underlying RPages that are related with each other by semantic links.

In our approach, the RDF2HTML transformation is specified on two separate levels. Firstly, on the *HTML level*, the layout of HPages, IPages, and RPages is specified using layout templates. The layout can be specified by a layout designer using ordinary HTML extended with a few additional tags. Programming skills are not needed. Secondly, on the *RDF level*, the semantic linkage between the pages is determined using logical predicates. These predicates define the semantics of the tags used on the HTML level. The definitions are provided by an application programmer.

The set of RPages to be generated is constrained by a selector predicate. It selects resources—*context resources*—from the RDF repository. The RPages are generated one after in relation with the selected context resource. For example, data record resources representing the metadata of photographs can be selected as context resources. In this way, one RPage for each photograph will be generated.

The HTML pages corresponding to context resources are rendered by evaluating an HTML template. Reference from the HTML level to the underlying RDF level is made with tags. For example, the tag `<swehg:getProperty name="year"/>` on an RPage template refers to the "year" property value of the particular context record at hand. The semantics of the tags, such as the attribute "year" above, are specified declaratively by logical rules described in the following section. The advantage of the tag mechanism is that the layout designer does not need to know how "year" is actually implemented by the rule designer.

For example, consider an application containing personal tourist photograph records. The HPage can be an HTML page containing frames for RPages and IPages. The photo records can be selected as context resources and the corresponding RPages render the images with the metadata, such as the title and photographer. The page may also contain semantic links to other pages based on logical rules and the underlying RDF(S) data. Tags for doing this will be described later. For example, if an RPage depicts the Eiffel tower, it may have a link "Another tower photo" pointing to another RPage depicting a tower, e.g., the Tokyo tower.

IPages can be created to index the RPages along different views. For example, if the underlying RDF(S) ontology has a "Building" class with subclasses "Tower", "Castle" etc., this hierarchy can be rendered to the user on the HTML level by an IPage depicting a view hierarchy whose leaves are links to RPages depicting towers, castles, etc. The view is rendered on the IPage by using a special tag.

# 3. PREDICATES AND TAGS

The RDF2HTML transformation is based on a set of declarative logical definitions of *selectors*, *properties*, *links*, and *views*. The tags used in generating the HTML pages are based on these logical definitions. The tags are application independent, but make reference to the current RDF repository by application specific logical *predicates* that are used as tag attribute values. In the following the idea of logical selectors, properties, links and views is presented on the RDF level and their use in tags on the HTML level is explained. The examples are presented in Prolog syntax; our implementation is based on SWI-Prolog[5]. Here RDF triples define the predicate rdf(Subject, Predicate, Object). Notation `rdf:type` refers to the `type` property in the 'rdf' namespace of the W3C recommendation[6]. Underscore "_" means that the value can be anything. For example, for the variable `_:PhotoRecords` any namespace will match.

## 3.1 Selectors

A selector is defined by a unary predicate that evaluates "true" for the context resources to be selected. The names of the selectors are user-definable. For example, the selectors `photo` and `address` below evaluate true for instance resources of the corresponding ontology classes. In the latter case, the context resource is not the address instance itself, but the value of its property `representedBy`. Selectors can be arbitrarily complex Prolog predicates.

```
PhotoRecord = 'http://example.org/meta#photo'.
AddressClass = 'http://example.org/meta#address'.
```

[5]http://www.swi-prolog.org/
[6]http://www.w3.org/RDF/

```
photo(URI) :-
    rdf_instanceOf(URI, PhotoRecord).

place(URI) :-
  rdf_instanceOf(AddressURI, AddressClass),
  rdf(AddressURI, _:representedBy, URI ).
```

A selector can be used to define a set of context resources that should be rendered uniformly by an HTML template. For example, the HTML template below can be used for generating photo pages by the selector `photo` above. Selectors are RDF Schema dependent and have to be defined for different applications separately.

```
<swehg:templateFor selector="photo"/>
<html>
  <body>
    <h1> Photo page </h1>
    ...
  </body>
</html>
```

## 3.2 Properties

RPages are generated by HTML templates associated with selectors. The corresponding context resources typically have properties that should be rendered on the generated HTML pages. For instance, the photograph itself and its title are typical properties of a photo data record and should be shown on the RPage.

A property $P$ is defined as a function $P(x) : R \rightarrow R$, where $R$ is the set of resources in use. A property in *SWeHG* is defined by the `swehg_property` predicate:

```
swehg_property(Resource, Namespace,
               Property, Value)
```

The predicate evaluates the unique literal value of a resource property in the given namespace. For example, the "Title_of_photo" property of a photograph record can be defined in a generic namespace on the RDF-level as follows:

```
swehg_property(Resource, '',
               'Title_of_photo', Value) :-
  rdf(Resource, _:title, Value).
```

The tag `<getProperty name="Title_of_photo"/>` can then be used on the HTML template level to render the title of the context resource at hand. By using the `swehg_property`-predicate for naming properties, the HTML tag properties (here `Title_of_photo`) can be separated from the application specific properties on the RDF level (here `_:title`). By providing appropriate definitions on the RDF level for the HTML level `Title_of_photo` property, the same tags can used for publishing photo repositories based of different RDF schemas.

The HTML level properties are not necessarily direct RDF properties of the context resource, as in the above example, but the function definition can be arbitrarily complex. For example, below the "Resource_location" property is not a property value of the photo record itself, but the `_:name` property value of a related _:place property value. Notice also the re-use of the selector `place` predicate defined earlier.

```
swehg_property(Record, '',
               'Resource_location', Value) :-
  photo(Record),
  rdf(Record, _:place, Location),
  place(Location),
  rdf(Location, _:name, Value).
```

After defining a property on the RDF level, the given property name can be used on the HTML level as an attribute value of the `getProperty` tag. For example, the tag

`<getProperty name="Resource_location"/>`

expands into the name of the location at which the photograph was taken.

## 3.3 Links

In *SWeHG* a *link* is formally defined by the pair $L = \langle Name, Rel \rangle$, where $Name$ : is the name of the link (a string) and $Rel = R \times R$ is a binary relation in the set of resources $R$ in use. Given a context resource *c* and a link $\langle N, S \rangle$, the set $\{\langle N, x \rangle \mid \langle c, x \rangle \in S\}$ of URIs is called the *link group* $N$ of *c*. Intuitively, a link group defines a set of named associations from a resource to other resources. Such associations can be rendered on the HTML level as labeled HTML links.

A link (or relation) rule is defined with the ternary predicate:

```
swehg_relation_rule(
  TagAttributeName,
  HTMLLabel,
  PredicateName).
```

The predicate defines a link name to be used in the HTML tags (TagAttributeName) and associates it with a label to be used as the name of the link in the rendered HTML page (HTMLLabel), and with the name (PredicateName) of some freely definable binary predicate $p(c, t)$. This predicate should succeed when the context resource *c* has the link to the target *t*.

For example, the predicate

```
swehg_relation_rule(
  'SameLocation',
  'Photos from the Same Place',
   photosWithSameLocation).
```

ties the tag name `SameLocation` with the HTML link label and the predicate that defines the link relation. The `photosWithSameLocation` predicate can be defined as follows:

```
photosWithSameLocation(Context, Target) :-
    photo(Context),
    photo(Target),
    rdf(Context, _:place, Location),
    rdf(Target, _:place, Location),
    not(Context == Target).
```

When the relation rule is applied with respect to the whole resource set defined by the selector `photo`, the result is a group of links that can be rendered on the HTML page. We can point out the specific spot of the links on the HTML page with the `<getLinks>` tag. For example, the tag

```
<swehg:getLinks name="SameLocation"
   listType="ul"
   listStyle="text-color: red; text-size: 10;"/>
```

on the HTML template for photo RPages expands into an unordered list (ul), where each list item is a link pointing to another RPage that corresponds to a photograph taken in the same location. The resulting HTML code is given below (slightly abbreviated):

```
<ul style="text-color: red; text-size: 10;">
  <li><a href="entry.Mediacard_00071.html">
     View from Eiffel-tower</a></li>
  <li><a href="entry.Mediacard_00143.html">
     Cafe Parisienne</a></li>
  ...
</ul>
```

Notice that the logical definition of links in this example is simple but can in principle be arbitrarily complex. This approach provides a very powerful means for linking different kinds of resources on the RDF level. Selectors can be used to bring out logical groups of resources and predicates be used to define the semantic linkage between the group members. The linkage is rendered on the HTML level by the `<getLinks>` tag.

In order to get human readable link labels for individual RPages, a predicate `swehg_label(URI, Label)` has to be defined. This predicate is called always when links are generated. The labels used can, for example, be simply a `rdfs:label` property connected to the resource at hand.

## 3.4 Views

A *view* is a hierarchical index-like decomposition of category resources where each category is associated with a set of subcategories and additional individuals of the categories. A view is defined in an HTML template by specifying 1) the root resource selector, 2) a binary subcategory relation predicate, and 3) a binary relation predicate that maps the hierarchy categories with the individuals used as leaves in the view. A view is rendered on the HTML level with the tag `<swehg:getView>` that has the compulsory attributes `selector`, `branches`, and `leaves`, respectively. For example, the tag

```
<swehg:getView
   selector="buildings"
   branches="subclass"
   leaves="photoOf"
   listType="ul" target="dataPane"/>
```

expands in any context into a hierarchical unordered list (ul) of categories. The predicate definitions could be following ones:

```
Building = 'http://example.org/meta#building'.
buildings(URI) :-
  rdf(URI, rdf:type, Building).

subclass(SuperCategory, SubCategory) :-
  rdf(SubCategory, rdfs:subClassOf, SuperCategory).

photoOf(Class, Record) :-
  rdf(Instance, rdf:type, Class),
  rdf(Record, dc:subject, Instance).
```

Here the selector selects the class `building` as the root of the view. The view hierarchy is expanded along the `rdfs:subClassOf` property that is used in RDFS for representing class hierarchies. The `photoOf` predicate relates each leaf $c$ of this class tree with a set of photo record resources. They are used as the leaf categories of $c$ in the final view rendered on the HTML page. Given the above specifications, a view hierarchy represented by hierarchical unordered lists (ul) is generated in HTML. When the end-user clicks a link, the corresponding RPage is shown. The frame in which the linked page is rendered can be selected by the optional the attribute `target`.

The definition of a view in terms of a selector, a branching relation, and a leaf relation yields in general all permutations of view trees. Only the first one found is used as the view. What view is obtained first can be controlled by defining the predicates carefully. The final structure of the view can also be controlled with the help of an additional tag attribute `orderBy` whose value is an ordering criterion defined as a binary predicate. By using this attribute, the categories in the view can be listed in alphabetical or in any user-defined order.

For another example, assume that there is a partOf meronymy of location concepts (e.g., `Paris _:partOf France` in the knowledge base. We would like to render the meronymy as an index from the root class `place` by using the `getView` tag in the following way:

```
<swehg:getView
  selector="place"
  branches="hasPart"
  leaves="photoOfLocation"
  listType="ul" target="dataPane"
/>
```

If a photograph record is annotated using the property arc `_:place` from the photo data record to a location resource, then the predicates in the tag can be defined in the following way:

```
hasPart(SuperCategory, SubCategory) :-
   rdf(SubCategory, _:partOf, SuperCategory).

photoOfLocation(Location, DataRecord) :-
   photo(DataRecord),
   rdf(DataRecord, _:place, Location).
```

Based on these definitions, it is possible to generate a location index hierarchy for any location concept (root) and link the photo RPages related to each index class entry.

# 4. GENERATION PROCESS

## 4.1 Generation Procedure

The process for transforming an RDF(S) repository into an HTML page repository is defined by the algorithms 1 and 2. The input of the procedure is 1) a set of HTML templates, 2) an RDF(S) repository, and 3) the logical rules for selectors, properties, links, and views. The output is an HTML page repository conforming to the templates.

The process is based on generating pages using the HTML templates one after another. If a template is associated with a selector, then it is expanded into a set of RPage HTML pages corresponding to the selected context resources, else it is expanded once without reference to context resources. In the latter case, the HPage and IPages are created. When generating an HTML page, the tags are expanded into HTML by the procedure $L$ in the ways described in the previous section.

```
Data: Templates T, RDF(S) repository R
HTMLPageRepository H = empty;
foreach Template t in T do
    if T has a selector rule S then
        foreach RDF Resource r in R do
            if S(r) == true then
                h = createHTMLpage(r, t);
                add h to H;
            end
        end
    end
    else
        h = createHTMLpage(T);
        add h to H;
    end
end
```

**Algorithm 1:** Main procedure for the RDF2HTML transformation

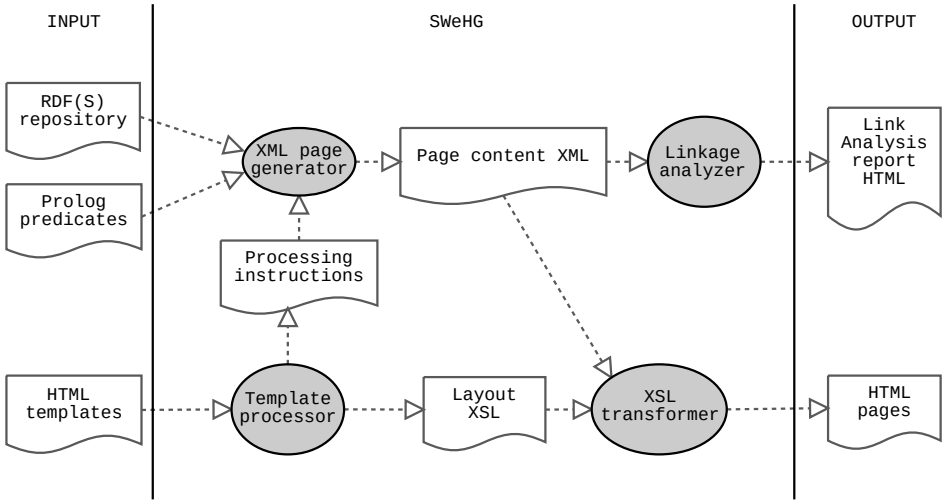


**Figure 2: Internal architecture of *SWeHG*.**

Algorithm: createHTMLpage
**Data**: Template t, Context Resource r
**Result**: HTML page
HTML H = ";
**foreach** *Tag in t* **do**
  **switch** *type of Tag* **do**
    **case** *getProperty*
      h = executeRule(getProperty, r)
    **case** *getLinks*
      h = executeRule(getLinks, r)
    **case** *getView*
      h = executeRule(getView)
  **end**
  H = H + h;
**end**
return H;

**Algorithm 2:** Algorithm createHTMLpage for rendering an HTML template

An example of an IPage template is given below:

```
<swehg:templateFor language="fi"/>
<html>
<body>
<h1>Building index</h1>
  <swehg:getView
    selector="buildings"
    branches="subclass"
    leaves="photoOf"
    orderby="order_alphabetically"
    listtype="ul"/>
</body>
</html>
```

The following template for RPages could be used for rendering the images using the HTML img-tag and links to related RPages:

```
<swehg:templateFor selector="photo"
                   language="fi"/>
<html>
<body>
  <h2><swehg:getProperty name="Title_Of_Photo"/></h2>
  <p><img src="<swehg:getProperty
               name="PhotoURL"/>" /></p>
  <h3>Photos from the same place:</h3>
  <swehg:getLinks predicate="sameLocation"
                  listtype="ul"/>
</body>
</html>
```

## 4.2 Analysis of Results

The layout of the pages in the HTML page repository can be checked fairly easily by browsing the pages. A more difficult question is whether the generated linkage between the pages matches the designer's expectations. It would be difficult and time consuming to verify this by just browsing the pages one by one because the amount of pages generated can be quite large. Furthermore, pages that have no links pointing to them would never be found in this way.

To support the linkage analysis, *SWeHG* generates as a side effect of the transformation process an additional analysis report in HTML. The analysis points out linkage problems such as pages linked to no other pages. By reviewing the analysis, the user to determine more easily whether the templates and the underlying predicates are working correctly.

## 4.3 Prototype Implementation

We have implemented a prototype tool *SWeHG* for transforming RDF repositories into HTML pages. The system implements the template tags (properties, links, and views), predicate definition facilities, the page generator, and an site analyzer as described in the previous sections.

*SWeHG* automatically generates a frameset as the HPage. This page has on the left a narrow frame for each IPage template. On the remaining space on the right, there is a larger view area frame for the RPages.

Figure 3 illustrates the HPage of an application with two IPages depicting an alphabetical list of photographs ("Aakkostettu hakemisto") and a classified index ("Luokiteltu hakemisto"). On the RPage on the left, a photograph is seen with some metadata and recommended links to other photos.

Figure 2 depicts the architecture of our implementation. The

main program is a Perl script which first builds an XSLT template out of the HTML template using the module "Template processor". This module also writes out a set of "processing instructions" into a separate Prolog source code file. These instructions link template tags with the Prolog predicates used in them as attribute values. The module "XML page generator" is a Prolog program that applies the predicates used in the HTML tags with respect the RDF(S) repository according to the processing instructions. The result is a set of XML files describing the page contents. These XML files are then transformed using Apache Xalan and with the help of the XSLT templates generated earlier into the final HTML pages. The intermediate XML files are also used as a basis for the "Linkage analyzer" module that tries to identify unlinked pages, empty link groups and other anomalies. The analysis results are represented in HTML.

# 5. TEST CASE: MUSEUM PHOTO ARCHIVE

A couple of HTML repositories have been generated to test and evaluate the usability of the RDF2HTML transformation method presented in this paper and its implementation *SWeHG*. This section presents one of our test cases, the generation of a virtual exhibition of the Helsinki University Museum[7] photo archive.

The archive contains 629 photographs about the promotion ceremonies of the University of Helsinki. The content of the archive was transformed into RDF(S) format in an other application project [6] and was used as it is by *SWeHG*. The domain knowledge consists of six ontologies with 329 promotion-related concept classes, such as "Person" and "Building", 125 properties, and 2890 instances (individuals), such as "Linus Torvalds" and the "Entrance of Cathedral of Helsinki".

In the photo annotation schema, the subject of a photograph is represented by a collection of ontology classes and individuals that appear on the image[8]. For example, if Linus Torvalds appears in a photo on a particular street, then the photo record is related *directly* with the corresponding person and street instances with a property corresponding to `dc:subject`. However, the relation between photos and subjects can be *indirect*, as well, involving traversal through several RDF arcs in the underlying knowledge base. For example, Linus Torwalds is present in a photograph as a Honorary Doctor. Then only an instance of such a role is associated with the image and the person instance in not directly linked with the image but with the role instance. *SWeHG* predicate definition facility is very handy in hiding such annotation schema specific details from the HTML designer: the persons can be associated with images either directly or indirectly through roles or by other mechanism by defining a single predicate for the relation.

By publishing the archive as a web site generated by *SWeHG* provides the end-users two with two services. Firstly, the photos can be easily indexed along different orthogonal views based on the ontologies. The view indices provide an overview of the repository contents and can be used for finding photos of interest. Secondly, the hidden semantic associations between the photos can be made made explicit to the user and can be used as the basis for browsing the photos.

Figure 3 presents a browsing view to the photo exhibition generated for the Helsinki University Museum. On the left, two frames containing index views are seen: an alphabetical index and a classified index based on concepts and subconcepts. In the frame on the right, a selected photo with recommended links to related photos is rendered. In this case, links to photos about the previous and later happenings in the promotion sequence are seen.

Figure 4 depicts a portion of the result from the analyzer. On this page the number of in-coming and out-going links can be seen for each RPage together with a status explanation. The analyzer has found out that with the predicate definitions used the page with label "Aikaisempien yleisten ..." is not connected with any other page or index. Furthermore, the page "Airueet" has one incoming and two outgoing links but was not included in any index. This kind of connectivity information is vital when debugging the system.

# 6. DISCUSSION

## 6.1 Benefits Obtained

This paper argued that there should be a publication method for content on the Semantic Web that is independent from semantic portal providers. Such a method could be created based on the customary "HTML in a public directory" approach, but requires that the content in semantic web formats, such as RDF(S), can be transformed easily into a set of HTML pages. To approach the problem, a method and a prototype tool *SWeHG* were developed for transforming RDF(S) repositories automatically into HTML page repositories. The system is based on a tagging mechanism that separates the HTML design from the underlying logical rules based on RDF.

Our initial experiences indicate that the presented approach feasible. HTML templates can be created fairly easily and can be adapted to different RDF repositories. The idea of using logic and Prolog for defining the semantics of the tags seems very powerful. Complicated semantic link relations and views can be defined with a few lines of code.

## 6.2 Static vs. Dynamic Pages

The HTML pages generated by *SWeHG* are static and created in a batch process before publishing them on the web server. This approach has the following benefits when compared to dynamic HTML applications:

- Simple independent publication. Static pages can be published easily on the web.
- Search engines can access and index the content more easily. For example, if a museum collection is published in this way, individual artifact pages can soon be found by Google and other engines.
- Static pages can be served efficiently. Dynamically generated HTML pages may not scale up with large or complex RDF(S) repositories and inference mechanisms. In *SWeHG* complicated inferences and links can be computed beforehand and hence online processing is not needed.
- Security and control. Creating dynamic web applications requires always special attention to security issues. Inputs from user users must be checked, firewalls must be configured when connecting the public web servers to operative internal systems, etc. When dealing with static pages such security difficulties are do not arise.
- It is easy to control what data is published when it is represented as explicit HTML pages.

[7] http://www.helsinki.fi/museo/

[8] The annotations also include other metadata, such as the photographer, free text descriptions and some technical information of the images etc.


**Figure 3: The entry page of a photo exhibition generated with *SWeHG*.**

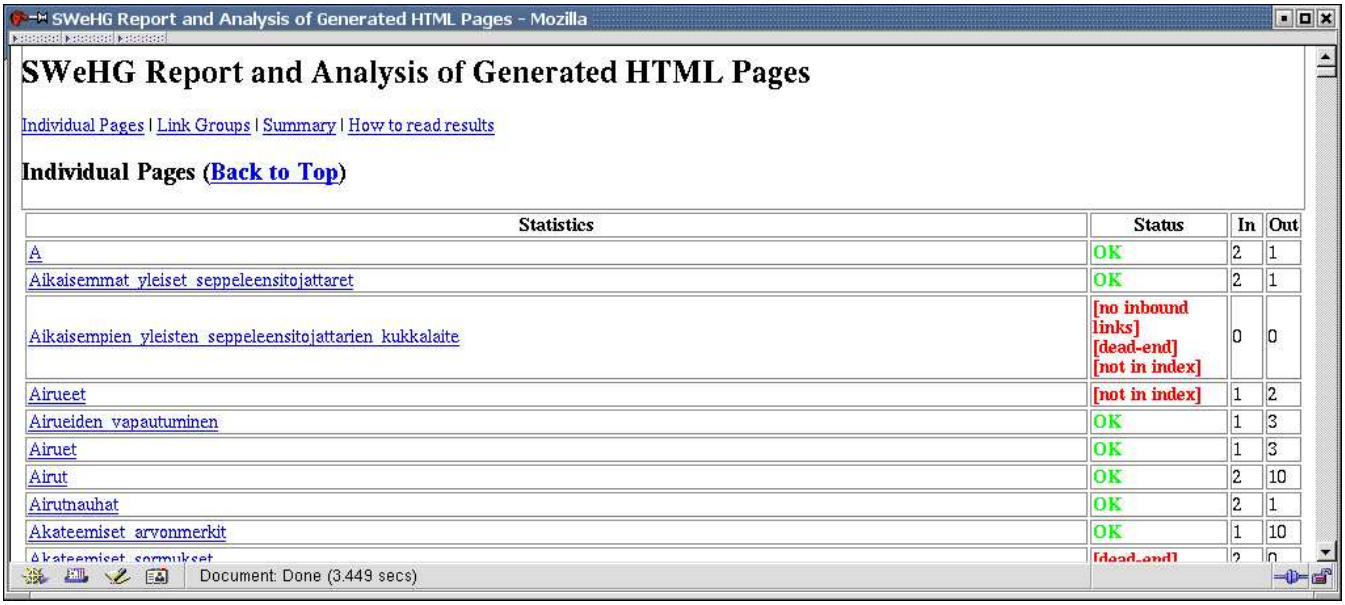


**Figure 4: An analysis page created by *SWeHG*.**

- Properties of the resulting HTML page set can be analyzed and tested easily when they are generated explicitly. The linkage analyzer of *SWeHG* provides an example of this.

On the other hand, the static approach also has its limitations:

- The static pages can not adapt their content dynamically to different users or patterns of usage. Dynamic systems can have, for example, an internal state and remember how they have been used.
- Dynamic systems can be connected more easily with other services providing additional online functionality.
- If the RDF(S) repository, the rules, or the HTML templates change, the site has to be regenerated usually from the scratch. Dynamic systems can adapt better to such changes.
- If the RDF(S) repository is large and many templates are used, then the number of generated pages can be large .

## 6.3 Related work

Our work was inspired by the idea of semantic portals. However, we wanted to create a "poor man's" approach to publish semantic RDF(S) content on the web without dependency with a particular semantic portal service provider and an application.

The idea of using tags in *SWeHG* is to separate the design of the layout from internal implementation issues. This idea is similar in tag languages, such as JSP[9][11], PHP[10], and ASP[11]. However, in our case the semantics of the tags are defined by logical predicates and the underlying RDF(S) data. Furthermore, the tag approach is used for generating static pages instead of dynamic ones.

The idea of linking pages with semantic associations is related to that of Topic Maps [9]. However, in our case the linkage structure is not given by an explicit topic map, but is inferred by logical linking predicates that reveal both explicit and implicit associations in the underlying RDF(S) knowledge base.

## 6.4 Further work

More work and testing is needed in order to evaluate the usability of *SWeHG* in practice and in determining what additional features may be needed in the system. It seems like a handy tool in the hands of the people who created it, but more user experiments with predicate and template designers that have not been involved in developing *SWeHG* themselves are needed. We plan to apply the system next to the RDF collection repositories created from the databases of several museums in the "MuseumFinland" semantic portal project [5].

## Acknowledgements

Before the prototype described in this paper, an initial version of *SWeHG* was created by a student software engineering project conducted at the Helsinki University, Department of Computer Science during spring 2003. In addition to the authors, the initial programming team included Mikko Kiesilä, Ville Komulainen, Ritva Köppä-Laitinen, and Joonas Muhonen.

## 7. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[2] D. Brickley and R. V. Guha. *Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 2000-03-27*, February 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[3] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors. *Weaving the Semantic Web*. The MIT Press, 2002.

[4] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, , and K.-P. Lee. Finding the flow in web site search. *CACM*, 45(9):42–49, 2002.

[5] E. Hyvönen, M. Junnila, S. Kettula, S. Saarela, M. Salminen, A. Valo, and K. Viljanen. Publishing collections in the Finnish Museums on the Semantic Web protal – first results. In *Proceedings of the XML Finland 2003 conference. Kuopio, Finland*, 2003.

[6] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: combining view- and ontology-based search with semantic browsing. In *Proceedings of the XML Finland 2003 conference. Kuopio, Finland*, 2003.

[7] E. Hyvönen, A. Styrman, and S. Saarela. Ontology-based image retrieval. Number 2002-03 in HIIT Publications, pages 15–27. Helsinki Institute for Information Technology (HIIT), Helsinki, Finland, 2002. http://www.hiit.fi.

[8] O. Lassila and R. R. Swick (editors). Resource description framework (RDF): Model and syntax specification. Technical report, W3C, February 1999. W3C Recommendation 1999-02-22, http://www.w3.org/TR/REC-rdf-syntax/.

[9] Steve Pepper. The TAO of Topic Maps. In *Proceedings of XML Europe 2000, Paris, France*, 2000. http://www.ontopia.net/topicmaps/materials/rdf.html.

[10] A. S. Pollitt. The key role of classification and indexing in view-based searching. Technical report, University of Huddersfield, UK, 1998. http://www.ifla.org/IV/ifla63/63polst.pdf.

[11] G. Shachor, A. Chase, and Magnus Rydin. *JSP Tag Libraries*. Manning Publications Co., 2001.

[9] http://java.sun.com/products/jsp/

[10] http://www.php.net

[11] http://msdn.microsoft.com/asp/