Self- Describing globally accessible software components

Igorco Pandurski, Marjan Gusev

Institute of Informatics, Faculty of Natural Sciences, Skopje
Republic of Macedonia
pandurski@ii.edu.mk, marjan@ii.edu.mk

Abstract. The last decade, the world had been introduced with multiple and different Web technologies and development paradigms. The main idea-driver behind that boom is to create globally accessible software components. Web Services and the Web agents are the most utilized software components and are considered as self-describing and self-contained. While using meta-data containers interpreted via XML structure they are considered as self-describing and self-contained. But the question is, are they really self-describing and what are the all aspects that needs to be considered while self-describing labeling is taking place?

Keywords: XML, Functional programming, WEB Services, RDF, OWL, Semantic web, fXML, Semantic Web Services

1 Introduction

Globally accessible software components are considered all software components that possess an exposed interface, usually on the Web, and while using specific industry based standards and protocols can be embedded or integrated with other software systems that are independent from the location or the technology behind. With that, the logic or the business processes captured inside these software components are becoming an integral part of the new system.

In order to be able to perform such integration, the software components must possess features to be discoverable by the possible integrators. The model behind making the resources discoverable, today, is consisted of various architectures, technologies and protocols. The most known architecture is SOA (Service Oriented Architecture) and the technologies include Web Services, UDDI (Universal Description Discovery and Integration), WSDL (Web Service Description Language) and the SOAP (Simple Object Access Protocol). The sharing feature of all the after mentioned technologies is the XML (eXtensible Markup Language).

Having this in mind, the scenario goes like this: 1 A new system needs to integrate a function,2 The resources that poses that function are exposed on some of the UDDIs or the location is known 3 The new system lookups for the most appropriate resources that already posses that function checking up the WSDL file which posses the information about that function; 4 Integration is performed and specifically formatted messages are exchanged using SOAP and the paradigm behind globally accessible software components is fulfilled. Independently of the technologies behind every component of the chain uses XML structure as a markup language used for collaboration.

All this sound perfect and possible. And it is. Many of today's web resources are acting on that way very successfully. So, where is the challenge?

Well, the demand today for new web services that are incorporating complex functions in their structure, is rapidly increasing. WEB 2.0 based sites are requiring more than a service that calculates the currency exchanges, or conversion of Celsius degrees in Fahrenheit. They require some intelligence incorporated within, they need to provide you with an info which bank has the best exchange rate so the consumer can gain more benefits or, to suggest when to perform the exchange itself having in stack the stats of the currency flow in the last period.

The demand for exposing the currently incorporated business functions (processes and rules) in the legacy systems and making them globally available is also in search for appropriate exposure, having in mind the factor of global economy and entities mergers increases on a daily basis.

Incorporating a partial functions form one resource and partial functions from another while creating a new one is another hot topic.

Finding the right service that provides the right function is the real challenge. Lately, finding the right set of functions that can provide end-to-end process implementation in the application layer globally available is more suitable to be said.

2 Model for Self-describing Web resources

In the past, the software components written in different languages could have been very hardly integrated with other software components. There were too many obstacles that needed to be overrun in order to make this happen - complexity, preservation models, specific structural requirements, accessibility (communication patterns), proper ways of exposing the functions captured inside etc. There were too many initiatives and ideas how to make this happen. One of the streams was to develop a unified markup language and all the development frameworks and paradigms to develop a support for it. That is how the XML was born. Once the XML emerged on the surface, it was really lightweight document-like language and the rest of the obstacles were still remaining uncovered. The preservation model and the complexity behind the software components had to be addressed if all the business functions behind the specific software components want to be exposed and available for integration [Software as a Service (SaaS) paradigm] . The XML was about to be expanded with support for this as well.

Once this all was covered from one side, and from other side Web emerged as a global area where the software components are distributed and incorporated, mostly

as a Web Services, the real need and challenge become finding the exact function or set of function that you need. That means that the software components need to be self-describing and self-contained and the way how this can be done is adding new feature to the components – metadata. Metadata is binary information describing your software component that is stored either in a common language or is inserted into one portion of the file that is accessible to all the parties. In more technical jargon, metadata stores info about the identity (component, version, public key etc), the types that are exported, security permissions, attributes (base classes, methods, fields, nested types, elements etc).

Once the metadata is processed by the consumers, new challenge is faced again- what is the preferred method of processing of all the consumed web resources, and what is the real meaning, the semantics of the output. To address this challenge, a new model of the self-describing and the self-contained web resources is needed. That model is the Semantic Web Services and the Semantic fXML.

2.1 Self-Describe labeling

The label of Self-describing in order to be put as an attribute to a specific software component or service, the service itself need to go through a specific maturity model. It needs to support flexible explorations and interpretations while the information behind are comprehensive and enough explanatory for the consumers. On the figure 1 is explained the maturity model of a specific software component that need to fulfill before it is considered as a self-describing: to be standard machine readable, to support widely deployed formats, to inhere machine-processable specifications, to be grounded in web, to convey the RDF triple and finally a standard HTTP based algorithm to be used while deploying.

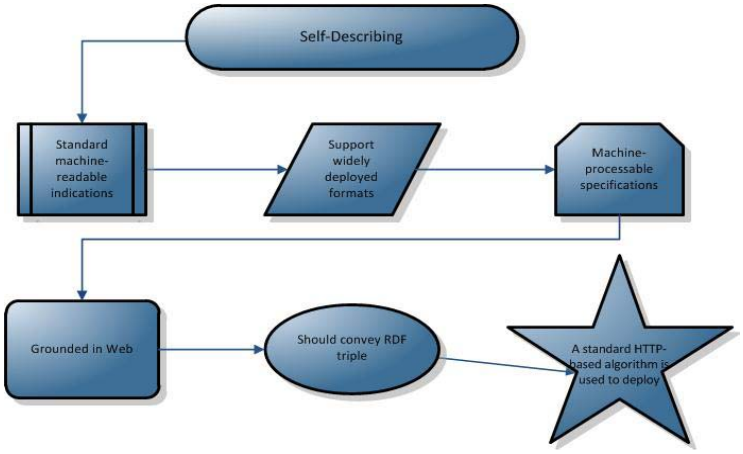

Fig. 1. Self-Describing Maturity Model

In particular, every stage means: [1] standard machine readable indications are the standards and conventions to be used , like XML encoding, HTTP content types

headers etc; [2] support for widely deployed formats- text/html, image/jpg; [3] machine- processable specifications – OWL ontologies; [4] grounded in web – all the specifications needed for proper interpretation of the information should be discoverable over recursively used links starting with the initial URl; [5] should convey the RDF triple (subject-predicate-object) – or simplified, the subject is denotation of the resource, the predicate is the denotation of the aspects of the resource and the object is the relationship between the subject and the object.

2.2 XML, is it really self-describing?

Having in mind the maturity model previously explained, the XML as we know it, is it really self-describing markup language? It is suitable to fulfill all the needs for publishing and linking the Web resources?

The elements and attributes provided in the XML structures are self-describing to some extent. They are providing the information for the usage, data types validated against a XML Schema and with that enforcing specific constrains. But the tags itself, doesn't give enough info about the semantics of the data, validation will pass and the XML document will be valid but you may still not getting the right data. Harry Halpin and Henry S. Thomson [1] are giving very visual explanation: Tag names are not enough. If a document is shared between two or more parties, there is an assumption that the meaning of the document is implicit is some common understanding (Halpin & Thompson, 2006).

Today, with the global market expansion, multiple integration projects are emerging and the parties are not even aware of each other. Agents, on behalf of the requestor go on the Web, searchers for function that he need within all the available Web Resources and the integration starts. So, if the XML based documents (WSDL files or RDDL documents) doesn't include all the information then the requestor will never know if this function or the set of functions are the right one he needs.

Therefore, XML as we know it, it is safe to be said that is not self-describing.

2.3 fXML

To make XML to be a self-describing structure, or at least to extend the level of self-describing, again, Harry Halpin and Henry S. Thomson are suggesting embedding functional programming structures in the XML. This new approach it is suggested a usage of an fx namespace where some of the elements and the attributes are no longer just data carriers but becoming functions or processors. That enables within the XML structure to embed variables, conditions, functions.

The intent of this paper is not to go in details and the ups and downs of this approach but to provide a view to all the efforts in making the XML really self describing. Therefore, we are saying the fXML is an approach that with attaching processors to the XML documents the level of self-describing is raised and the requestors can expect more quality based data that are matching the needs to acceptable altitude.

3 Semantic Web Services

Now, when the XML has reached the acceptable altitude of self-describing, the practice needs this in reality. Web Services, which are currently widely examined, researched and explained, are in their flourishing phase. The business are exposing their business functions (processes and rules) in Web Services and all the clients, independently if they are internal within the enterprise, or external and are located 10 000 miles away or next door, are consuming them just very easily. New systems that increase the productivity are emerging daily and are generating a new value for the owner.

Based on my experience, still these integrations are based on a mutual agreement between the parties, exchange of the necessary files, types and locations independently are we using cutting-edge technologies for implementations or not. Something is still missing. The human factor probably will be never excluded, but the human factor slows down the integration, increases the probability of errors and finally, humans are subjective, always approaching and addressing the challenges on the easiest way. The easiest way, I hope you would agree, is not always the most productive, the most stable en cetera. Actually, my experience is saying that this is almost never the case.

The idea is to expose all the business function into a "pool" of functions that are enough self-describing and then the web crawlers or the web agents to come, perform the search for proper function and do the integration. The human factor will be involved just in exposing the current functions into web resources and deploy them to the "pool". On the top of it, all the corporate, governance and security policies will be applied so all the confidential and business related secrets will be prevented from abuse.

As of now, the portion of available technologies and techniques and the paths of the search for proper concept end on the Semantic Web Services.

The most sustainable definition of the Semantic Web Services and in the same time enough self-explanatory without confusion is the one provided by the Semantic Web Services Initiative Organization [2] which is in the same time their mission - Infrastructure that combines Semantic Web and Web Services technologies to enable maximal automation and dynamism in all aspects of Web service provision and use, including (but not limited to) discovery, selection, composition, negotiation, invocation, monitoring and recovery. The only missing part in this definition is the intelligence. Semantic Web Services are distinguished by the remaining forms of Web Resource according one feature- The intelligence.

What is the relationship between the Semantic Web Services and the other forms of Web Resources?

In order to answer that question on Figure 2 is given an abstract relationship between the technologies and the domains that are considered as a Web Resources. The story starts with the expansion of the Web, when form human-centric environment transcended to software components-centric. This happens when the Static Web with its basics of HTML used for marking up documents and HTTP transferring them over the web, started to be shifted toward dynamic and opening the possibility of semantic way of marking up the documents which is centered on the previously mentioned metadata, ontologies, logic and autonomy. From this shifting;

web services were the result, which in a simplified manner are considered as software components that supports universal interoperability.

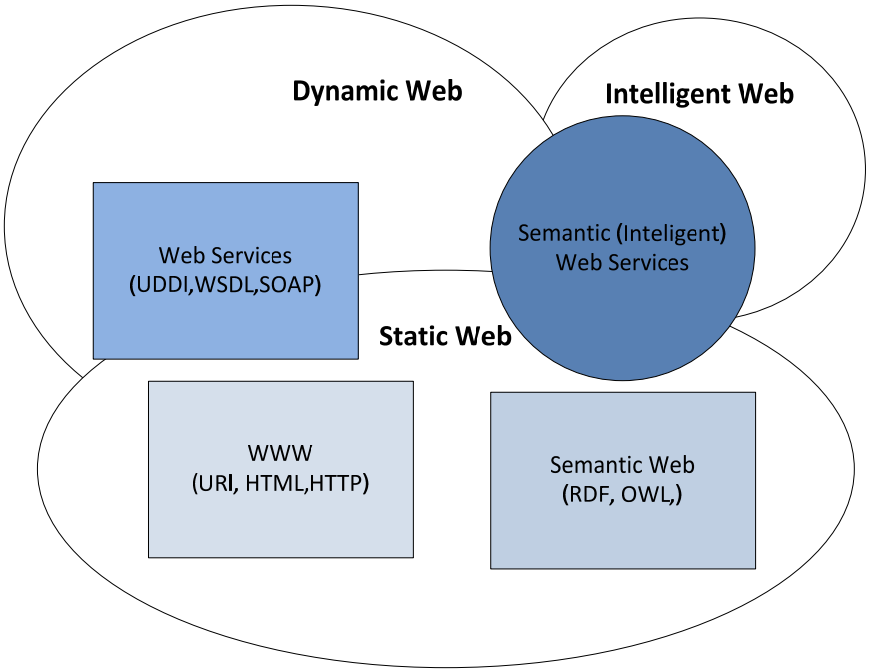

Fig. 2. Semantic (Intelligent) Web Services

Now, the missing link: the intelligence. Once the shifting form static to dynamic domain of the web became stabilized, the intelligence as an attribute was the new challenge.

Jagadeesh Nandigam [3] explains this challenge as synergetic confluence of the Semantic Web and Web Services which have the potential to provide value-added services by autonomously discovering and assembling Web Services to accomplish a domain task.

The way how this challenge is addressed we will explain the two main drivers that enable the addition of the Intelligence to the stack- RDF and OWL.

3.1 RDF

The shifting from Static Web towards Dynamic Web and lately towards Intelligent Web was in need of more suitable way of marking up the documents an enable the data interchange much easily and in the same time enable the implementation of the semantics. RDF (Resource Description Framework) is exactly that- it enables the data interchange even the data schemas are structured, semi-structured or even different.

The most suitable definition for RDF according my perception is the one that RDF is the language used for representing the metadata of the web resources.

The concept behind the RDF is having a URl (Uniform Resource Identifier) for every property and property value of the resource which when interpreted, statements about the resource as graph of nodes is generated as shown bellow.

```
<?xml version="1.0" encoding="utf-8"?>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"

xmlns:paper="http://www.pandurski.com/paper#">

        <paper:Author
          rdf:about=" http://www.pandurski.com/about">
          <paper:fullName>Igorco Pandurski</paper:fullName>
          <paper:mailbox
rdf:resource="mailto:pandurski@ii.edu.mk"/>
          <paper:personalTitle>Mr.</paper:personalTitle>
          <paper:conference>ICT2010</paper:conference>
        </paper:Author>


      </rdf:RDF>
```

This piece of code can be interpreted as graph with multiple nodes:

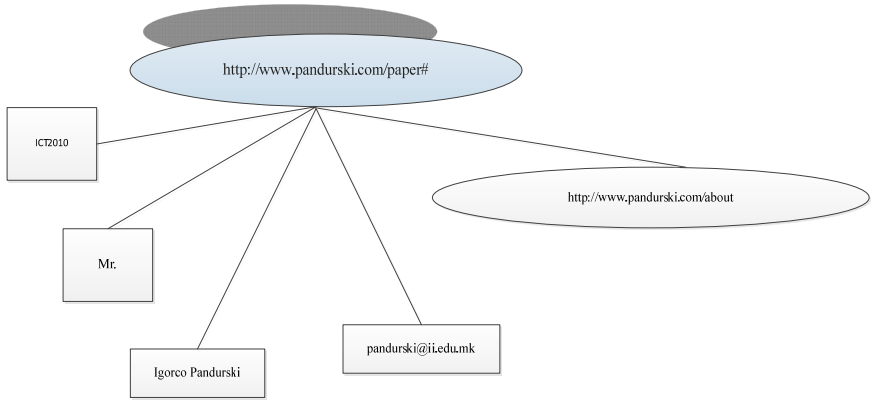

Fig. 3. Semantic (Intelligent) Web Services

This primer actually confirms that the RDF is an extension of the XML and the structure itself is even called RDF/XML.

3.2 OWL

Following the shifting from Static Web to Dynamic and lately to Intelligent Web the metadata is covered with RDF; it remains to cover the ontologies, the logic and the autonomy as paradigms.

Ontology is a visualization of the concept of the domain where the specific resources exists. In this case Web Resources. This visualization of the concept is nothing else but capturing the taxonomy of the environment and expressing it thru a machine readable language.

OWL (Web Ontology Language) is a language developed for expressing the ontologies while using URl for naming and description framework. OWL posses few more distinctive features: ability to be distributed across many systems, it is scalable, it is compatible with the Web Standards (many other ontology languages are not), it is open and extensible. OWL is build on the top of XML and RDF with expanding the vocabulary which can very easily deliver the semantics – relation between the classes' types of properties, equality, cardinality etc.

We are not going to examine the gods of the OWL language but we will show how the ontologies are build- form declaring the namespaces, building simple classes and defining properties:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"

xmlns:paper="http:"//www.pandurski.com/paper#
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#"
         xmlns:xsd
="http://www.w3.org/2001/XMLSchema#">


 <owl:Ontology>
    <rdfs:comment>An example OWL
ontology</rdfs:comment>
       <owl:imports
rdff:resource="http://pandurski.com/document"/>
        <rdfs:label>Sample Ontology</rdfs:label>

        <owl:Class rdf:ID="Paper">
        <rdfs:subClassOf
df:resource="&document;PotableDocument"/>
        <rdfs:label xml:lang="en">Self-Describing
globally accessible software components</rdfs:label>
          <rdfs:subClassOf>
              <owl:Restriction>
```

```
                    <owl:onProperty
rdf:resource="#canBeDistributed"/>
                     <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCard
inality>
                  </owl:Restriction>
              </rdfs:subClassOf>
            </owl:Class>

  </owl:Ontology>
```

Once the ontologies are built, multiple, autonomous software agents can be build that can crawl over the web resources and provide the requestor with specific function or set of functions. Having in mind the previous ontology it can be a part of a Web Resource that stores scientific or best-practices papers and blueprints. Than the web agents can come autonomously, search over the ontologies and find the right document that treats the self-describing as subject.

4 The vision for Globally Accessible Software Components

The most interesting part from always for me was- how to implement the achievements in the science into the real life world and generate a new value for the clients. Being a part of the happenings within the financial world in US and being a leader of some of the biggest integration software projects , the limitations of the technologies used for building the legacy systems, the time consumption in creating appropriate interfaces for the functions behind and finally integrate the systems is frustrating. The projects itself has a scope, has a timeline and has a stakeholders. When I am saying stakeholders I mean a limited budget. Now, I am facing legacy systems that are robust, non-flexible, non scalable, distributed, practically indefinite scope and a limited time and budget. Whatever you decide to play with, you will face a wall. If you are on time with building the interfaces of the functions than, you have no time and budget for fully implementations and integration with the other systems. Or if I put a balance between all the three peers, you will never succeed in fulfilling the scope and the stakeholders will go crazy.

Therefore, while the projects are ongoing I am in a search for appropriate and cutting-edge technology or technique that will provide me with abilities to safe a time on creating the interfaces, increase the portability and quality and of course, to satisfy the scope.

That is how the ideas of generating a “pool of functionalities” are born- The pool will represent all the functions captured behind the legacy systems and enable them for the software agents to be searchable and integral, independently if the poll is going to be clouded within the company, on the public web or some of the registries available today.

Once this criterion is satisfied, every integration will be much faster, will assure that the scope is fulfilled, the timelines will be preserved and the numbers of

integrations is practically unlimited. Re-usability of the code is another great feature of the Web Services, in this case Semantic Web Services.

The market today goes thru a phase of global merger and consolidation; companies with a similar core business are merging and the data including the functions aggregated within the decades are enormous. Very often, the core-business is not within the software industry, but always are depending on the software and the paradigms of the software – Enterprise Application Integration (EAI) or Inter-organizational systems (IOS).

The globally accessible software components are the most suitable way of performing (1) Legacy-Systems Integration, BPI (Business-Process Integration) and Software as a Service (SaaS) implementations.

4.1 Legacy -Systems Integrations

"Over the years, IT environments have become more complex and more heterogeneous due to diverse customer needs and rapid innovation in the IT industry. Many government and enterprise customers require integration with legacy systems to maintain critical business and operational processes." – Microsoft [4].

This definition explains very visually what the legacy-systems integration all about is. The integration is usually performed, but not limited to, due to potential problems and non-compatibility with the emerging trends, improvements, maintainability, replacement and the lack socially addictiveness. At this time, having in mind the shimmering budgets [5] for the ICT, the Semantic Web Services are the key players in this legacy –systems integrations.

The systems that are integrating are starting from mainframes ending up to corporate websites. In between pleads of applications (both, web oriented and desktop) are passing thru this phase.

4.2 Business Process Integration (BPI)

With the legacy-Systems integration many of the challenges that today's business are facing are addressed- transparent and real-time access to information for the entire enterprise. The functions captured inside are exposed and the business continues to use the functions form all the systems behind. Now, having the functionalities, every new merge requires accommodation of specific business processes and business rules. To avoid the processes duplication and inconsistency which can result in ineffective and inaccurate decision-making, business process integration takes places. How the Semantic Web Services are making this happen?

Well, the "pool of functionalities" first eliminates the silos of stand-alone applications. Now they are all connected. On the top of the pool, BPEL process (Business Process Execution Language) is applied. The BPEL process has the complete same pattern as the Web Service, except that has a possibility to embed graph-like regimes, manipulation and decision-making functions and transactional processes. The BEPL process can embed different Web Services or different Semantic Web Services and apply composition or orchestration over them and with

that specifies and exact order of services invocation and satisfies the desired business process workflow (Figure 3).

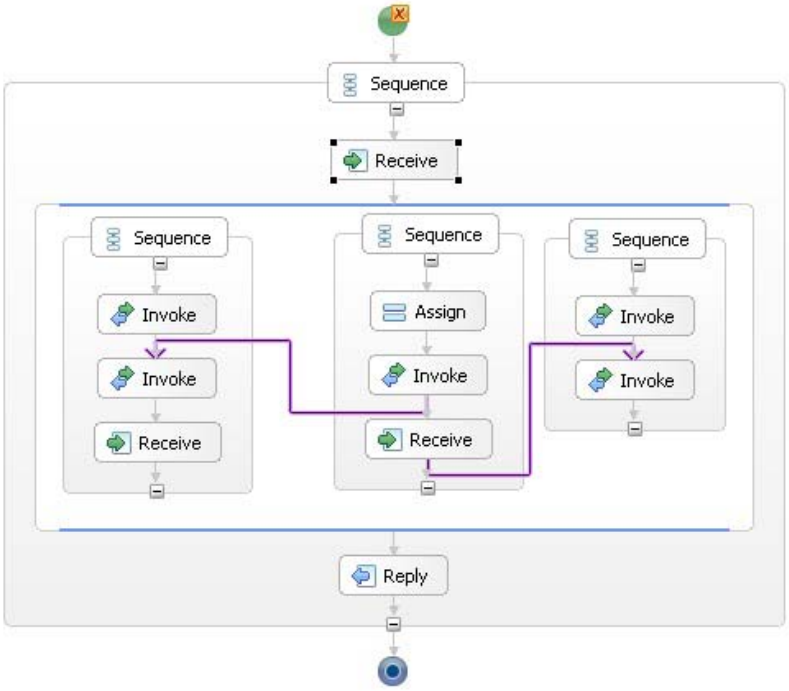

Fig. 4. BPEL Process

4.3 Software as a Service (SaaS)

One of the hottest topics on the Google I/O 2010 [6] was the SaaS topic. In general terms SaaS is software components deployed on the Internet. That concept is nothing new, but what distinguish the SaaS from the other Web Resources is that gives end-to-end execution. The components are accessed remotely and the benefits are that no infrastructure or specific deployment is needed.

On the Google I/O 2010 was exposed a new HTML5 standard that the entire browsers family are going to implement it in the near future. Most of them are already supporting it. For the business SaaS means availability of business applications, including collaboration software and line-of-business applications that require them to run their business without an infrastructure or specific development. Most of the times it SaaS are commercial solutions and all the possible clients need is an Internet and appropriate browser.

This achievement incorporates all what is stated in this paper and it is subject of future deeper analisis.

5 Future work and conclusions

This paper should give us the idea how the Web Resources while using the basis of XML are going to real self-describing attribute and are passing the maturity model. Once the maturity model is satisfied, all the pre-requisites are achieved and the shifting of the Static Web towards Intelligent via Dynamic is possible. This progress flow of the technologies, techniques and software paradigms are implemented in the real life world and are enabling the business to move forward.

With that said, it is safe to be concluded that the Semantic Web Services are the solution in need of a problem.

The Semantic Web Services and the properties they pose can be considered as really self-describing globally accessible software components.

The future works are to examine the benefits of applying SaaS model in the real business and what is the scientific approach behind it.

References

1. Halpin, H., and Thompson, H. S.: One Document to Bind Them: Combining XML, Web Services, and the Semantic Web. Proceedings of the 15th international conference on World Wide Web, 2006
2. The Semantic Web Services Initiative (SWSI), http://www.swsi.org/
3. Nandigam, J.: School of Computing and Information Systems, Grand Valley State University, Allendale, MI, 2005
4. Legacy System Integration, Link
5. Gartner EXP Worldwide Survey, http://www.gartner.com/it/page.jsp?id=1283413
6. Google I/O 2010, http://www.google.com/events/io/2010/