Is Semantic Web Technology Taking the Wrong Turn?

Christoph Bussler

Developers create Semantic Web technologies (SWTs) not only to overcome the syntactic, semantic data heterogeneity problem at design time but also to support uniform agreements on the meaning of data and processes at run time. As defined in the landmark article in the space[1] (and reprised in *IC*'s Peering department in 2007[2]), the greatest promises of the Semantic Web and SWTs are

- seamless interactions among agents (people and services) based on reliable communications and uniform data and process semantics;
- a solution to the heterogeneity and interoperability problem in data and processes via dynamic and automatic discovery and integration; and
- semantic correctness and dependability (including trust and explanation of reasoning results).

That article clearly outlines the data- and process-integration challenges between humans and computerized services in the form of a medical scheduling process for organizing regular doctor visits involving a patient and her family. In addition to these requirements, Tim Berners-Lee, Jim Hendler, and Ora Lassila outlined nonfunctional requirements such as trust or the on-demand explanation of formally derived reasoning results.[1] They put forth the Semantic Web and its related technologies as a possible solution to the outlined requirements — first and foremost, the problem of semantically correct and consistent data and process integration.

In response to their article, a whole academic research field emerged, complete with annual conferences, and industrial software development began encompassing research results into new commercial products (see the "Ongoing Work in Semantic Web Technologies" sidebar). Equally impressive is the fact that companies and research funding organizations, such as Darpa, the European Commission, and various Asian funding organizations have invested significant effort and money into research and industrial projects. In all, this upcoming computer science field looks very successful, and its future seems bright.

Yet, the results are much less impressive in terms of actual applications. An architectural analysis suggests why and that SWT is headed for a disaster unless there is a change of course.

Requirements for Semantic Web Technology

One way to take stock of an area of computer science's overall achievement is to categorize academic research results and industrial technology products and plot them on a timeline. If we followed this approach with the Semantic Web field, the visible progress would be impressive. The number of industrial technologies, standards, publications, research prototypes, workshops, and conferences in the space is vastly increasing (apparently almost exponentially), as are the number of ontologies being discussed, developed, and supposedly used in real settings and applications for managing real information.

However, this approach defines progress in terms of the work achieved (the more, the merrier) rather than the degree to which real-life problems are being solved (Berners-Lee, Hendler, and Lassila emphasized the real problem space for solving real problems[1]). In this context, the goal isn't simply to solve the problems (which could be achieved via "conventional" software technologies) but rather to solve them at least a magnitude "better" with SWT.

On a higher level, the core problem in the medical patient example is about distributed calendar scheduling. Three people must coordinate their calendars (two family members and a physician) to schedule treatment for a third family member. As the first two take the third to and from the physician's practice, driving distance and time of day play big roles. Moreover, the family can consider only physicians within its healthcare provider's network.

As derived from the article, SWT's major concrete requirements are

- data interpretation and mediation (between the calendars of the family, physicians, and hospitals, as well as routing planners and traffic-monitoring systems);
- process interpretation and mediation (between calendar systems, hospital systems, and public traffic information systems);
- data storage and retrieval (calendar state, physician quality ratings, and traffic patterns);
- business logic execution based on data (selecting appropriate times based on availability constraints and physicians' eligibility); and
- agent interoperability (all the involved systems have to interact).

In principle, the example calls for a software application that works with any number of calendar systems of any make and model and any number of software applications used by physicians, hospitals, and healthcare providers, and it should be available to everyone with Internet access — not just in the US, but in all countries. And, of course, traffic-monitoring and status-reporting systems must also be integrated. In this sense, the system must be extremely open, extensible, and dynamically changeable forever, which implies that the requirements implementation doesn't have a fixed end (as in many software development projects). This software application's enormous complexity becomes clearer as we consider all the possible combinations.

With my background in agents, enterprise application integration (EAI), and business-to-business (B2B) technologies, I've attempted to measure SWT's success by analyzing the overall situation in the context of a concrete application example. In contrast to the standard approach in research publications, I use the requirements stated by the original authors, although I take a much more engineering-oriented viewpoint.

Conventional Application Architectures

Interactive software applications supporting end users like those discussed here generally have at least seven architectural layers (of course, variations exist):

- graphical user interfaces (GUIs) in Web browsers,
- user interface logic drivers,
- business processes,
- business logic implementations,
- business rules constraining valid operations,
- a persistence layer, and
- storage systems for storing and recalling data.

These seven layers execute any successful user request on the GUI, and any response travels through them all on the way back to the GUI — 14 layers in total.

Developers can use many current software technologies to implement these layers (see the sidebar for examples). In our context, it's noteworthy that each of these technologies has a data-representation as well as data-interpretation model, and a notion of execution in terms of handling requests at runtime. These properties are independent of the particular business problem to be solved. Yet, even before business-specific requirements and challenges arise, merely using these technologies presents a heterogeneity challenge because the layers must interact with each other. Consequently, the data structures must be mapped or transformed between the layers when executing end-user requests.

In addition to the seven layers, GUI-based end-user applications feature two distinct cases of remote integration[3] — intra-enterprise integration (also called EAI) of (end-user) applications, and inter-enterprise (or B2B) integration. The distributed calendar scheduling example covers all aspects of industrial software application architecture, which makes it extremely relevant in the context of SWTs.

In contrast to GUI-based applications, B2B applications use a B2B communication layer that knows how to remotely communicate data and processes. B2B communication includes the seven layers at both trading partners, and if a communication requires an acknowledgment or return message, it must cross the layers as well, bringing the overall count to 28 layer crossings in a single request–reply communication.

Integrating with Conventional Architectures

SWT doesn't propose a different application architecture. Instead, it proposes languages and technologies that are intended to make the application development process and integration efforts a lot simpler, faster, and more reliable, especially in the areas of data and process mediation to achieve uniform semantic interpretation.[1] For good reason, SWT doesn't propose replacing core technologies either; from a pragmatic viewpoint, trying to replace existing database technologies, programming languages, or communication infrastructure would be futile.

Yet, for SWT to have an impact, it must be integrated somewhat with current core computing technologies. Semantic Web services (SWSs) are a good example. SWS technology augments, rather than trying to replace, commercial Web service technology. For example, developers can describe Web service interface definitions via semantic languages such as OWL-S (www.daml.org/services/owl-s/) or the Web Service Modeling Language (WSML; www.wsmo.org/wsml/) rather than the Web Services Description Language (WSDL; www.w3.org/TR/wsdl/) — or in conjunction with it, as in Semantic Annotations for WSDL (SAWSDL; www.w3.org/2002/ws/sawsdl/). Although we can semantically describe Web service interfaces, we (still) implement the Web services using existing (nonsemantic) programming languages such as Java or C#.

Database technology presents a slightly different approach. Oracle implemented the Resource Description Framework (RDF) model directly into its relational database management system (RDBMS) as standard database technology, whereas others have proposed stand-alone database systems, such as Jena (http://jena.sourceforge.net) or Sesame (www.openrdf.org). In either approach, some data will continue to be stored outside RDF structures and RDF databases for some time to come. RDF databases represent all data as triples — in Oracle, it's possible to collocate data in relational form as well as in triples. In contrast, SWT has yet to touch user interface technology. Neither process nor workflow execution environments use SWT at all at this point. The only available mechanism is to refer to data (only) using SWT. For example, developers can use the RDFa standard (www.w3.org/TR/xhtml-rdfa-primer/) to work with HTML pages and XML documents with embedded RDF statements.

To clarify the significance of this approach to integrating SWT with "conventional" software technology, let's look into the details of a Web service invocation, using SWS as an example in describing the interfaces. Let's assume that one SWS invokes another, and to make things a bit more interesting, let's assume that this is a remote invocation in which the communication data is represented in RDF. Let's say that the invoking SWS is implemented in Java and the invoked SWS is implemented in Lisp, which means that the invoking SWS must mediate between Java and RDF and the invoked SWS must mediate from RDF to Lisp after the remote data transport. The mediation includes a syntactic as well as a semantic re-representation because the invoked SWS's interface has a separate definition of its interfaces. In total, three languages are involved and two mediations occur when crossing this one layer (for one direction of invocation!). Because this is the general case for any layer, the extreme but not unlikely case with the 28 layers I mentioned earlier would lead to 56 mediations, as well as up to 28 language shifts involving 14 interface definitions. Although each communication partner has to worry about "only" 14 layer crossings, 7 interface definitions, and 28 mediations, achieving the requirements stated in the *Scientific American* article remains very hard. (And this discussion doesn't even consider the case in which different services are defined in different SWS languages.)

This application's complexity becomes clear as we consider all possible combinations.

In summary, SWT today either works as wrapping technology to enable semantic interfaces for layers or introduces additional component technology alongside existing components, as with databases. Additional component technologies complicate the challenges as more "moving parts" must be integrated in an EAI sense, thus increasing the number of interfaces and mediations required. Additional component technology also turns individual layers into heterogeneous implementations. When SWT is used as wrapping technology, the heterogeneity problem sharply increases the number of data models that require additional mediation. Anyone who doubts that integrating heterogeneous systems becomes more difficult with SWT should try the simple but real examples of the Semantic Web Services Challenge (www.sws-challenge.org).

Disaster Analysis

Many recent publications start by assuming a homogeneous environment (language, ontology). They often begin with statements such as, "we developed an ontology that we use exclusively," "we assume OWL-S as the SWS language," "all data is stored as triples in an RDF store," and so on. Rarely (if ever) do authors extend one or more existing ontologies or assume that services on the Web can be described in any implemented (Semantic) Web service interface-definition languages. Indeed, doing so would tremendously increase the heterogeneity and mediation. Although SWT seeks to address the heterogeneity problem, researchers generally try instead to avoid it by making assumptions or putting constraints in place that give them homogeneous environments. A very noteworthy exception

Ongoing Work in Semantic Web Technologies

The number of research results, software products, and conferences in the area of Semantic Web technologies (SWT) are clear evidence of significant efforts. Substantial development is already evident in formal Semantic Web languages, including the Web Ontology Language (OWL; www.w3.org/2004/OWL/) and Resource Description Framework-Schema (RDFS; www.w3.org/TR/rdf-schema/).

We've seen additional advances in the areas of semantic libraries (JeromeDL; www.jeromedl.org), ontology modeling and management (KAON2; http://kaon2.semanticweb.org), social networks (the Friend-of-a-Friend project; www.foaf-project.org), and databases (Jena, http://jena.sourceforge.net; and Oracle, www.oracle.com/technology/tech/semantic_technologies/).

SWT efforts also include the Semantic Web Rule Language (SWRL; www.w3.org/Submission/SWRL/), the SPARQL Query Language for RDF (SPARQL; www.w3.org/TR/rdf-sparql-query/), reasoners (http://en.wikipedia.org/wiki/Semantic_Reasoner), semantic search (Swoogle, http://swoogle.umbc.edu; Hakia, www.hakia.com; and Powerset, www.powerset.com), and ontology mediation (J. Euzenat and P. Shvaiko, *Ontology Matching*, Springer-Verlag, 2007).

Semantic Web services are particularly popular research and development topics:

- DAML Services (DAML-S; www.daml.org/services/owl-s/);
- Web Service Modeling Ontology (WSMO; www.wsmo.org); and
- Semantic Annotations for WSDL (SAWSDL; www.w3.org/2002/ws/sawsdl/).

Numerous SWT-related conferences cover the topic space. The recently introduced First IEEE International Conference on Semantic Computing (http://icsc2007.eecs.uci.edu) added the dimensions of natural language processing and multimedia object processing to the semantic mix. The industrial Semantic Technology Conference series (www.semantic-conference.com) focuses on the semantic software technology industry, and many researchers and developers show their work there. Several ongoing academic conferences also cover research work and achievements:

- European Semantic Web Conference (www.eswc2008.org);
- International Semantic Web Conference (http://iswc.semanticweb.org); and
- Asian Semantic Web Conference (www.sti2.org/events/events).

Current implementation technologies for conventional industrial software application architectures include the following:

- J2EE (http://java.sun.com/javaee/);
- Apache Java Persistence API (OpenJPA; http://openjpa.apache.org);
- Java (http://java.sun.com);
- Jess (http://herzberg.ca.sandia.gov/jess/); and
- JavaServer Faces (http://java.sun.com/javaee/javaserverfaces/).

Both the SWT and implementation-technology communities are showing progress in achieving their own agendas, but relatively little cooperation exists between the two. Working from a common framework could help bring them closer together and begin to realize some of the promise of the Semantic Web vision.

is the Semantic Computing Research Group's public sector work in Finland, where the whole country works on a uniform ontology across all its industries and companies![4]

A second observation is that the original Semantic Web article[1] assumed at least a homogeneous data representation format in RDF (in conjunction with XML). In addition, the authors assumed that all interacting agents would comply with this constraint, at least at their interfaces. Although this approach wouldn't reduce the number of mediations, it would at least create a common format used between layers and inside applications, as well as across those in the EAI or B2B sense. This fits the W3C's "layer cake" vision in which languages are built on top of each other (extending rather then redefining) in the sense that they increase expressiveness without creating heterogeneity. In a sobering follow-up article, Berners-Lee, Nigel Shadbolt, and Wendy Hall analyzed the progress of the plans set out in the original article, reiterating the goals and emphasizing the need for a more structured architecture as well as data (and process) standards.[5] The role of data standards is noteworthy as their uptake across the layers would begin to significantly reduce required data mediations.

How could technology that set out with the best intentions to simplify the integration and interoperability problem actually lead to the opposite? One possible answer is that the research community and industry took a wrong turn in dividing the whole space according to the classical lines of distinction between layers and components in software architectures, as well as the classical academic research fields. For example, the database community started to apply SWT to databases; the Web service community did the same to its work; and so on. Each community has thus extended its own technology, causing a disaster for software architects and engineers who must use the results from several communities in building software applications that are hosted and interconnected. Some industrial efforts are currently focusing a little beyond the classical layers,[2] but a lot more is necessary.

One possible direction for semantic technology research and development is to continue developing point solutions for individual areas and technology components. Straight ahead from here leads to more SWT languages, hard-to-integrate ontologies, and technology components such as libraries, RDF databases, and logic reasoners. Those who build real-world applications will have to integrate all those elements to use them holistically, thus leaving the integration problem unresolved. As this approach increases the effort required in every part of the software engineering life cycle, chances are that developers will adopt the SWT only for very specific areas and solutions, rather than for general use across all domains in which computing is applied.

One possible turn would be to start addressing the problem of data and process heterogeneity, not only among systems but also among the layers within them to reduce or eliminate the number of mediations necessary. Rather than looking at SWT as interface-wrapping technology, it seems appropriate to make it the foundation for all aspects of information technology and scientific computing. In concrete terms, one way to eliminate mediations when crossing layers is to ensure that data objects are encoded in a single format (such as RDF) and not mapped between layers but rather handed over from layer to layer without change. This, in turn, would challenge the various technologies used for implementing these layers to become totally SWT aware.

Another handy tool would be a semantic programming language with language primitives that enable the direct processing of semantic data, thus avoiding representation in classical programming language data types. Finally, efforts like those in Finland[4] can help ensure that the data interpretation problem is addressed in a serious way, across industries and governments. The ultimate question is whether the SWT community can step up to this challenge.

Acknowledgments

This article benefited from discussions with Charles Petrie.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.*, May 2001; www.sciam.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.
2. O. Lassila and J. Hendler, "Embracing 'Web 3.0,'" *IEEE Internet Computing*, vol. 11, no. 3, 2007, pp. 90–93; www.mindswap.org/papers/2007/90-93.pdf.
3. C. Bussler, *B2B Integration*, Springer, 2003; www.springer.com/dal/home/computer/database+management+&+information+retrieval?SGWID=1-153-22-2236421-0.
4. E. Hyvönen, "Semantic Web Applications in the Public Sector in Finland – Building the Basis for a National Semantic Web Infrastructure," white paper, presented at the Norwegian Semantic Days, 2006; www.seco.hut.fi/publications/2006/hyvonen-FinnONTO-2006-04-14.pdf.
5. N. Shadbolt, W. Hall, and T. Berners-Lee, "The Semantic Web Revisited," *IEEE Intelligent Systems*, vol. 21, no. 3, 2006, pp. 96–101; http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1637364.

Christoph Bussler is author of several books and journal articles on integration and semantics. His research interests include workflow and process management, B2B and EAI integration, and semantic computing. Bussler has a PhD in computer science from the University of Erlangen, Germany. Contact him at chbussler@aol.com.