Journal of Universal Knowledge Management, vol. 1, no. 3 (2006), 271-294
submitted: 5/10/06, accepted: 15/11/06, appeared: 28/12/06 © J.UKM

Performance Solution of SOA Infrastructure for Knowledge Computing

Miroslav Kubásek
(Faculty of Informatics, Masaryk University, Czech Republic
xkubasek@fi.muni.cz)
Jan Pavlovič
(Faculty of Informatics, Masaryk University, Czech Republic
pavlovic@fi.muni.cz)
Tomáš Gregar
(Faculty of Informatics, Masaryk University, Czech Republic
xgregar@fi.muni.cz)

Abstract: In this paper we will present a complete solution of SOA designed for knowledge computation encapsulation. SOA brings a lot of advantages to the whole ICT process when a difficult on-demand task is computed. On the other hand SOA overhead is nowadays unacceptable for this kind of computation tasks. We have used a semantic approach to describe SOA. Some contributing ideas have appeared – for example a possible approach to cache web services ontologically. This can help in knowledge computing.
Key Words: SOA, Web Services, JMS, Ontology, Cache
Category: D.2.2, D.2.11, H.3.5

1 Introduction

Almost every scientific on-demand computation has a similar structure and obeys similar rules. We have some input data with a hidden knowledge we are to obtain as a result of computation process, which requires large hardware and software resources.

This means two main problems. We need to build a new hardware and structure for each computation task. The second problem is visualization of the result or other form of result representation. The whole solution consists of a great number of particular computations which are somehow represented to the user. SOA represents a functional encapsulation of computations and simplifies the communication interface between application and presentation logic. Such architectures can be easily integrated to other software solution and information systems [Král 05]. However, we need to solve the large overhead of SOA.

1.1 Data Security Management

Although security denotes different things with respect to software systems, in general, it is associated with: *Confidentiality* – Access to information/service

is granted only to authorized subjects; *Authenticity* – We can trust that the indicated author/sender is the one responsible for the information; *Integrity* – Information is not corrupted; *Availability* – The information/service is available in a timely manner.

Security is a major concern for SOA and Web services. Messages often contain data in text format (e.g., XML), and, even worse, metadata is embedded. Encryption must be in place to preserve privacy.

A system built using a SOA approach may encompass services provided by third-party organizations. Trust must be built into the security of such external services (e.g. authentication, protecting stored data, authorization to allow configuring and enforcing permissions of specific user, groups or roles etc.). An SOA solution may rely on looking up services in a public directory. It is important to ensure that information in the directory is up to date and was added by valid publishers.

Web services solutions have been addressing some of the security concerns at the network infrastructure level. For example, Web servers that host Web services can be configured to use Secure Sockets Layers (SSLs) and digital certificates to encrypt data transmission and authenticate the communicating parties. In intranet solutions, Kerberos is an option. However, these solutions merely help to protect point-to-point interaction: A comprehensive mechanism that covers end-to-end security is required.

The architect should be aware of the security features offered by the target Web services platform, because security mechanisms often have a negative impact on performance and modifiability. Also, adherence to security standards is important for preserving of interoperability. In addition to the SOA security mechanisms available (e.g., encryption, authentication, and authorization), the architect should consider the configuration required for the infrastructure of the chosen technology with respect to security. In a Web services solution, the firewall rules don't have to change because the SOA interaction is over a protocol that is normally open (e.g., HTTP or SMTP).

2 ESB

The last year brought some significant technology trends like Service-Oriented Architecture (SOA), Enterprise Application Integration (EAI), Business to Business (B2B), and web services. These technologies have tried target the challenges of improving the results and increasing the value of integrated business processes, and have gathered attention of industry analysts and IT specialists. The Enterprise Service Bus (ESB) combines the finest traits from these and other technology trends [Chappell 04].

The ESB principle represents a new way of looking at integration of loosely coupled and highly distributed integration network that can rise above the lim-

its of a hub-and-spoke EAI broker. An ESB is standardized integration platform for messaging, web services, data transformation. Moreover it connects ad coordinates the interaction of large numbers of different application throughout enterprises applications with transactional integrity.

The ESB brings a new way of incorporating web services and SOA into a robust architecture that integrates services and applications into a backbone that bridges the enterprise applications. An EBS enables immediate usage of web services and other integration technologies with the modern technologies of nowadays.

The ESB is a platform independent concept. Ideally an EBS could be implemented without any particular technology. Although a perfect ESB has to take advantage of Java components since the Java technology is largely used in many IT areas.

An ESB can profit from many of the technologies belonging to the Java EE and Java SE range, such as JMS, JCA, EJB, JSP, JAAS, SAAJ, JSSE, JSP, JAXB, JAX-RPC, and JMX. They do not need to be used all together at every installation. There exist a small number of Java specifications that deserve particular attention due to their impact on the functionality of an ESB.

2.1 Java Business Integration (JBI)

The Java Business Integration (JBI) initiative (JSR-208[1]) is a pursuit in the JCP to create a specification that describes the way how to integrate components. These components can be integrated together in a vendor-neutral style.

JBI is regarded as having an "enterprise" ability. In the main goal of the JBI is not to require the whole Java EE application server. JBI is especially aimed toward allowing vendors no matter if application server based or not to be able to plug components together in a interoperable way. Although JBI can use the Java EE server, the main goal of JBI lies in broader adoption beyond the handful of Java EE application server vendors. The main idea is to acquire an environment where any vendor dealing in the field of integration components and infrastructure may provide a JBI-compliant infrastructure, or plug their wares into one.

JBI offers a way of integrating services hosted in a managed environment allowing pluggable Service Engines from third-party vendors to interoperate together. These engines and their corresponding services could be anything that provides integration and process management. Service Engine could be as XSLT transformation engine, a CBR service or an WS-BPEL orchestration engine. These services can be provided by several different vendors, and still be able to work together in the same managed environment.

[1] `http://jcp.org/en/jsr/detail?id=208`

The acceptance of JBI will help to acquire and accelerate an environment around pluggable, interoperable integration components. This enables to build around a model of third-party services that can easily plug into an ESB environment.

2.2 Other Solution

The SOA solution is nowadays definitely a modern way of designing an information system. Even for scientific computations there are vast projects dealing with the idea of a virtual grid based on web services. There are two main representatives in this area - the Globus[2] and the gLite[3] projects. These solutions are suitable for complicated tasks where you can spend a lot of time with configuring the underlying layer.

In Supercomputing Centre of Brno[4] Masaryk University is running the Globus and it has taken months to configure it.

3 Design

The design of system consists from several layers Figure 1. The first layer is the Knowledge Layer this is represented mainly by Data Warehouse. Upon Knowledge Layer is hardware including clusters or standalone computer. Individual SWS (Semantic Web Services) are mapped to this layer in M:N relationship. Next layer contains from description units as UDDI and ontologies.

With the SOA Bus Layer the system is encapsulated and client applications can consume the services. The Bus Layer can be easily done with any implementation of Enterprise Service Bus. We have tested the Sun Open ESB project[5]. There is possibility to add JMS server to increase performance. The visualizations of received knowledge are upon the clients.

4 Performance

In general, performance is related to response time (how long it takes to process a request), throughput (how many requests overall can be processed per unit of time), or timeliness (ability to meet deadlines, i.e., to process a request in a deterministic and acceptable amount of time). Performance is an important quality attribute that is usually affected negatively in SOA. Careful design and

[2] http://www.globus.org/wsrf/
[3] http://glite.web.cern.ch/glite/
[4] http://scb.ics.muni.cz/static/index_en.html
[5] https://open-esb.dev.java.net/

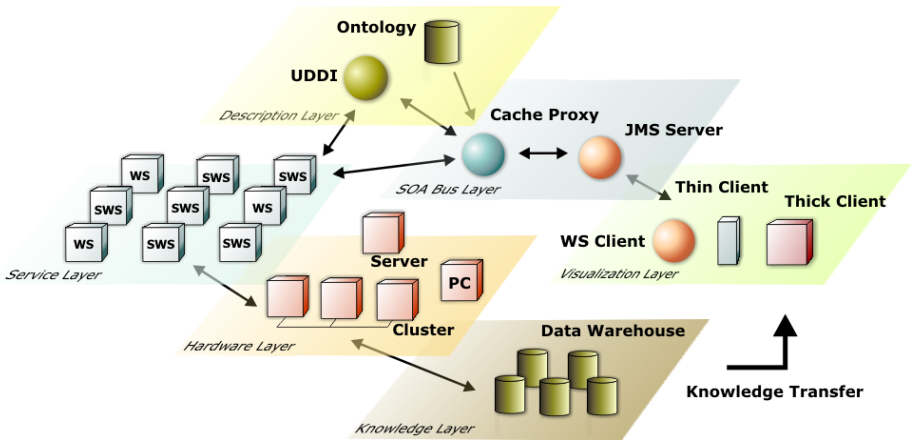

Figure 1: System Architecture

evaluation of the architecture for the specific solution is necessary to avoid performance pitfalls. The key factors in SOA that contribute to performance issues are mentioned below [OBrien et al. 05].

SOA involves distributed computing. Service and service user components are normally located in different containers, different machines. The need to communicate over the network increases the response time. Typical networks used for SOA, such as the Internet, do not guarantee deterministic latency. Therefore, SOA is not considered a feasible solution for real-time systems, but presents challenges for near real-time systems, where latency is not a safety-critical requirement but rather a business one (i.e., meet business goals). For a heavily used service, many queued requests may already be outstanding, and they are usually serviced in a FIFO manner. Such a situation can have a significant impact on latency, though it can still be predicted stochastically. However, if more queue space has to be created dynamically, latency will be impacted further.

The interaction protocol sometimes requires a call to a directory of services to locate the desired service. This extra call increases the total time needed to perform the transaction. One way to reduce the response time and improve throughput is to prevent the call to the directory by having the location of the provider end point hard-coded (or cached after the first lookup) in the service user. However, hard-coding reduces availability, and caching must be reestablished after failure when another replica is found.

The ability to make services on different platforms interoperates seamlessly request data marshalling and handling all communication between a service consumer and a provider. Depending on the SOA technology or framework being used, stubs, skeletons, SOAP engines, proxies... are in place. All such interme-

diaries negatively impact performance.

The use of a standard messaging format increases the time needed to process a request. With messages described by ontology it is increased further.

On the positive side, SOA provides location transparency. Service users do not necessarily know the location of the service until they look it up in the registry. Thus, a deployed service can be moved from location to location without affecting the consumers. This feature permits the deployment of services to multiple locations (replicas), which can be allied to a load-balancing strategy to improve the total throughput and availability of the system.

Many SOA technologies permit the service user to call the provider asynchronously. In that case, the user does not get blocked waiting for the response. For operations that fit that model of interaction, asynchronous calls should be used to reduce the response time.

5 Caching solution

Current Web caching systems leverage unique identifiers pointing to content fragments. They leverage the underlying protocols to keep track of the timeliness of data bits, such as HTML documents or images. While Web services over the HTTP transport have endpoints (URLs), the parts of the Simple Object Access Protocol (SOAP) envelope that make it unique are the values for parameters in the SOAP envelope.

Because SOAP is XML, several tags can have many different labels but still mark up the same data. Taken as a whole, it is possible for you to create a SOAP envelope that is different from the next while representing exactly the same data. Moreover, if you decided to cache based on the individual unique parts of a SOAP envelope, there is no way to indicate if the query is suitable for such caching.

Possible solution for quick matching with stored SOAP messages is to use hash of canonicalized XML messages as a key value [Boyer 01]. For simplifying the generation of canonicalized messages we can use some kind of templates which can be generated from ontology description of used Web Services (extended WSDL).

The Figure 2 describes the structure of proxy cache which uses the knowledge about WS from its WSDL description (i.e. description of its input and output) provided by UDDI. The SWS (Semantic Web Service) in this figure is the extension of classic Web Service with semantics knowledge about request and response messages.

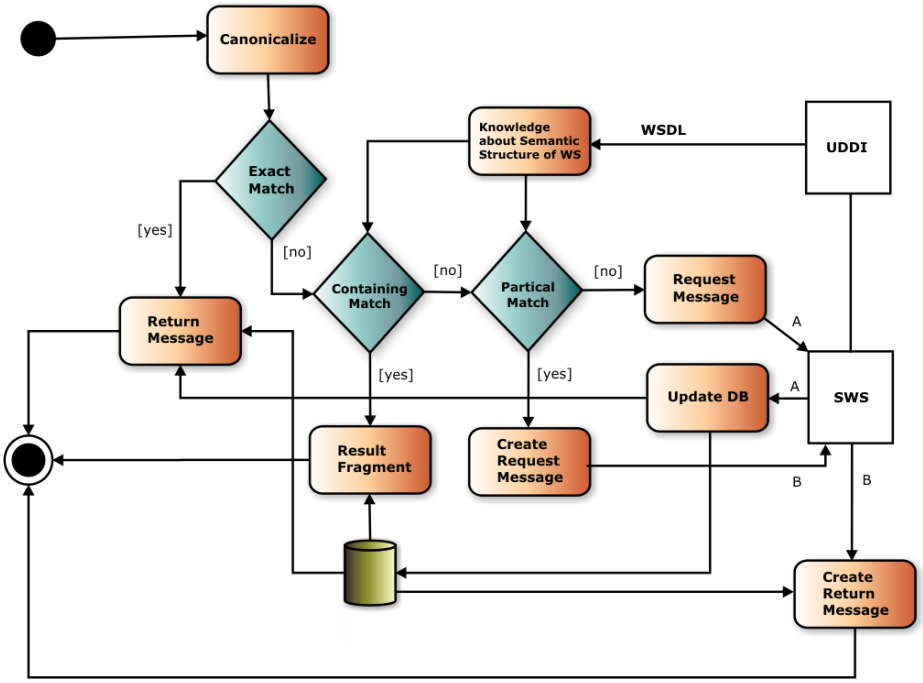

Figure 2: Cache with Semantic Knowledge

5.1 What and where we can cache?

Most processes involve calculation. The calculation can be an equation or a network transaction that results in a new value. Either way, you should identify the parts of the process that waste time.

How long do you keep these cached items around, and how often are other clients going to call the Web service?

The first question goes to the fact that every system has limited resources. Keeping 100 MB of audio data in memory might be too storage intensive. You can design a cache to have a maximum size. It can keep the most used entries around while purging the oldest and least used. You might decide not to have a time limit for this cache.

The second issue highlights that the service might be so infrequently leveraged by other clients that caching anything might be useless. For example, assume the total transaction time of an imaginary Web service is two seconds. If the calls to this Web service are infrequent (less than one in a minute), it might be determined that there is no need for caching. However, if the load is 100 calls in a second, it might be justified. An alternative approach is to have a self-pruning cache that stores entries, as needed, but prunes them at an expira-

tion time. Pruning the cache object reduces its memory footprint. Depending on the implementation, the cache could afford the ability to keep popular cached objects around longer than the infrequently used items.

Service in SOA is behavior-based and its functionality is encapsulated. Demands and its responses are transferred by messages with defined syntax. But web services are meaningful only if potential users may find information sufficient to permit their execution. Traditional implementations of SOA use strictly syntactic standards – WSDL, SOAP, REST (meintioned earlier). Services, agents and applications have no knowledge about content of the messages. The whole collaboration is possible only because of use the same standards.

With some semantical information, ie. information about messages' content and/or system architecture, we can improve caching algorithm. We can store more than knowledge of exact match of demand-message with its response. Connection of concepts (keywords) found in query messages with data returned builds logical predicates, which can be further used (with other information of system architecture and rules).

6 Caching

Data caching and query caching are two widely used strategies for Web caching. As regards to the data caching strategy, a major concern is whether the data granularity in the cache is page or tuple. In this strategy, special proxy servers maintain the pages or tuples recently accessed and when users request same pages or tuples by URL, the cache will return its corresponding content directly to users so that it can save the communication cost between proxy servers and user clients. In data integration system user queries do not contain URL information, but they contain bag of keywords. By these reasons the query caching strategy works well, and is also known as semantic caching. The semantic caching strategy manages semantic region grouping a collection of data objects corresponding to a specific query. For example, in the cache of search engine system, a semantic region normally contains a set of Web documents returned by Web search engines on the query of some specified keywords.

6.1 Semantic Based Caching

Generally, there are three main issues concerned for a semantic caching strategy: cache region organization, matching mechanism, and cache replacement policy[Kubasek et al. 06].

With respect to the first issue, data objects are organized into corresponding to queries e.g., relational predicates in relational database or set of variables in Web Services area. Further the regions may or may not be disjoint, depending on design of different systems. There are also special techniques used for cache

organization in different systems, for example, signature method or reference counter technique.

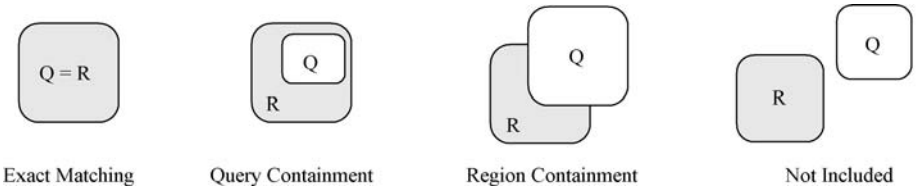

Figure 3: Matching types in semantic caching

As for the issue of matching mechanism, generally there are these matching types: *exact matching*, *query containment* and *region containment*. Briefly put, exact matching happens when a new query is exactly the same as the region formula, i.e., the description of the region; query containment indicates the case in which a new query is subordinate to at least one region formula so that it can be entirely satisfied by the cache, here "a formula A is subordinate to a formula B" means that the complete query result corresponding to A is a proper set of that corresponding to B; inversely, region containment means at least a region formula in the cache is subordinate to the query formula so that the query can be satisfied partially by the cache; at last, the intersection matching type is the case when the result of the query and the data residing in a semantic region are not disjoint, but neither of them is subordinate to the other. Since both of the query containment and the region containment can be associated with multiple semantic regions, there are altogether four matching cases, which are demonstrated in the following Figure 3. Once the matching case is determined, a probe part for the query can be generated from the matched semantic regions, and the rest results i.e., the remainder part of the query, can be fetched by information retrieval. At last the probe part and the remainder part together make up the query result.

There is a lot of research work on semantic caching, e.g., Chidlovskii's semantic caching strategy on Meta Web search engine [Chidlovskii and Borghoff 00]; Lee's semantic caching strategy in CoWeb [Lee and Chu 01]; and Chen's work on XCache, a semantic caching architecture for XML [Chen et at. 02].

6.2 Semantic Based Caching Model

We will introduce a caching model to represent the cached SOAP messages to decide that the cached information is enough to answer a query. In definitions we will use the notation described in [Miklau and Suciu 02].

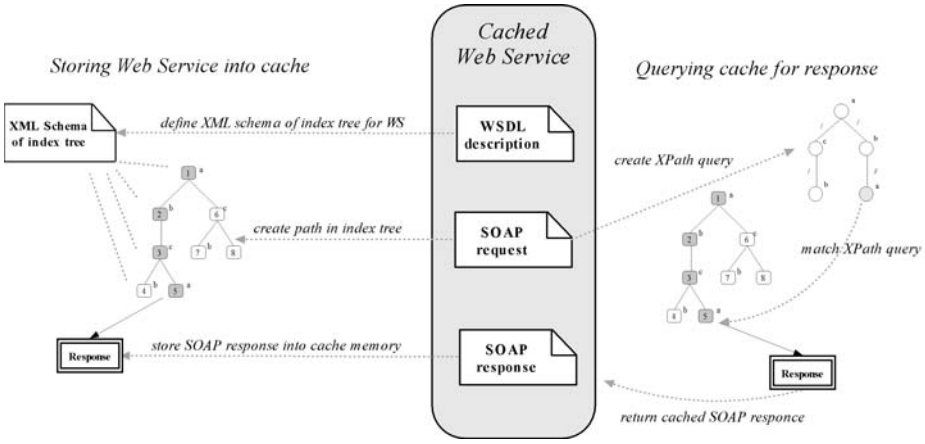

Figure 4: Model of cache for storing and quering Web Services messages

Generally, an cache memory consists of a set of SOAP messages. We model the whole memory as an unordered rooted node-labelled XML tree over an infinite alphabet Σ. A virtual root node might be introduced to connect all XML documents if necessary. In this XML tree, each internal node's label corresponds to an XML element or attribute name, and each leaf node's label corresponds to a data value. In addition, we assume that each node has a unique node identifier. We let T_Σ be the set including all possible XML trees over Σ. Formally, we have:

Definition (WS tree): *An Web Service Cache database is a tree* $T = \langle V, E, r\rangle$ *over* Σ *called* **WS tree**, *where*

1. V *is the node set and* E *is the edge set.*
2. *The* $r \in V$ *is the root of* T.
3. *Each node* $n \in V$ *has a label, denoted as* $n \diamond label$, *whose value is* $n \diamond label \in \{'*','//'\} \cup \Sigma$.
4. *Each node* $n \in V$ *has a unique node identifier denoted as* $n \diamond id$. □

Definition (Size): *Given an WS tree* $T = \langle V, E, r\rangle$, *the* **size of** T *is defined as the cardinality of* V, *and we also say that* $T' = \langle V', E', r'\rangle$ *is a* **subtree** *of* T *if* $V' \subseteq V$ *and* $E' = (V \times V) \cap E$.

Definition (Rooted subtree): *Given an WS tree* $T = \langle V, E, r\rangle$, *a* **rooted subtree** $T' = \langle V', E', r'\rangle$ *is a subtree of* T *if it satisfied the folowing conditions:* $r' = r$ *and* $V' \subseteq V$ *and* $E' \subseteq E$.

Next I will use fragment of XPath queries denoted as X^{PQ}. This fragment consists of label tests, child axes(/), descendant axes(//), branches([]) and wild-cards(*). It can be recursively represented by the following grammar:

$$q \longmapsto v \mid * \mid q/q \mid q//q \mid q[q] \quad \mid \quad v \in \Sigma$$

Any $q \in X^{PQ}$ can be represented as a labelled tree (called **query pattern tree**) with the same semantics Figure5 b).

Definition (query pattern tree): *A* **query pattern tree** P *is a tree* $P = \langle V, E, r, o \rangle$ *over* $\Sigma \cup \{'*'\}$*, where* V *is the node set and* E *is the edge set, and:*

1. *Each node* $n \in V$ *has a label from* $\Sigma \cup \{'*'\}$*, denoted as* $n \diamond label$*;*
2. *Each edge* $e \in E$ *has a label from* $\{'/','//'\}$*, denoted as* $e \diamond label$*. The edge with label* $'/'$ *is called child edge, otherwise called descendent edge;*
3. $r \in V$ *is the root node of* P *and* $o \in V$ *is the output node of* P*.*

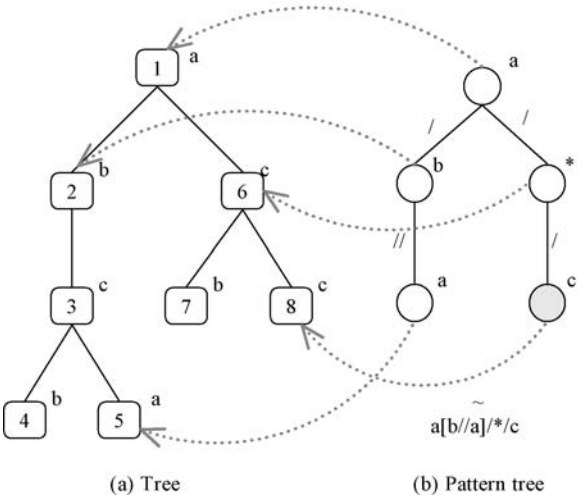

Figure 5: Example of tree and pattern tree

In order to decide if a WS tree T is included in some query pattern tree P, we need to define the semantics of tree inclusion. Several definitions of tree inclusion exist including subtree inclusion [Ramesh and Ramakrishnan 92], tree embedding [Zaki 02] and tree subsumption [Giunchiglia and Walsh 92]. The most

relevant definition is the subtree inclusion, which states that a WS subtree T' is included in some WS tree T if and only if there exists a subtree of T that is identical with T'.

However, this definition is too restrictive for XML query pattern trees where handling of wildcards and relative paths are necessary.

Consider the an WS trees $T = \langle V_t, E_t, r_t \rangle$ and query pattern tree $P = \langle V_p, E_p, r_p, o_p \rangle$ in Figure5. Let $match(p, q)$ denote that a node $p \in V_t$ is mapped to a node $q \in V_p$.

Since we are dealing with rooted subtrees, we can carry out a top-down matching. Here, ('a','a') is mapped first. Next, we check that each subtree of 'a' in T matches with some subtree of 'a' in P. This requires that the subtree rooted at section of T (denoted as $subtree(section)$) has to be matched against the subtrees rooted at 'b' and '*' of P. We need to consider whether '//' indicates zero or many nodes in the path:
Case 1: '//' means zero length. Then subtree(2) must be included in either subtree(5) of P, which is not the case here.
Case 2: '//' means many nodes. This implies that section has been mapped to some 'unknown' node in P. From all the possible subtrees of section, only one subtree, i.e., subtree(3), must be included by subtree('//').

I now define an **pattern match** (also called embedding) from a query pattern tree to an WS tree as follows:

Definition (pattern match): *Given an WS tree $T = \langle V_t, E_t, r_t \rangle$ and a query pattern tree $P = \langle V_p, E_p, r_p, o_p \rangle$, an pattern match from P to T is a function $match() : V_p \longmapsto V_t$, with following properties:*

1. *Root preserving: $match(r_p) = r_t$;*
2. *Label preserving: $\forall n \in V_p$:*
   if $n \diamond label \neq '*' \Rightarrow n \diamond label = match(n) \diamond label$;
3. *Structure preserving: $\forall e = (n_1, n_2) \mid n_1, n_2 \in E_p$:*
   if $e \diamond label = '/'$ then $e(n_2)$ is a child of $e(n_1)$ in T; otherwise, $e(n_2)$ is a descendent of $e(n_1)$ in T.

Function $match()$ maps the output node o_p of P to a node $n \in V_t$. We say that the node n is the result of this embedding. As an example, dashed lines between Figure5 (a) and (b) shows an embedding, and its result is the node with $id = 8$. Actually, there could be more than one embedding from P to T. We define the result of P over T, denoted as $P(T)$, as the union of results of all embeddings, i.e.,

$$P(T) = \bigcup_{m \in M} \{m(o_p)\}$$

where M is the set including all pattern matches from P to T.

For a given XML tree T, we also consider evaluating a set of patterns $S^T = P_1, P_2, ..., P_n$ over T. The result, denoted as S^T, is the union of the result of evaluating each $P_i \in S$ over T, formally defined as:

$$S^T = \bigcup_{P_i \in S} \{P_i(T)\}$$

Lemma: *For any two patterns P_1 and P_2, we said that P_1 is contained in P_2 (denoted as $P_1 \sqsubseteq P_2$) if* $\forall T \in T_\Sigma \mid P_1(T) \subseteq P_2(T)$.

Lemma: *For any pattern P and pattern set S^P, we said that a pattern P is contained in a pattern set S^P (denoted as $p \sqsubseteq S^P$) if* $\forall T \in T_\Sigma \mid P(T) \subseteq S^T$.

Lemma: *For any two pattern sets S_1^P and S_2^P, we said that pattern set S_1^P is contained in a pattern set S_2^P (denoted as $S_1^P \sqsubseteq S_2^P$) if* $\forall T \in T_\Sigma \mid S_1^T \subseteq S_2^T$.

We can also show that $S_1^P \sqsubseteq S_2^P$ if $\forall P_i \in S_1 \mid P_i \sqsubseteq S_2^P$. However, it's not always true that $P \sqsubseteq S^P \Rightarrow \exists P' \in S^P \mid P \sqsubseteq P'$.

Proof: Below is describet algorithm for *contains* which check the containment of tree patterns. It maintains a two-dimensional array *status* , which is initialized with $stutus[v, w] = null$ to indicate that $v \in nodes(p)$ and $w \in nodes(q)$ have not been compared. Otherwise, $status[v, w] \in \{true, false\}$ sutch that $status[v, w] = true$ if and only if $subtree(w, q) \sqsubseteq subrtee(v, p)$. Clearly $q \sqsubseteq p$ if and only if $status(v_{root}, w_{root}) = true$ where v_{root} is the root node of p and w_{root} is the root node of q. □

Given an WS T created from SOAP message of Web Service and a set of tree pattern S^P, we have the S^T in the cache. We want to use this S^T to answer patterns created from SOAP requests. But, we need to assure that S^T can totally answer them, before evaluating them against S^T. From Corollary 1, we know that S^T can totally answer those patterns included in S^P. However, S^T can totally answer more patterns not only in S^P. This advantage of Semantic Caching Model will be further discuss. The problem is how to decide whether S^T can totally The problem is how to decide whether S^T can totally answer a pattern P or not. The basic idea is to check whether T^P (represented by P) is a rooted subtree of S^T or not. We have the following result:

Lemma: *Given an WS tree T, a pattern P and a pattern set S^P, S^T can totally answer P if T^P is a rooted subtree of S^T.* □

From the above lemma, the problem is reduced to decide whether or not T^P

$contains(p, q)$
Input: p, q // tree patterns
Output: Returns $true$ if $q \sqsubseteq p$; $false$ otherwise

$status[v, w] = null \mid \forall v \in nodes(p), \forall w \in nodes(q)$
v_{root} = root node of p
w_{root} = root node of q
if $child(v_{root}, p) = \emptyset$ **then**
  return $true$
else
  return $containssub(v_{root}, w_{root}, status)$
end if

$containssub(v, w, status)$
Input: v, w // nodes in tree patterns p, q
$status[v, w]$ // 2-dimensional array such that each $status[v, w] \in \{nul, false, true\}$
Output: $status[v, w]$

if $status[v, w] \neq null$ **then**
  return $status[v, w]$
end if
if v is a leaf node in p **then**
  $status[v, w] = (label(w) \preceq label(v))$
else
  if $label(w) \not\preceq label(v)$ **then**
    $status[v, w] = false$
  else
    $status[v, w] = \bigwedge_{v\prime \in child(v,p)} (\bigvee_{w\prime \in child(w,q)} containssub(v\prime, w\prime, status))$
  end if
end if
if $(status[v, w] = false)$and$(label(v) = //)$ **then**
  $status[v, w] = \bigwedge_{v\prime \in child(v,p)} containssub(v\prime, w, status)$
end if
if $(status[v, w] = false)$and$(label(w) = //)$ **then**
  $status[v, w] = \bigvee_{w\prime \in child(w,q)} containssub(v, w\prime, status)$
end if
return $status[v, w]$

Algorithm 1: Pattern contain checking Algorithm

is a rooted subtree of S^T. We can further reduce our problem to decide whether $S^{PT} \subseteq S^{S^T}$ or not in our next result.

Lemma: *Given an WS tree $T = \langle V, E, r\rangle$ and two node sets $N \mid N \subseteq V$ and $N' \mid N' \subseteq V$, the minimal covered tree for N over T is a rooted subtree of the minimal covered tree for N' over T if $N \subseteq N'$.*

Proof: Hence, T^P is a rooted subtree of S^T if $S^{PT} \subseteq S^{S^T}$. However, there could be some patterns that S^T can totally answer but their embedding node sets are not included in S^{S^T}.

For a given pattern P, the nodes in S^{PT} mapped from the internal nodes of P are redundant to represent an WS tree. The following definitions and lemmas are given to deal with this case.

Definition (matches leaf node set): *Given an WS tree T and a tree pattern $P\langle V, E, r, o\rangle$, an* **matches leaf node set** S^{PT}_{leaf} *is defined as*

$$S^{PT}_{leaf} = \bigcup_{m \in M} (\bigcup_{v_i \in V_{leaf}} \{m(v_i) \diamond id\})$$

where M is a set including all possible pattern matches from P to T and $V_{leaf} \subseteq V$ includes all leaf nodes of P. □

For an WS tree T, we similarly define that the matches leaf node set for a pattern set S^P, denoted as $S^{S^P T}_{leaf}$, is the union of the embedding leaf node set for each $p_i \in S^P \mid \bigcup_{p_i \in S^P} S^{PT}_{leaf}(P_i, T)$. We have the following results:

Lemma: *Let t be an WS tree. For a given pattern P and a pattern set S^P, the following hold:*

- *The minimal covered tree for C^{PT}_{min} over T^P is identical to the minimal covered tree for $S^{C^{PT}_{min}}$ over T.*
- *The minimal covered tree for $C^{PT^P}_{min}$ over S^{T^P} is identical to the minimal covered tree for $S^{C^{PT^P}_{min}}$ over T.* □

From previous three lemmas we easily have that T^P is a rooted subtree of S^T if $S^{C^{PT}_{min}} \subseteq S^{C^{PT^P}_{min}}$.

So far, we reduce the problem of deciding whether S_T can totally answer P or not to the problem that whether $S^C PT \subseteq S^C S^P, T$ or not. We next consider how to decide $S^C PT \subseteq S^C S^P, T$.

We denote a pattern $P\langle V, E, r, o\rangle$ as $P_{\triangleright o}$, where $o \in V$ is the output node. We also can choose any node $v \in V$ as the output node of P. For example, the pattern P with a node v_1 instead of o as the output node can be denoted as $P_{\triangleright v_1}$

. For a pattern P, we introduce a tree pattern set $S^{P_{\triangleright}T}$ including all patterns by choosing each node in P as the output node, and this pattern set can be formally defined

$$S^{P_{\triangleright}T} = \bigcup_{v_i \in V} \{P_{\triangleright v_i}\}$$

If we only choose leaf nodes in P as the output node to build the set, we denote it as $S^{P_{\triangleright}T}_{leaf}$.

Next result shows that the node set $S^{C^{PT}}$ is equal to the result of evaluating the pattern set $S^{P_{\triangleright}T}_{leaf}$ over T.

Lemma: *Given an WS tree T and a pattern P, $S^{PT}_{leaf} = S^{P_{\triangleright}T}_{leaf}$.*

For a pattern set S, we similarly define a pattern set

$$S^{S^P_{\triangleright}T}_{leaf} = \bigcup_{p_i \in S^P} S^{P_{i\triangleright}T}_{leaf}$$

From the above lemma, we can have $S^{S^PT}_{leaf} = S^{S^P_{\triangleright}T}_{leaf}$.

By combining all above lemmas, we finally have the following conclusion:

Corollary 2: *Given an WS tree set S^TT, a pattern P and a pattern set S^P. S^T can totally answer P if $S^{P_{\triangleright}T}_{leaf} \sqsubseteq S^{S^P_{\triangleright}T}_{leaf}$.*

We reduce the problem deciding whether S^T can totally answer P or not to the containment problem between two pattern sets $S^{P_{\triangleright}T}_{leaf}$ and $S^{S^P_{\triangleright}T}_{leaf}$.

6.3 Semantic Based Caching in the Web Service Cache

Basically, semantic caching in our proxy cache is done by annotating WSDL documents with information about the caching-relevant semantics of services. This information is used for mapping SOAP requests to predicates, for fragmenting responses, and for reassembling responses. Thus, adapted semantic caching algorithms can be applied.

Using a annotations we are now able to understand the caching-relevant semantics of requests and responses. The schema of Transparent Semantic Cache is shown in Figure6.

7 Semantic architecture

As written before, our caching metodology annotate web services description (WSDL) with semantical knowledge. This is one of possible sources of semantical description of services, and more generally - whole architecture.

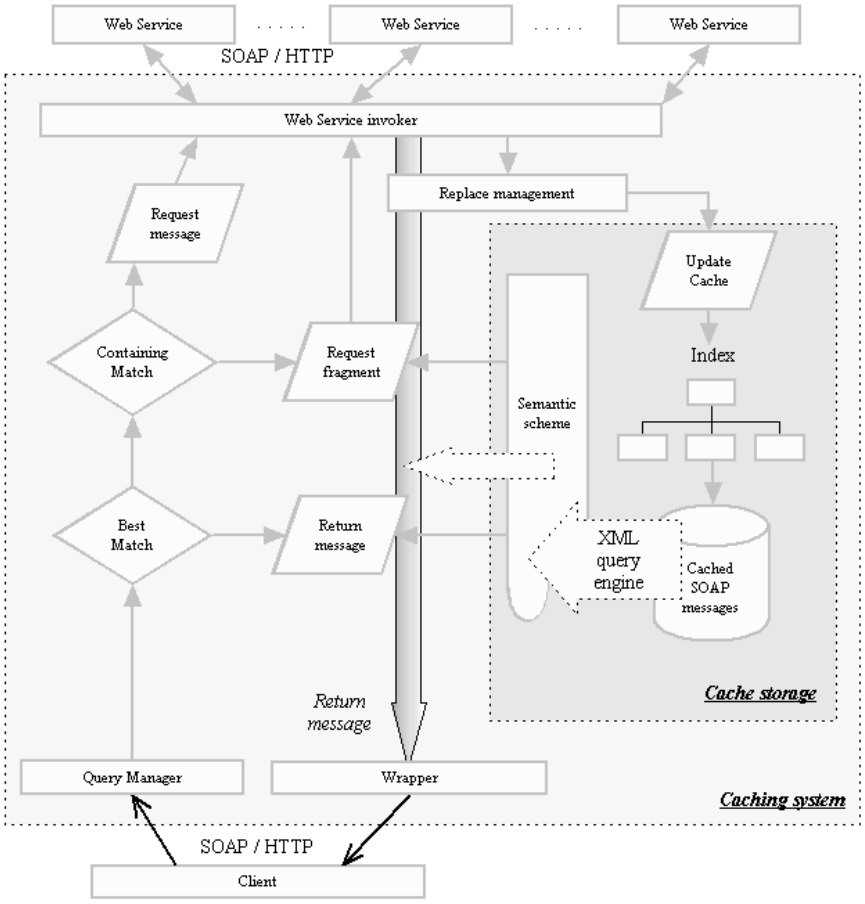

Figure 6: Schema of Proxy Cache with semantic knowledge

Formal description of architecture, like UML, is used in enterprise development. With technology such as MDA (model driven analysis), the coding can be automated with supporting tools from formal description. But any formal description can be viewed as constrained domain ontology, so it is possible to build a description of architecture with help of ontology of studied domain. Ontology definition metamodel (ODM) and Ontology UML profile was developed for such process (or build ontology with help of MDA) [Gašević et al. 06]. Binding ontology and system model in UML describes [Pondrelli 05].

Quality of service (QoS) ontology can be built and merged with such system architecture ontology [Lock and Sommerville 05]. Ontology description of an ar-

chitecture allows to prepare variety of service-structure views [Nitzsche et al. 05].

7.1 Semantic services

Within SOA methodology, service is defined by a set of messages that it can handle [Korotkiy and Top 05]. Its structure must be defined (typically implicitly in some scheme). The structure can be also defined explicitly, when the concepts are based on ontology – this is how "common language of the system" can be found (and also changed through the processing).

For the semantic description of web services (these are called SWS, Semantic Web Services then) standards like OWL-S [OWL-S 04] or WSMF/WSMO were proposed [Korotkiy and Top 05]. These standards goes further than described semantic annotation.

A goal of OWL-S or WSMO projects is to build ontology (set of ontologies, respectively) for full description of web services (i.e. its properties and capabilities).

OWL-S standard built set of ontologies, which can system engineer use for describing web services. Main OWL-S ontologies transcribes rules and concepts for services, its profiles, its processing and its grounding with WSDL. As can be viewed from the name, it is based on W3C-standard OWL language. As an alternative to more common way (WSDL, SOAP) its helping ontology QWL-S Grounding supports exchanging messages.

WSMO, on the other hand, is framework to build such ontology structure oneself. It also differs in philosophical level. It does not use standard *service-message* relation (where service is described by WSDL and message structure by SOAP/REST). Every web service should be built as isolated entity. WSMO provides mediation (via mediators) of input and output messages between web services.

But – despite of different approach, both provide the possibility of acquiring and resolving information about web services.

7.2 Semantic architecture

With further level of abstraction, we can think about the *entire* application as a semantic network. We can model business logic of application with an ontology technologies like BPEL4WS [Andrews et al. 03] (Business Process Execution Language for Web Services), WSMF (framework, which WSMO is part of) or SWSF [Battle et al. 05].

Semantic web service framework (SWSF), and also BPEL4WS – uses main idea of *message-driven architecture*, i.e. service is described by messages it can handle, it is process-oriented. WSMF use different approach – it describes Web service choreography through guarded transition rules.

SWSF contains:

- Semantic Web Services Language (SWSL) – language for formal description of web service (and its concepts). One part is based on first order logic (named SWSL-FOL) and is used mainly for SWSO ontology building, other part is based on logic programming (SWSL-Rules) and supports reasoning and executions above service ontology.
- Semantic Web Services Ontology (SWSO) – conceptual model for web services description (and model axiomatization). Subontologies are *FLOWS* (First-Order Logic Ontology for Web Services) (draws many of its intuitions and lessons-learned from OWL-S) and *ROWS* (Rules Ontology for Web Services). FLOWS ontology provides constructs for modeling of the internal processing of the Web services, and also service side-effects (on the world). It is also intended to enable reasoning about essential aspects of web service behavior for a variety of different purposes and contexts.

7.3 Semantic SOA

With this technologies in mind, we can view the ontology-bound web services in two equivalent ways:

1. *Web services processing some function, transforming input into output.* Services are describable by ontology (see preceding sections).
2. *Part of ontology.* In this case we can identify the web service as a special predicate. Its output, value of predicate, is not predefined. Web service works as a black box reasoner with narrow, specified domain. Operating semantics of a system is defined by these service concepts in ontology. We can use general-purpose ontological reasoners on this-way defined ontology. Reasoning could find hidden information about system and help to make its behaviour more effective.
3. The Semantic description allows dynamic service integration. It means that the binding of the services of the system can be performed at run time. Such binding follow rules provided by (or reasoned from) an ontology. Integration scenarios of static integration solutions have to predefine every detail. This allows automatic propagation of the system without some manual scenario alteration. If some equivalent and faster service was added, this information as described in ontology will be taken in account while reasoning which will conclude with use of the better one.

This alternatives are more described in [Korotkiy and Top 05], where ontological expansion of SOA, name *Onto-SOA* was declared. With such semantic

description of messages, services and whole architecture, broad possibility of reasoning is open, such as web service replacement (with equivalent one, or some collection providing same functionality).

Standard semantic caching can be complemented with further semantic-architectural caching, when we change web service in the system (described by ontology, in system as predicate) with its declarative form.

8 Cache Study: Caching in Waste Site Energy Management Calculator

We started the collaboration with U.S. EPA (Environmental protection Agency to develop energy management decision support tool [Pavlovič and Mahutová 04]. One of the results of this mutual collaboration is on-line ICT tool: Waste Site Energy Management Calculator. Using this tool, domain experts can compare several remedial technologies according to input constraints. Each remedial technology is represented by an energy equation developed by U.S. EPA. This decision support tool can find the optimal combination of remedial technologies that should be used in cleaning process. Since there exist vast number of technology combinations performance is important. With designed semantic caching we can significantly improve the computation performace. System is using Genetic Algoritm and Genetic Programmig to calculate the best sequence of techologies used in remedial process [Pavlovič and Hřebíček 06].

8.1 Service-Oriented Architecture Implementation

We used Open ESB engine [6] that implements an ESB runtime. Open ESB allows us to easily integrate applications and web services as loosely coupled composite applications. This allows us to seamlessly compose and recompose our composite applications, realizing the benefits of a true SOA.

An ESB can be used to intelligently and reliably route data from the backend sources to the cache service. Because the ESB is coordinating the data flow between the applications and the cache service instead of just being used as a message bus, a variety of backend technologies can participate in publishing their data using the connection interface that best suits their needs.

On the Figure 7 is described the implementation schema of Waste Site Energy Management Calculator. Main part of the system are connected to the ESB as JBI containers. ESB has the role of data controler (middleware) and sends the computation requests to the semantic cache, which is connected as JBI container as well. Data are sent to the another JBI component: computation cluster. And the result is process to the visualization component.

[6] `https://open-esb.dev.java.net/`

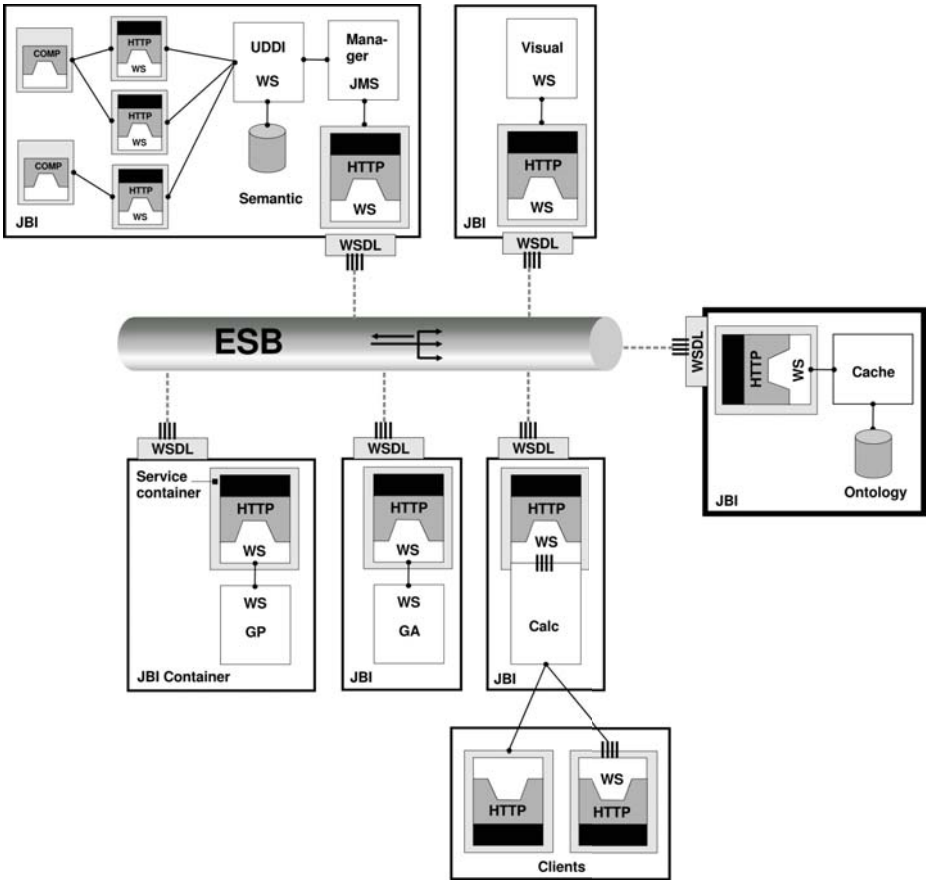

Figure 7: Implementation Schema

8.2 Algorithm Implementation

Since the system is using GP to compute the best sequence of remedial technologies and the GA to find the optimal technology parameter setting the caching algoritm has two parts. The cache is searched for the full match in the first part. Then if the full match is not found algoritm tries to find partial match according to the ontology for definitions of remedial technologies. Each technology in the sequence is searched in the cache using defined semanatic caching methodology. If the search in the cache fails the technology has to be computed.

Architecture of replacement algorithm is based on two queues which are in the same cache memory. The first queue (Q_1) is simply organized using a FIFO strategy. Every XML message which is requested for the first time is inserted

Initialize
$L \Leftarrow 0$

for each request to a message q **do**
  if q can be answered from cache **then**
    if q is answered from one XML message m **then**
      if $m \notin Q_m$ **then**
        bring m into Q_m
      end if
      $H(q) \Leftarrow L + n * latency(q) * \frac{size(q)}{size(m)}$
    else
      for each message m_i answered q **do**
        if $m_i \notin Q_m$ **then**
          bring m_i into Q_m
        end if
        $H(m_i) \Leftarrow L + n * \frac{size(q)}{size(m_i)}$
      end for
    end if
  else
    while there is not enough free space in the cache **do**
      if $\frac{size(Q_1)}{size(Q_m)} > P$ **then**
        remove m from bottom of Q_1
      else
        $L \Leftarrow min(H(m) \mid m$ is in the $Q_m)$
        remove m such that $H(m)$ value
      end if
    end while
    bring q on top of Q_1
  end if
end for

Q_1, Q_m - cache memory queues
P - proportion of queues
$latency(m)$ = downloading latency, $size(m)$ = message size, n = reference count

Algorithm 2: Replacement Caching Algorithm

into queue Q_1. If an XML message is requested for a second time while it is still contained in Q_1, the object is considered as a hot spot and is moved to the other queue Q_m which is organized using the GDSWS strategy. Every time an XML

message contained in Q_m is requested, the corresponding entry is moved to the top of the queue. Objects reaching the tail of Q_1 or Q_m are removed if memory is required for new objects. Which queue is selected for deletion depends on the proportion P size of Q_1 to Q_m. In implemented cache is used proportion:

$$P = \frac{maxsize(Q_1)}{maxsize(Q_m)} = 1$$

9 Conclusions

We described our research in this area. Caching in SOA brings a lot of benefits (and also some disadvantages) in the e-science computing. We used the described approach in VEZMU search engine [Pavlovič et al. 05], where document rank is computed this way. SOA brings lot of possibilities how to access the functionalities of the system for other application.

Another project particularly using ideas of this architecture is Waste Site Energy Management Calculator [Pavlovič and Mahutová 04]. There is massive use of Parallel Genetic Algorithms in the new version of this system. And the SOA encapsulation enables integration of this system to the existing environmental IS.

This approach was also used in project SVOD[7] [Dušek et al. 05]. It is a portal built up on very representative database of National Cancer Register. Nowadays the database consists of more than 1 200 000 cases stratified according to main risk factors. The automated system of on-line analyses is very computation demand. By this reason the system is realized as a set of web services with shared proxy cache. The portal is available at http://www.svod.cz.

Acknowledgements

This paper has been supported by the Czech National Program Information Society "E-learning in the Semantic Web Context", grant No. 1ET208050401.

References

[Andrews et al. 03] Andrews, T. et al.: "Business Process Execution Language for Web Services Version 1.1"; `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`, 2003.

[7] Research and developmental teams of portal SVOD are granted by research project MZO 00209805 solved in Masaryk Memorial Cancer Institute, Brno. Risk assessment analyses are supported by research project INCHEMBIOL, Ministry of Education Czech Republic project no. 0021622412.

[Battle et al. 05] Battle, S. et al.: "Semantic Web Services Framework (SWSF)"; Overview Version 1.0, http://www.daml.org/services/swsf/1.0/overview/, 2005.

[Boyer 01] Boyer, J.: "Canonical XML, W3C Recommendation"; http://www.w3.org/TR/xml-c14n, http://www.ietf.org/rfc/rfc3076.txt, 2001.

[Chappell 04] Chappell, D.: "Enterprise Service Bus: Theory in Practice"; O'Reilly Media, 2004.

[Chen et at. 02] Chen, L., Rundensteiner, E., A., Wang, S.: "XCache - A Semantic Caching System for XML Queries"; Proc. SIGMOD Conference, 2002.

[Chidlovskii and Borghoff 00] Chidlovskii, B., Borghoff, U., M.: "Semantic Caching of Web Queries"; VLDB Journal 9(1): 2-17, 2000.

[Dušek et al. 05] Dušek, L. et al.: "The National Web Portal for Cancer Epidemiology in the Czech Republic"; Proc. ENVIROINFO BRNO 2005, 2005.

[Gašević et al. 06] Gašević D., and Djurić D., and Devedžić V.: "Model Driven Architecture and Ontology Development"; Springer; ISBN: 3-540-32180-2, 2006.

[Giunchiglia and Walsh 92] Giunchiglia, F. and Walsh, T.: "Tree Subsumption: Reasoning with Outlines"; Proc. 10th European Conference on Artificial Intelligence, 1992.

[Korotkiy and Top 05] Korotkiy, M., Top, J.: "Onto, SOA: From Ontology-enabled SOA to Service-enabled Ontologies"; http://www.cs.vu.nl/~maksym/pap/Onto-SOA-WEBSA.pdf, 2005

[Král 05] Král, J., Žemlička, M.: "Service Orientation in Environmental Information Systems"; Proc. ENVIROINFO BRNO 2005 - Informatics for Environmental Protection, 2005.

[Kubasek et al. 06] Kubásek, M., Pavlovič, J., Gregar, T.: "Performance Solution of SOA Infrastructure for Knowledge Computing"; Proc. 6th International Conference on Knowledge Management. Austria, 2006.

[Lee and Chu 01] Lee, D., Chu, W.: "Towards Intelligent Semantic Caching for Web Sources"; Journal of Intelligent Information Systems 17(1): 23-45, 2001.

[Lock and Sommerville 05] Lock, R., Sommerville, I.: "A QoS Ontology for Service-Centric Systems"; Proc. EuroMicro2005, 31st EUROMICRO CONFERENCE on Software Engineering and Advanced Applications, 2005.

[Miklau and Suciu 02] Miklau, D., Suciu, F.: "Containment and equivalence for an xpath fragment"; Proc. PODS, pages 65-76, 2002.

[Nitzsche et al. 05] Nitzsche, T., Mukerji, J., Reynolds, D., Kendall, E.: "Using Semantic Web Technologies for Management Application Integration"; Proc. Semantic Web Enabled Software Engineering, 4th International Semantic Web Conference, 2005.

[OBrien et al. 05] O'Brien, L. et al.: "Quality Attributes and Service-Oriented Architectures"; Technical Note, Software Architecture Technology Initiative, 2005.

[OWL-S 04] OWL-S: "Semantic Markup for Web Services"; http://www.w3.org/Submission/OWL-S/, 2004.

[Pavlovič et al. 05] Pavlovič, J., Pitner, T., Kubásek, M., Svoboda, L.: "Searching and E-learning System above Digital Sources"; Proc. Sharable Content Objects, 2005.

[Pavlovič and Mahutová 04] Pavlovič, J., Mahutová, K.: "Energy Management at Waste Clean up Sites, Avoiding Secondary Air Impacts to Human Health"; Proc. First Biennial Central & Eastern European Environmental Health Conference, 2004.

[Pavlovič and Hřebíček 06] Pavlovič, J., Hřebíček, J.: "Finding Optimal Solutions of Energetic Remedial Equations with Genetic Algorithms"; Proce. Enviroinfo2006 Conference, 2006.

[Pondrelli 05] Pondrelli, L.: "An MDD annotation methodology for Semantic Enhanced Service Oriented Architectures"; Proc. CEUR Workshop, 2005.

[Ramesh and Ramakrishnan 92] Ramesh, R. and V. Ramakrishnan, L.: "Nonlinear pattern matching in trees"; Journal of the ACM, 1992.

[Zaki 02] Zaki, M.: "Efficiently Mining Frequent Trees in a Forest"; Proc. ACM SIGKDD, 2002.