SMART - AN APPROACH FOR INFORMATION SYSTEMS DEVELOPMENT ON THE BASIS OF RDF-TECHNOLOGY

ANDREY GUSKOV; VLADIMIR SHRAIBMAN; YURII MOLORODOV

Institute of Computational Technologies,
Siberian Brunch of the Russian Academy of Sciences

This paper introduces a new technology of information systems creation based on the declarative descriptions of document collections in RDF. This approach is intended to improve the quality of information published on Web and the interaction between various web-applications. In this framework a model of the document generation is presented, a pictorial example is provided. The benefits and disadvantages of this technology in comparison with traditional XML-based approaches are discussed.

Keywords: information system, publishing system, web application, document, collection, XML, RDF, Semantic Web

1. INTRODUCTION

Fast development of modern computer technologies has resulted in a rich variety of public-access information systems (IS) on the World Wide Web. These systems may contain substantial or, in general, unlimited number of interconnected documents

$$IS = \langle \{d_i\}_{i \in X} \rangle = \langle D \rangle,$$

where *D* is a set of documents d_i. The era of the systems, in which the documents are static, is probably coming to an end - those systems become unmanageable as the number of documents increases. Such systems will either collapse due to the lack of support or adopt a dynamical approach to the document formation. To this end, we shall only consider the systems with the dynamical formation of documents, i.e. the systems, in which documents are created right after a submission of request.

Another important feature associated with these systems, is a possibility of their parameterization. Clearly, traditional information systems allow grouping of documents on the basis of some common logical property. Such grouping is not always unique and depends on the specific objective, however it always exists. Typical examples include a group of documents providing information about users, or a group of documents containing articles description, such as their titles, authors and abstracts. Thus, we arrive at the concept of *collections* - which are sets of documents revealing the common structure and describing the same entity.

Let us define a *collection* C_γ as the set

$$C_\gamma = \{d \in D : \exists \alpha \in A^n, g_\gamma(\alpha) = d\}, \gamma \in \Gamma.$$

Here $g_\gamma : A^n \rightarrow D$ is a function, which maps parameter into document T, is a finite set of the collections of IS. α denotes n-tuple vector of parameters that are required to produce requested

document. There are no assumptions made about the set of parameter values *A*. Generally it could contain boolean, integer, string and values of the other types.

Now let us define a *collectional information system* IS_c as the system which can be divided into the collection:

$$IS_c = \langle \Gamma, \{g_\gamma\}_{\gamma\in\Gamma}, A^n \rangle \sim \langle D \rangle,$$

or, in other words,

$$D \subseteq \bigcup_{\gamma\in\Gamma} \{g_\gamma(\alpha) : \alpha \in A^n\}.$$

Before continuing, some definitions have to be made. First of all we should specify the term *content* as a reflection of some information. This definition is quite abstract, it does not assume any assertion about what the content really is. The systems that implement this model, could use their own internal representation of the content, one of such representations is considered in this paper. Second, let us define a character sequence *document* as some image of the content. The *content function* is the function $h : A^n \to C$, which maps the parameters vector into the content. And, finally, *style* is the function $s : C \to D$, which maps the content into the document.

Information systems are intended to provide some information (content). However, the format of this information, in general, depends on the type of the recipient. For the ordinary Internet user the documents should be provided in HTML-format, but also, if one desires, the same documents could be sent via e-mail as a plain text. For Palmtop users, the same information must be provided in WAP-format. If some service or intellectual web agent makes a request, he must obtain document in a very specific, usually XML-based format. Thus modern information system must be able to represented the same content in the distinct formats.

The *multistyle information system* is

$$IS_s = \langle \Gamma, \{h_\gamma\}_{\gamma\in\Gamma}, \{s_{\gamma,\phi}\}_{\gamma\in\Gamma,\phi\in\Phi}, \Phi, A^n \rangle.$$

Here Φ is a finite set of possible document formats, and $s_{\gamma,\phi}$ is a style function. Thus, for multistyle information system, we need to perform transformation $s_{\gamma,\phi}\circ h_\gamma(\alpha)$ to get the requested document from $\gamma \in \Gamma$ collection for $\alpha \in A$ parameters in ϕ format. Generally speaking, α can contain μ and, even, $\varnothing$ parameters, but this is meaningless.

Let us call the aforementioned function $G_{\gamma,\phi}(\alpha) = s_{\gamma,\phi}\circ h_\gamma(\alpha)$ the *document-forming function*. In the majority of information systems, the requested document is generated by some server-side program, which, actually, is a document-forming function $\boldsymbol{G}_{\mu,\varnothing}$. Such an approach is the simplest one and it is widely used when the information system is designed from scratch. However, it also has a number of defects. Firstly, the major part of the development time for such program is spent on GUI design, transfer and processing of parameters, and various queries to databases. It is inevitable but a very time consuming process and therefore should be optimized. Secondly, the logical structure and the design of documents are usually interconnected in quite an inflexible manner.

In other words, the *content* and *style* function cannot be extracted from the document-forming function $\boldsymbol{G}_{\mu,\varnothing}$. Such information systems, which use document-forming function we call *traditional*:

$$IS_t = \langle \Gamma, \{G_{\gamma,\phi}\}_{\gamma\in\Gamma,\phi\in\Phi}, \Phi, A^n \rangle.$$

Obviously, the class of *traditional information system* is equal to the class *collectional information system* $IS_t = IS_c$. Indeed, we need extended functions set $\{g_\gamma\}$ $|\Phi|$ times to get $IS_t \subset IS_c$, and take $|\Phi| = 1$ and $G_{\gamma,\phi} = g_\gamma$ to get $IS_t \supset IS_c$.

As a result, we have the following relationships for informational system classes:

$$IS_s \subset IS_t = IS_c \subset IS.$$

In this paper, the technology of *multistyle information systems* developing is considered, which is based on the representation of the *content* in the format of *Resource Description Framework* (RDF) [1]. It was implemented in SMART project [2], which became a foundation for several information systems developed in ICT SB RAS. Thus let us reveal the advantages and disadvantages of this technology that will be summarized in the end of the paper.

2. THE MODEL OF DOCUMENT FORMING

First of all, we should specify how the *content* of documents will be represented. As it was defined before, the content is the reflection of the information. The information on the web could be considered a set of facts. A simple fact indicates a characteristics of the thing, e.g. it indicates that a thing has a named property that has a value. In other words, some simple facts indicate a relationship between two things [3]. Such fact can be represented as a triple (*subject*, *property*, *object*), where *subject* and *object* denote the things and *property* declares the relation between them. This concept has been put into the basis of *Resource Description Framework* [1] — the language for resources description. It does not specify what kind of resources can be described and it provides various tools for the RDF-documents processing. Thus, we believe that at the moment RDF is the most suitable format for the content representation.

The next thing we should specify is what the *content function* h^a really is. As we have just defined, the range of this function is RDF-documents. For the traditional IS developing the content function must be some program written on the imperative language (Perl, PHP, Java). In our case this approach leads us to the increase in the development time, because the same programming tools are still used and, moreover, we added new developmental stage after separating content programming from design (style) programming.

As it has already been noticed, most of the scripts perform the same set of operations when generating the document: processing parameters, inquiring database and results substitution, checking some conditions and so on. Besides, documents generated by the same script have identical or almost identical structure. Recall that such document set is called *collection* in our terms. According to this argumentation, we have come to the conception of the declarative (instead of imperative) description of the documents, when we indicate **what** the content is, but not **how** it is constructed.

Let us introduce some notions: a set of *transformers* $Tr_1, \ldots, Tr_k;\ Tr_i : A^n \times C \to C$ and a set of *collection sources* $\{src_\gamma\}_{\gamma \in \Gamma}; src_\gamma : A^n \to C$ for multistyle information system, if $\forall \gamma \in \Gamma\ \exists i_1, \ldots, i_\gamma : h_\gamma(\alpha) = Tr_{i_\gamma}(\alpha, Tr_{i_{\gamma-1}}(\alpha, \ldots Tr_{i_1}(\alpha, src_\gamma(\alpha)) \ldots))\ \forall \alpha \in A$.
It is easy to see that we can always specify the set of transformers, which consists of the content function $h^{\bar{a}}$. But later we will see that the proper selection of a transformer set can greatly enhance the efficiency of the IS development.

Thus, we can consider collection $\gamma \in \Gamma$ as a tuple $(src_\gamma, \{Tr_{i_k}\}_{k=1..\gamma}, \{s_{\gamma,\phi}\}_{\phi \in \Phi})$, where src_γ describes the structure of the content for all the documents from ã, the set of transformers specifies how the content of the concrete document must be obtained and the set of styles — format this collection is supported for every document — provides the style for human-readable or any other representation of the content.

Now we can introduce our technology of IS development named **SMART** (System for Managing Applications based on RDF Technology [2]), which implements these theoretical considerations. The main aspect of this technology is the process of document forming. To obtain the requested document, the SMART-server must get the required *collection source*, which actually is some RDF-model[1] and holds the structure description for the content of the documents from this collection.

Retrieving a document from a collection is carried out in two stages (Figure 1).

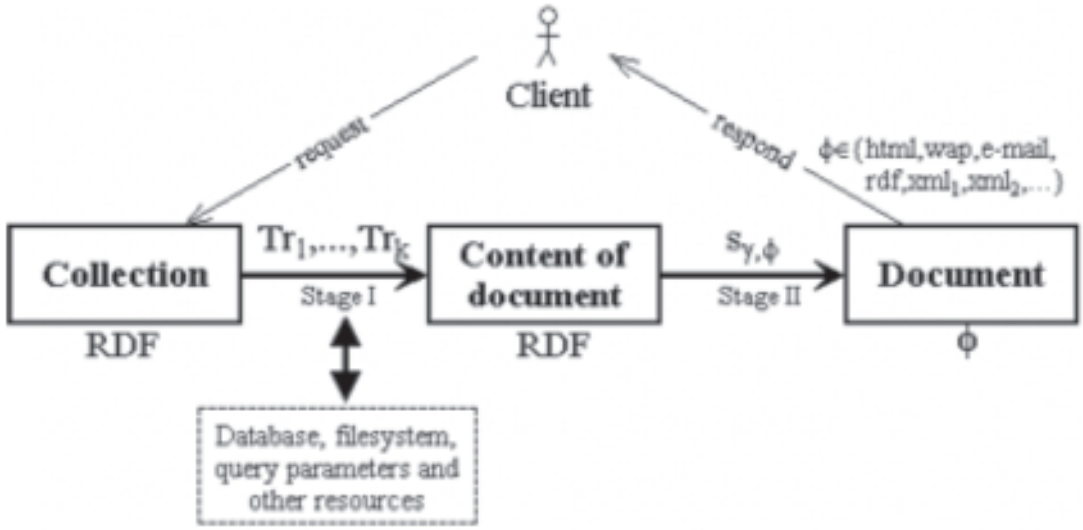

FIGURE 1: THE SCHEMA OF THE DOCUMENT FORMING

[1]The RDF-model is a syntax-neutral way of representing RDF documents [1]

At the first stage SMART-server carries out a number of standard transformations $\{Tr^{ik}\}^{k=1..\tilde{a}}$, which may include, for example, request parameters substitution and extraction of results from the data store, etc. Thus, after the first stage, the RDF-description of the document content in a pure

form is obtained. So it contains the entire data and its logic for the requested document that is written in RDF. The document's design is added at the second stage when the style template $s_{\gamma,\phi}$ is applied to the obtained RDF-description. In fact, the style template is just a XSL-Transformation stylesheet, which is applied to the XML-serialization of the RDF-model. Such a method was employed due to the preference of the web application to the XML-based formats for data exchanging (especially human readable HTML).

3. A PICTORIAL EXAMPLE

Finally, our approach results in the successful development of SMART project. Project description can be found at http://web.ict.nsc.ru/smart. SMART is a library that provides tools for a development of various information systems based on the aforementioned approach. It is intended for use in the framework of Java Web Container technology [4] and consists of a set of Java classes. The core of the library could be divided into three blocks according to the functions they perform:

1. **Model reader.** When a client submits a request for a document, the system reads the collection the document belongs to. This module will load and join together all the parts of the appropriate collection (which may consist of a number of subcollections located in various files and/ or Internet sites). Then the description of the collection will be parsed by a RDF-processor, which creates a RDF-model that corresponds the required collection. In fact, this module implements the *collection sources* function src^{a} in the terms declared above.

2. **Model transformer.** A model transformer is defined as a logically completed part of certain conversion of RDF-models. A sequence of transformers, according to the client's request parameters, forms the content of the requested document from the RDF-description of the collection. A transformer is a Java-class, which implements interface *ModelTransformer*. A specific sequence of transformers corresponds to each collection. "Collections-Transformers" relation is declared in the information system configuration files. Note that this kind of architecture allows users to add their own specialized transformers, which dramatically extends the system applicability. It is easy to see that model transformers are the implementations of the *transformers* function Tr^{i}. Usually a transformer finds a certain submodel, analyzes its content,

and transforms it according to the request parameters and the submodel information. Some examples of transformers provided by the standard edition of SMART are as follows:

- *Parameters provider*. Transformer looks throughout the model and replaces every special combination of symbols with the value of parameters transferred by a client query. For example, the combination *${Value name = "id" default = "0"}* will be replaced with the value of parameter *id* if it was transferred in the query, or value *0* otherwise.

- *Conditions processor*. In the model it finds a statement so that its object contains some logical expression condition. If the value of the expression is *false*, a submodel created by the subject of the conditional statement is deleted from the model (in a case of tree - a subtree which root is the subject).

- *Database data retriever*. Due to a specifics of information systems, this transformer is the most important one. Its aim is to search a submodel, in which data are to be inserted; to create and execute SQL-query and then to fill the model with the selected facts. The sequence of queries, the priority of processing of the multiple selections, and query customization are important, because this allows us to use in query the results of the performed ones and to embed submodel obtained by the query multiple times into the main model. All the technical information about the query execution is included into the source collection description. Thus, taking into account standard Java interfaces,

this transformer does not depend on the type of the DBMS used. The specifics of the datastore used is declared in the application settings and in the descriptions of the collections. The example of the execution of database transformer is shown in the pictures. Figure 2 represents a RDF-model before the transformer execution.

FIGURE 2: SOURCE RDF-MODEL OF A SAMPLE COLLECTION

The documents of this collection are intended to list all the conferences stored in a database and selected with respect to some parameters, e.g. the year of holding (for the sake of simplicity, all parameters were omitted). The subgraph, being included in the final content, is typed boldface; the rest is a temporary information, which is necessary for transformer execution, it will be removed after that. There are two queries on this figure — *ConferenceQuery* and *SectionQuery*, and the latter depends on the former due to the *smart: DependsOnQueries* property. The model resulting from the first query is shown in Figure 3.

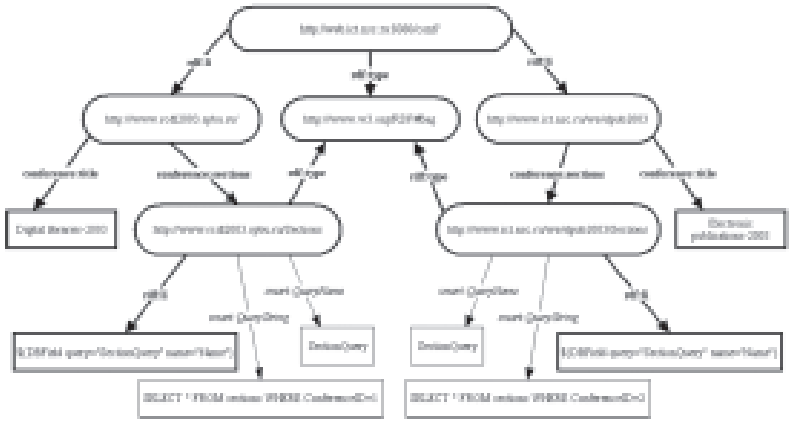

FIGURE 3: INTERMEDIATE RDF-MODEL AFTER THE FIRST DATA SELECTION

There were two conferences found in the database, therefore two copies of the result submodel were created. After that the second query can be executed. Actually, there could be a number of the queries with the same name after replication. But there is no conflict between them, they can be

executed in any order. The final RDF-model representing the content of the document is shown in Figure 4.

- **Model formatter.** When all the content transformations are done, we get the RDF-description of the document's semantics. First of all, various intellectual programs collecting and processing the information can use such a description just being serialized without any additional changes, because RDF becomes the most popular format for knowledge exchange. To implement the style function s^{a} for other XML-based formats, including HTML, we used XSLT [6] technology.

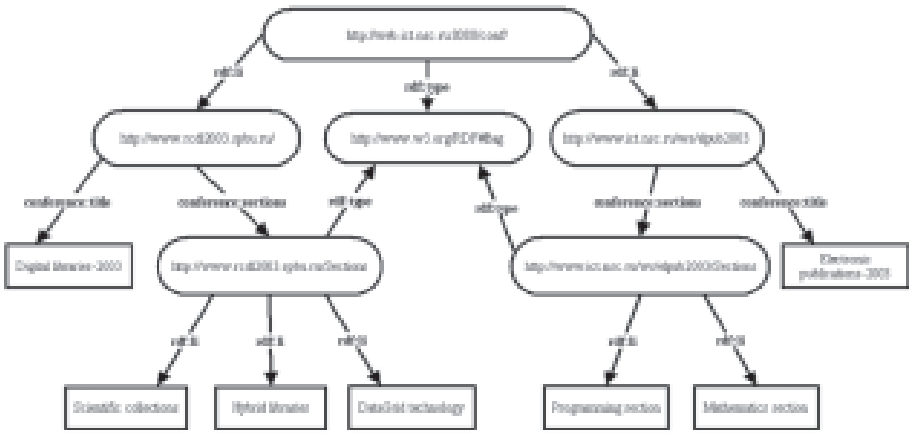

FIGURE 4: FINAL RDF-MODEL REPRESENTING THE CONTENT OF THE DOCUMENT

Notably, that is is designed for XML to XML conversions, whereas we need a RDF into XML stylesheet transformation, so using XSLT for our purpose can cause some technical difficulties. With some minor reservations, these difficulties were successfully overcome. However, in future we intend to develop a specialized tool for the stylesheet transformation of the RDF documents into the XML documents. The result of the XSL transformation of the content obtained is shown in Figure 5.

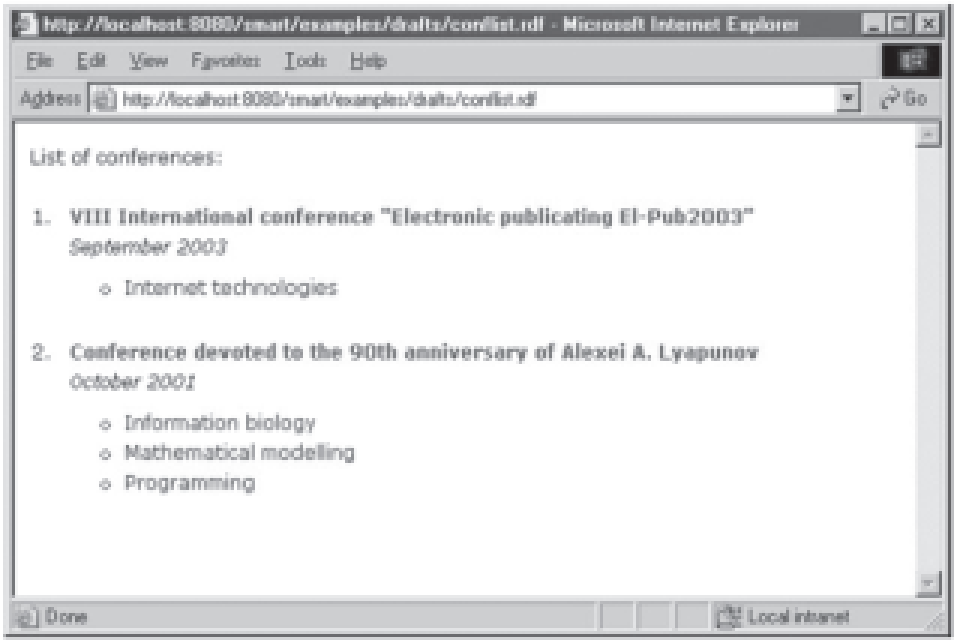

FIGURE 5: FINAL REPRESENTATION OF THE DOCUMENT IN HTML-FORMAT

4. THE ANALYSIS OF THE RESULTS

In order to analyze the presented technology, we need to investigate the achievements of the existing technologies and compare them with our solution.

4.1 CONTENT MANAGEMENT SYSTEM OR PUBLISHING SYSTEM?

There is a widespread misleading term *content management system*, which is applied to all the systems oriented to publishing documents on the web. So first of all, we should classify similar web-development technologies to determine its application range.

In essence, there are two main classes of technologies for web-publishing: *content management systems* and *publishing systems*. A number of the former systems is listed in [7] and [8], the latter ones — in [9]. A general description and main features of the both classes are shown in Table 1.

TABLE 1: COMPARATIVE CHARACTERISTICS FOR *CONTENT MANAGEMENT SYSTEMS* AND *PUBLISHING SYSTEMS*.

Content Management Systems (CMS)	Publishing systems (PS)
CMSs define a *content* as a set of resources, such as text, pictures, clips. Content of documents is created and edited manually.	PSs use the aforementioned notion of the *collection*. The operations on the document are always done in the context of the collection.
CMS is oriented to the site building, basically, static. General features of CMS stressed the visual representation of the site, the simplicity of the resources management and the tools for site development. It is impossible to implement database interaction properly.	PS is oriented to the informational system development, in which most of the data is stored in some external data-stores, mainly databases.
Mostly, CMS is a hybrid, which consists of a textual editor and processor, resources manager and a web-server. It has pretty-looking user-friendly interface and it is compatible with such widely used software, as Microsoft Office. An advanced user skills are not necessary.	PS are usually written in a script language (PHP, ASP, Java, Python) and distributed as a library, which could be used in conjunction with a popular web-server (Apache, IIS). The collections descriptions implementation is typically based on the XML technology. These descriptions contain instructions about how the final document must be obtained. A user is assumed to have high developer skills to create or modify collections.
In CMS, the context-search function is easy to implement, because all the resources of site are stored in explicit form. The high-level search is beyond the scope of such systems.	The implementation of the context search is almost impossible, whereas there are no complete documents available. Instead, a functionality of the data-store search is recommended, which has a number of benefits in comparison with context search.

As may be seen, there are fundamental distinctions between CMS and PS systems. The content management system is intended for producing sites as a set of manually created page. CMS provides some user-friendly functions for forming and stylizing site documents, but there are no tools for data sets processing. Thus, the site produced by CMS cannot be called an *information system*, because it does not process information, but just a text.

The SMART-system should be related to the publishing systems, that are used to represent a large data sets onto the Web. The main notion of PS is a *document template*, or a *collection*, in our terms. Basically, the process of the document generation in publishing systems is similar to the one described above. Fundamental distinction between such systems is in the way of the collections description and the functionality this descriptions can provide.

In general, publishing systems can be categorized into two classes: *style-supporting* and *HTML-oriented*. In HTML-oriented publishing systems the document template is just HTML document, which includes some generating instructions. After a user requests a document, the system reads the template, processes the instructions and inserts the relevant data. Thus, the content and the representation of the document are merged, neither the former nor the latter are available. Therefore, there is no representation in another format, except the source one, is possible. In our terms, such systems may be called the *traditional information systems*.

For the style-supporting publishing systems a document template description is typically represented in XML-like format. This description, generally, is not related to the resulting format; it implements the *content function*, which was defined above. The documents generation process is similar to the process described in previous chapter — at the first stage the relevant data is inserted into a template and at the second stage a style function is applied, which is, typically, XSLT transformation. Thus, SMART-system can be related to style-supporting publishing systems, which are intended for multistyle information systems creation.

This "multistyle collection" approach has a number of advantages. Firstly, it allows us to develop information systems at a qualitatively new level, by describing both logical structure and inter-relations of the collections of documents without additional writing of the routine codes. Secondly, such description of documents obviously separates a content of a document from its design. This enables programmers and designers to work independently, and also allows them to reuse already created code and style libraries.

4.2 INTEGRATING WITH THE SEMANTIC WEB

Now let us compare the presented SMART technology with other existing publishing systems. As it has been already said, the main feature of SMART is using of RDF language to describe the content of documents, whereas in other systems pure XML format is applied.

Undoubtedly, XML has a number of advantages. However, it is well-suitable for mark-up data transferring, but does not provide a proper way to explain the semantics of the data. Therefore, XML-based publishing systems (and not only XML-based) provide only a syntax for collections content description and do not handle the semantics. This not only forces the developer to invent the semantics scheme to be used (although XML is a perfect tool for such tasks). In most cases the content, which was resolved for that schemes, may be useful only inside this information system. To set some interaction with other information system, the format convertors would be required.

Unlike XML, RDF is the next level of web data representation and interchange. RDF documents themselves do not have great power, and sometimes it is less than evident why one should bother to map an application in RDF. The answer is that we expect these data, while limited and simple within an application, to be combined, later, with data from other applications into Web. Applications in the Web need to use a common framework to collect and process information from other applications [11].

The Semantic Web activity [12] provides such a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation of many researchers and industrial partners. The Semantic Web is an extension of the current Web, in which information is given well-defined meaning, enabling computers and people to work in a better cooperation. In order to make that possible, firstly, it will be necessary for communities to expose their data so that a program will not have to strip the formatting, pictures and ads from a web-page to guess at the relevant bits of information. Secondly, it will allow people to write (or generate) files, which explain — to a machine — the relationship between different sets of data [13].

The Resource Description Framework is a fundamental component of the Semantic Web, which integrates a variety of applications using XML for syntax and URIs for naming. Where XML is made up of elements and attributes - which tells you only about how things are written into the file - RDF data is made up of statements where each statement expresses the value of one property of something [14]. Thus, RDF may become the required basis for the content representation of the information systems in future.

The SMART-technology we propose conforms the ideas, which Semantic Web project suggests. The SMART allows us to create information systems, which could easily be integrated into worldwide knowledge framework, better than other XML-based publishing system does.

4.3 QUALITATIVE SEARCH FOR INFORMATION

A pictorial example to the said above is the implementation of the search function. Ordinary publishing system has actually no possibilities to make qualitative content search. There are only two ways: to implement the search function using the database SQL language (common way) or to store all generated documents and scan them for the context search. The first way seems to be much more preferable, but nevertheless it has a number of disadvantages:

- To implement some kind of search through database, like 'search papers by the author', individual SELECT-query must be indicated inside the system. Therefore, information system can provide only several search patterns, which cannot be extended for specific user needs.

- The data stored in database and the information published on Web are quite different and there is no one-to-one mapping between them. Thus, it is always possible to say what document must be in the results list and some documents cannot be found at all.

- SQL language has a number of restrictions, which do not allow one to implement some search types. The simplest example is to find some keywords inside the entire database. To implement such a function one has to write a set of ugly looking queries.

The context search is rarely implemented for a single information system, but there is some remarkable thing about it. The context search is the only possible way to create worldwide search systems, such as Google and Yahoo. The essence of this search is to find texts, which contain given keywords. There is a big number of additional services, like the distance between keywords or synonym substitution. And it is possible to find all the documents where the words 'John', 'Smith' and 'creator' are mentioned, but it is unfeasible to obtain all the documents, which John Smith has created.

The one of the features the Semantic Web project offers is a possibility to create semantic search over entire Web in the future and to make Internet more organized and structured. At present, there exist some common basic technologies, which are intended to integrate web-applications and to make some "intelligent" search and navigation through the web, such as *Dublin Core Metadata Set* [15] and *RDF Site Summary* [16], possible. But, actually, RDF has a big potential, which is to be used.

The further development of the field of global search is provided by the basic component of the Semantic Web, collections of information called ontologies. Ontologies, as a form of knowledge representation, are defined by T.Gruber as a systematic account of existence, a specification of a conceptualization [17]. The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. Ontologies can be used by automated tools to power

advanced services such as more accurate Web search, intelligent software agents and knowledge management.

As for SMART project, it contains some tools for creating RDF-store of the processed documents. Moreover, for information systems, which were created using SMART, some RDF-based semantic search engine over the RDF-store can be implemented. It uses *RDQL* [18], which is an implementation of an SQL-like query language for RDF. RDQL allows one to execute queries to the RDF models, which find resources of the specified type or resources with some property containing some string and much more complex ones. We have made some tests and can conclude that these ideas are quite applicable and noteworthy.

4.4 SOME DISADVANTAGES OF THE RDF-BASED SYSTEMS

During the SMART developing we have encountered some difficulties that are worth being mentioned. The first one is a poor readability of RDF document formats. Unfortunately, it has no such clearness, as XML documents have. To express some content of the average complexity seems a bit more structured and understandable that some binary code. This could be inconvenient for developing and debugging.

To handle this problem, some additional tools must be involved. For example, such tools could contain some user-friendly interface for documents development and could create the template of RDF document, based on the associated ontology. One of the software, intended for the similar purpose is Protégé [19], an ontology and a knowledge-base editor.

Another serious problem is a time of document creation. As everybody knows, the time of the response after the document request is critical for web-applications. Recall that the document generating process consist of two stages: some RDF-transformations executing and XSLT transformation. For documents, which contain a lot of information, this process can take too much time.

TABLE 2: PERFORMANCE TESTS. TIME OF DOCUMENT FORMING IN SECONDS

	Intel P1, 233 MHz RAM 128 Mb			Intel P3, 1000 MHz RAM 256 Mb		
	Stage I	Stage II	Total	Stage I	Stage II	Total
50 stmts, 4 qrs	0.6	0.2	**0.8**	0.1	0.1	**0.2**
100 stmts, 4 qrs	0.8	0.3	**1.1**	0.2	0.2	**0.4**
270 stmts, 18 qrs	3.8	0.7	**4.5**	0.6	0.6	**1.2**
530 stmts, 4 qrs	3.8	0.9	**4.7**	0.7	0.9	**1.6**
650 stmts, 2 qrs	2.2	1.2	**3.4**	0.5	1.2	**1.7**
1000 stmts, 4 qrs	6.8	1.9	**8.5**	1.3	1.5	**2.8**
1300 stmts, 7 qrs	9.2	2.4	**11.6**	1.7	2.1	**3.8**
1800 stmts, 7 qrs	13.1	3.1	**16.2**	2.4	2.8	**5.2**

We have made some performance tests, the results of which are shown in Table 2. The rows of the tables indicate the documents, which were generated by SMART system at two distinct workstations. In the first column some characteristics of documents are listed — the

number of statements in the content and the number of database queries executed during the document formation.

Firstly, we should note that the time of the style application (stage II) only insignificantly depends on the machine resources (about 10% of the time reduction) and linearly depends on the document size. This is explained by the streaming XSLT-processing using *translets* — Java-class compiled from XSL stylesheet. Translets do not seem to require a lot of resources, therefore there are not many things, the improvement of which greatly enhances the performance. In addition, we can conclude that this problem is common for all XML-based multistyle systems. Thus it is not recommended to create very big documents, since the new efficient facilities for style application will eventually appear.

Secondly, the time of the content generation (stage I) is linear dependent on the document size too and also it certainly depends on the queries number. All executed queries have a simple form, e.g. they have no JOIN operations or nested subqueries. After analyzing these tests we have found the following ways to increase the performance at the first stage:

- The documents of the IS should be lightweighted by splitting documents into logical parts. A big amount of various information is not only hard to process, but is also discouraging. Besides, the data queries executing has a notable influence, so the data extracting process also should be optimized.

- The system of caching of the documents should be established. This can dramatically accelerate the productivity in systems with low frequency of data updating. Actually, an effective caching system has to interact with a datastore to show whether some data were modified and whether the corresponding documents should be updated.

- It is clear from the performance table, that hardware configuration is drastically affected (up to 10 times) because the first stage includes a lot of different complex processes like an DBMS querying and various RDF data processing. Thus, installing more powerful hardware combined with the other optimizing methods could decrease the total time of big document generation to entirely acceptable one.

5. CONCLUSIONS

Briefly, the presented technology SMART has a number of benefits in comparison with the other publishing systems, due to the new view on the document representation. This is connected with the creation of framework for web-applications interaction and integration them into unified space of knowledge. Besides, SMART inherited most of the features from the XML-based publishing systems. The revealed disadvantage is mainly related to the poor readability of RDF documents that could be inconvenient during the developing and debugging phases. Another minor feature — performance of the document generating engine — has a number of improving methods, which have already been partly implemented.

From the standpoint of the obtained results we have found some interesting directions for further researches:

- Development of the Semantic Web oriented services seems to be very promising. The firm foundation for such tools is about to be created. We have already realized some basic possibilities of the semantic search and intend to proceed this researches.

- As important aspect of the publishing system we consider an interaction with the datastore. Despite the fact, that this has no direct relation with RDF, we intend to develop this component to create a fully functional publishing system.

- In the framework of previous topic a toolkit for processing of POST-requests, i.e. client requests containing the data of HTML-forms, was also created. These program components were implemented for the development of information systems, which are not only to publish information, but also to update it. Some results already have been obtained and will be published in the foreseeable future.

Moreover, we have started two information systems, which were implemented on SMART technology. First one is a "**Conference information system**", which is located in Internet at http://web.ict.nsc.ru:8080/conf. It is a new version of the existing system, which is intended to be a handy tool for conference holding both for organizers and participants. The second system called "**Atmospheric aerosols of Siberia**" is located at http://web.ict.nsc.ru:8080/aerosol and created for publishing various scientific information about aerosols: constitution, characteristics, behavior. It also includes some service function for mathematical data processing and its graphical representation. Both projects are still in test mode and sometimes they may be unavailable.

REFERENCES

[1] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation; [http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/]
[2] SMART: System for Managing Applications based on RDF Technology; [http://web.ict.nsc.ru/smart]
[3] Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation; [http://www.w3.org/TR/rdf-concepts/]
[4] Java Servlet Technology; [http://java.sun.com/products/servlet/]
[5] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000; [http://www.w3.org/TR/REC-xml]
[6] XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999; [http://www.w3.org/TR/xslt]
[7] XML Software: content management systems; [http://www.xmlsoftware.com/dms.html]
[8] Content Management Systems Suppliers and Vendors; [http://www.contentmanager.eu.com/providers.htm]
[9] XML Software: publishing systems; [http://www.xmlsoftware.com/publishing.html]
[10] The Apache Cocoon Project; [http://cocoon.apache.org/]
[11] *Tim Berners-Lee*. Semantic Web Road map; [http://www.w3.org/DesignIssues/Semantic.html]
[12] Semantic Web Activity; [http://www.w3.org/2001/sw/]
[13] *Tim Berners-Lee, Eric Miller*. The Semantic Web lifts off // ERCIM News No. 51, October 2002. [http://www.ercim.org/publication/Ercim_News/enw51/berners-lee.html]
[14] *James Hendler, Tim Berners-Lee, Eric Miller*. Integrating Applications on the Semantic Web // Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), October, 2002, p. 676-680. [http://www.w3.org/2002/07/swint]
[15] Dublin Core Metadata Initiative (DCMI); [http://www.dublincore.org]
[16] RDF Site Summary (RSS) 1.0; [http://web.resource.org/rss/1.0/]
[17] *Gruber, T.* A translation approach to portable ontology specifications. // Knowledge acquisition, 5(2), 199-220. [http://ksl-web.stanford.edu/knowledge-sharing/papers/README.html]
[18] RDQL - RDF Data Query Language; [http://www.hpl.hp.com/semweb/rdql.htm]
[19] The Protégé Ontology Editor and Knowledge Acquisition System [http://protege.stanford.edu/]
[20] Dave Beckett's Resource Description Framework (RDF) Resource Guide; [http://www.ilrt.bris.ac.uk/discovery/rdf/resources/]