United Business Media

Dr. Dobb's DIGEST

The Art and Business of Software Development November, 2009

Beware Open Source Encryption

Is there a time bomb ticking in your open source software?

By Jonathan Erickson,
Editor In Chief

There's nothing like an official letter from the U.S. Department of Commerce's munitions control office to make you choke on your morning coffee. At least that was my response upon receiving such a notice a few years ago. It turns out we were (and still are) exporting "Dr. Dobb's Essential Books of Cryptography and Security" CD-ROM which provides the full text of articles and books — as well as source code — about encryption algorithms and protocols.

We'd been exporting "munitions" without a license — that is, munitions that consisted of source code implementations of cryptography algorithms.

The solution was simple: I filled out some paperwork, wrote a $250 check, and Dr. Dobb's became a legally sanctioned munitions exporter.

Since then, the Department of Commerce, which administers munitions exports, has made some changes, forming the Bureau of Industry and Security to deal with the likes of Dr. Dobb's and acknowledging the concept of both open source software and the Internet. Now, organizations and individuals are required to notify the Bureau before making open source cryptographic software public on the Internet.

Still, open source cryptographic software can be a time bomb for companies using open source. A recent search of the Black Duck KnowledgeBase revealed more than 4,000 projects that include encryption algorithms strong enough to require a license if the code is exported, while another 3,900 projects might require an export license.

Which crypto algorithms are popping up most often in open source? The usual suspects lead the list — RSA, DES, MD5, SHA, Blowfish, Diffie-Hellman, ElGamal, and AES. As far as the government is concerned, if your company exports software that includes implementation in source code of even a single strong encryption algorithm, then you must get a license, no matter who wrote the software or when it was written. Violators of encryption export controls can be subject to fines and imprisonment. Open source is here to stay and increasingly central to the IT landscape, and for good reason. However, that doesn't mean caution isn't in order, especially when security is involved.

Return to Table of Contents

P vs. NP

The most notorious problem in theoretical computer science remains open, but the attempts to solve it have led to profound insights

In the 1995 Halloween episode of "The Simpsons," Homer Simpson finds a portal to the mysterious Third Dimension behind a bookcase, and desperate to escape his in-laws, he plunges through. He finds himself wandering across a dark surface etched with green gridlines and strewn with geometric shapes, above which hover strange equations. One of these is the deceptively simple assertion that P = NP.

In fact, in a 2002 poll, 61 mathematicians and computer scientists said that they thought P probably didn't equal NP, to only nine who thought it did — and of those nine, several told the pollster that they took the position just to be contrary. But so far, no one's been able to decisively answer the question one way or the other. Frequently called the most important outstanding question in theoretical computer science, the equivalency of P and NP is one of the seven problems that the Clay Mathematics Institute will give you a million dollars for proving — or disproving. Roughly speaking, P is a set of relatively easy problems, and NP is a set of what seem to be very, very hard problems, so P = NP would imply that the apparently hard problems actually have relatively easy solutions. But the details are more complicated.

Computer science is largely concerned with a single question: How long does it take to execute a given algorithm? But computer scientists don't give the answer in minutes or milliseconds; they give it relative to the number of elements the algorithm has to manipulate.

Imagine, for instance, that you have an unsorted list of numbers, and you want to write an algorithm to find the largest one. The algorithm has to look at all the numbers in the list: there's no way around that. But if it simply keeps a record of the largest number it's seen so far, it has to look at each entry only once. The algorithm's execution time is thus directly proportional to the number of elements it's handling — which computer scientists designate N. Of course, most algorithms are more complicated, and thus less efficient, than the one for finding the largest number in a list; but many common algorithms have execution times proportional to N2, or N times the logarithm of N, or the like.

A mathematical expression that involves N's and N2s and N's raised to other powers is called a polynomial, and that's what the "P" in "P = NP" stands for. P is the set of problems whose solution times are proportional to polynomials involving N's.

Obviously, an algorithm whose execution time is proportional to N3 is slower than one whose execution time is proportional to N. But such differences dwindle to insignificance compared to another distinction, between polynomial expressions — where N is the number being raised to a power — and expressions where a number is raised to the Nth power, like, say, 2N.

If an algorithm whose execution time is proportional to N takes a second to perform a computation involving 100 elements, an algorithm whose execution time is proportional to N3 takes almost three hours. But an algorithm whose execution time is proportional to 2N takes 300 quintillion years. And that discrepancy gets much, much worse the larger N grows.

NP (which stands for "nondeterministic polynomial time") is the set of problems whose solutions can be verified in polynomial time. But as far as anyone can tell, many of those problems take exponential time to solve. Perhaps the most famous problem in NP, for example, is finding prime factors of a large number. Verifying a solution just requires multiplication, but solving the problem seems to require systematically trying out lots of candidates.

Dr.Dobb's

EDITOR-IN-CHIEF
Jonathan Erickson

EDITORIAL
MANAGING EDITOR
Deirdre Blake
COPY EDITOR
Amy Stephens
CONTRIBUTING EDITORS
Mike Riley, Herb Sutter
WEBMASTER
Sean Coady

VICE PRESIDENT, GROUP PUBLISHER
Brandon Friesen
VICE PRESIDENT GROUP SALES
Martha Schwartz

AUDIENCE DEVELOPMENT
CIRCULATION DIRECTOR
Karen McAleer
MANAGER
John Slesinski

DR. DOBB'S
600 Harrison Street, 6th Floor, San Francisco, CA, 94107. 415-947-6000.
www.ddj.com

UBM LLC

Pat Nohilly Senior Vice President, Strategic Development and Business Administration
Marie Myers Senior Vice President, Manufacturing

TechWeb

Tony L. Uphoff Chief Executive Officer
John Dennehy, CFO
David Michael, CIO
John Siefert, Senior Vice President and Publisher, InformationWeek Business Technology Network
Bob Evans Senior Vice President and Content Director, InformationWeek Global CIO
Joseph Braue Senior Vice President, Light Reading Communications Network
Scott Vaughan Vice President, Marketing Services
John Ecke Vice President, Financial Technology Network
Beth Rivera Vice President, Human Resources
Fritz Nelson Executive Producer, TechWeb TV

techweb™

So the question "Does P equal NP?" means "If the solution to a problem can be verified in polynomial time, can it be found in polynomial time?" Part of the question's allure is that the vast majority of NP problems whose solutions seem to require exponential time are what's called NP-complete, meaning that a polynomial-time solution to one can be adapted to solve all the others. And in real life, NP-complete problems are fairly common, especially in large scheduling tasks. The most famous NP-complete problem, for instance, is the so-called traveling-salesman problem: given N cities and the distances between them, can you find a route that hits all of them but is shorter than whatever limit you choose to set?

Given that P probably doesn't equal NP, however — that efficient solutions to NP problems will probably never be found — what's all the fuss about? Michael Sipser, the head of the MIT Department of Mathematics and a member of the Computer Science and Artificial Intelligence Lab's Theory of Computation Group (TOC), says that the P-versus-NP problem is important for deepening our understanding of computational complexity.

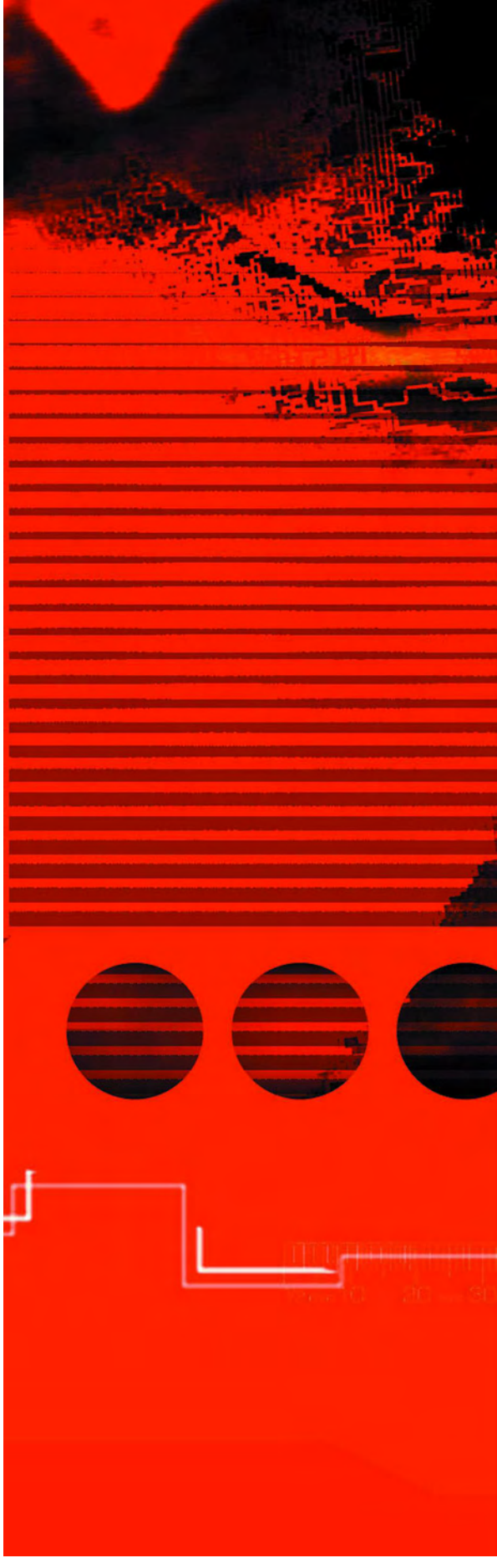
"A major application is in the cryptography area," Sipser says, where the security of cryptographic codes is often ensured by the complexity of a computational task. The RSA cryptographic scheme, which is commonly used for secure Internet transactions — and was invented at MIT — "is really an outgrowth of the study of the complexity of doing certain number-theoretic computations," Sipser says.

Similarly, Sipser says, "the excitement around quantum computation really boiled over when Peter Shor" — another TOC member — "discovered a method for factoring numbers on a quantum computer. Peter's breakthrough inspired an enormous amount of research both in the computer science community and in the physics community." Indeed, for a while, Shor's discovery sparked the hope that quantum computers, which exploit the counterintuitive properties of extremely small particles of matter, could solve NP-complete problems in polynomial time. But that now seems unlikely: the factoring problem is actually one of the few hard NP problems that is not known to be NP-complete.

Sipser also says that "the P-versus-NP problem has become broadly recognized in the mathematical community as a mathematical question that is fundamental and important and beautiful. I think it has helped bridge the mathematics and computer science communities."

But if, as Sipser says, "complexity adds a new wrinkle on old problems" in mathematics, it's changed the questions that computer science asks. "When you're faced with a new computational problem," Sipser says, "what the theory of NP-completeness offers you is, instead of spending all of your time looking for a fast algorithm, you can spend half your time looking for a fast algorithm and the other half of your time looking for a proof of NP-completeness."

Sipser points out that some algorithms for NP-complete problems exhibit exponential complexity only in the worst-case scenario and that, in the average case, they can be more efficient than polynomial-time algorithms. But even there, NP-completeness "tells you something very specific," Sipser says. "It tells you that if you're going to look for an algorithm that's going to work in every case and give you the best solution, you're doomed: don't even try. That's useful information."



Return to Table of Contents

4 Steps To Better SOA

Ontology gives semantic integration the boost it needs to speed data sharing

by Raghuraman Krishnamurthy, Vinod Ranganathan, and Baskar Senguttuvan

Looking back on NASA's effort to integrate data from many different sources, Richard Keller shakes his head.

The space agency wanted to take data collected during a field expedition, including sample data, photos, voice recordings, and GPS information and integrate it with satellite imagery, GIS data, and information about the characteristics of minerals found in the samples. "Not only were the types of information heterogeneous, but the formats ranged widely from spreadsheet files to SQL databases to Web pages," the senior research computer scientist says.

But that was only half the problem. Once he got the information, Keller then needed to determine how data stored in all those different formats was related.

"If you have fields in two separate databases that are both named 'temp,'is it legitimate to assume they represent the same quantity and can be integrated together?" Keller asks.

"One might represent a 'temperature' and the other a 'temporary' value. To properly combine these two fields, you really need to understand what the data represents." (Read more from Keller at www.ddj.com/architect/220700133.)

At NASA and other large entities — both in government and business — integrating heterogeneous data is a challenge, but it's one that must be faced in order to easily share information internally and with outside partners. The integration challenge is one reason why NASA and many organizations are turning to service-oriented architectures combined with semantic integration. SOA consists of services that offer interoperability capabilities built into the network. While SOA's business-centric approach has sparked enthusiasm, the challenge is now to build in inference capabilities to make intelligent and dynamic selection of Web services possible. This is where semantic technology — the modeling of an area of knowledge like biology or economics as close to natural language as possible — comes into play.

While SOA's business-centric approach has sparked enthusiasm, the challenge is now to build in inference capabilities to make intelligent and dynamic selection of Web services possible. This is where semantic technology — the modeling of an area of knowledge like biology or economics as close to natural language as possible — comes into play.

Semantic Technology

A SOA platform with built-in inference capabilities can let companies make runtime decisions based on service capabilities and invocation context where tasks are executed based on predefined conditions. On its own, SOA doesn't have this capability, but with the addition of inference, SOA gets the semantic semantic richness it needs to make runtime decisions Semantic integration techniques build context and awareness of the underlying services and data in a metalayer. However, many of these techniques aren't fully mature. Tools like Progress Software's DataXtend Semantic Integrator tackle this problem with a common data model that validates data exchanges.

Semantic integration techniques build context and awareness of the underlying services and data in a metalayer. However, many of these techniques aren't fully mature. Tools like Progress Software's DataXtend Semantic Integrator tackle this problem with a common data model that validates data exchanges.

Ontology-Enabled SOA

Semantic integration techniques are based on the idea that it's possible to find patterns in the origin of various pieces of heterogeneous data. Once done, then it's possible to define these patterns using a metamodel, and then connect several metamodels by defining how they relate to one another. One emerging approach to semantic integration is ontology. Ontology is a structured representation of a domain — by which we mean an area of knowledge like biology or economics — in

terms of classes and interrelationships between the classes that can be defined using the XML-based Web Ontology Language.

Any domain is usually expressed by splitting it into classes, and then expressing relationships among those classes. Ontology-enabled SOA extends this modeling technique by building a layer on top of the services dictionary that contains the ontology mappings of the services domain. The mapping is done during the design phase of building an ontology, then implemented at runtime to find semantic similarity for the service request. Policies are then built that contain logic for finding semantic similarity.

There are four steps in building an ontology-enabled SOA:

Step 1: Analyze The Business Process. A business process typically involves execution of a series of tasks based on some conditions. These conditions may represent intelligent routing decisions requiring semantic power. We define these as "points of variability," or areas where inference capability is required for semantic richness.

There are several variables for each task; and for each variable, there are multiple possible values. Future business needs may mandate additional variables or new values for existing variables (e.g., Physical State = "Gaseous"). Here is where defining an ontology model for these variables enables a vocabulary that can be expanded over time and a nondisruptive way to meet new business needs.

Step 2: Construct The Ontology Model. Ontology is defined in terms concepts and relationships. Ontology concepts are implemented in classes.

Relationships are implemented in terms of "object properties" and "data type properties." A mapping of components of points of variability to elements of ontology can assist in constructing the ontology model:

- Task List: Classes
- Variables List: Class Relationships
- Variables Value: Class Attributes

A sound knowledge of the business is a prerequisite to accurately identifying tasks and points of variability in the business process and ultimately mapping them to ontology elements to create a reusable model. Tools like the open source ontology editor and knowledge-base framework Protege can be used to create the ontology model.

Step 3: Create Context-Sensitive Invoker. Creating the runtime component that consults the ontology and makes dynamic context-sensitive service invocations is the next step. One way to implement this is by using Java-based technologies.

The business process flow can be encapsulated as a linear process in a Business Process Execution Language, or BPEL, block. The runtime component uses ontology mapping to determine the appropriate service end point.

This semantic richness is enabled by defining "policies" and "assertions." Policies operate on the available and published end points to a service and define who's eligible to use the service. Assertions are the ontology mappings of all possible context-sensitive variables. The runtime determination of the end point to invoke is based on the best match between the set of assertions and policies.

Step 4: Web Services. Web services provide the actual business service. They're the "consuming" points, invoked to achieve business functionality. The policy that operates on the Web services is part of the task required to incorporate a new service end point.

Does Ontology Make Sense for Your Business?

- If your business processes have a lot of variability, defining an ontology model could help streamline processes by delegating complexity to a layer that eases semantic integration.
- If your business is faced with a lot of change, then its business process implementation may get quickly outmoded. An ontology layer can provide a good reference model for the current implementation and seamless extensibility to encompass future variability.
- In some domains, such as biology and economics, shared ontologies are being created that could evolve into industry standards. An ontology-based approach lets you adhere to evolving standards.
- If you use public data sources, ontology is often the natural choice, especially in areas where knowledge representation is important. the network.

Putting It All Together

Semantic integration is being applied across a variety of industries, including financial services and pharmaceuticals, and as ontology-based semantic integration matures, it's likely to be applied broadly, too. A SOA architecture we built uses ontology to enable a typical business domain — drug discovery. All drug discovery activities need chemical and biological samples, and sample management deals with the way the samples are acquired, registered, stored, and distributed. Our ontology addresses the sample acquisition activity, dynamically invoking appropriate Web services.

The system works like this: A researcher requests samples (Step 1, above). The request triggers a BPEL process (Step 2) in which the business process modeler's BPEL engine passes the request on to a context-sensitive invoker (Step 3). The invoker then interacts with the ontology engine and retrieves the assertions based on the context. The result is the context-sensitive selection of Web services using ontological means (Step 4).

Ontology can be used to enable semantic integration by modeling the business domain process as a workflow, and constructing an ontology model alongside the business process modeling. The classes and relationships of the ontology model are created based on the domain knowledge and the business process.

On the downside, an ontology-based SOA can be complex and difficult to maintain if not implemented correctly. But when it's done right, using ontology to create intelligent software agents is an excellent way to create user-friendly software that helps in decision making and speeds information sharing.

— Raghuraman Krishnamurthy is a principal architect, Vinod Ranganathan a technical lead, and Baskar Senguttuvan a senior architect at Cognizant's Technology Consulting Group.

Return to Table of Contents

Strengthening Service-Oriented Architectures (SOA) with Semantics

by John Hebeler and Andrew Perez-Lopez

Service-Oriented Architecture (SOA) is the de-facto architectural standard for many small and large distributed solutions. Its growing relevance is reflected not only in the bevy of rich services the SOA stack offers, but also the variety of available implementations, many of which are open-source. Yet SOA remains challenged in several areas dealing with scale, growing complexity, diverse knowledge management, and the inherent dynamics of large, distributed systems. These challenges unnecessarily constrain SOA solutions causing them to fall short of their ultimate potential.

Semantics can help. Semantics form expressive, useful computing abstractions or concepts that address these challenges and strengthen the SOA mission to use and leverage complex information and services across an entire enterprise or the globe.

In this article, we examine adding semantics to SOAs through four examples, each contributing in different ways yet adhering to the SOA standards and goals. Two examples illustrate "SOA-ize" semantic services and the other two "Semantic-ize" the SOA. All contribute to improving the scope and capabilities of what is possible with a SOA solution.

SOA Struggles

SOA exists as a set of standards and services that form a dynamic distributed service fabric that provides a step forward into distributed computing possibilities. The interface standards allow services to interact despite fundamental platform differences. SOA and services is analogous to the World Wide Web (WWW) and documents — the WWW exists as a set of standards that allow document contributions based on many technologies and techniques all following the same document standards. SOA attempts to do the same with services. Whereas WWW standards unleashed a world of information, SOA attempts to unleash a world of services. However, services can be much more complex and temperamental than documents (and documents themselves have gotten pretty complicated). The SOA stack addresses some of this complexity by integrating a set of management services. The vastness and richness of the network services can easily overwhelm a SOA solution even with a powerful stack of services.

Theoretically, a SOA implementation can exist at a variety of levels from a small solution integrating a handful of services up to a large, dynamic solution integrating thousands of services. In practice, SOA strongly leans to the former — integrating a few, well-controlled services. Yet the real power and possibilities require a much bigger leap into the thousands. That leap incurs a multitude of challenges including service complexity, information complexity, and service management complexity.

Services are the fundamental building blocks of a SOA solution. Service complexity refers to the degree of sophistication associated with the tasks the service actually accomplishes. Whereas a document is merely a bunch of organized words, services do things — potentially very complex things. Additionally, they are dynamic. Services are often bound to the objects that drive their behavior. But these objects themselves can limit the ultimate complexity of the overall application. Objects or instances are bound to their creating class and often that class is static. Attributes and relationships are typically also bound to the creating class. Too often in object oriented software for services, there is little analysis of the object structure as the underlying data model. Is it consistent? Is it logical? Is it cohesive? These answers are left solely to the programmer, and the hard-coded software. Complex behavior frequently yields complex code — often unreadable code. Some tools exist to help externalize this inherent complexity but most do not. Programming languages can also limit the complexity and thus the power of individual services. SOA solutions integrate multiple services, which, of course, compound this phenomenon.

Information processing is often at the heart of

a SOA solution or almost any computer solution — information is the "crown jewels" of computing. Just as the complexity of services stands as a challenge to SOA system designers, so too does information complexity. Most data technologies incorporate limitations that force the information to reside in multiple places and to be reassembled to produce the desired results. A typical way of addressing this data distribution issue is with XML. Databases, combined with business logic, produce data elements that can form the desired answer in XML. XML is a step in the right direction, but it struggles with complex relationships and dynamic constraints. The actual meaning of the data, the information, is separated in the document and in the XML schema, or even worse, the schema of the database back-end, and therefore very difficult to manage. These limitations of the information format and knowledge representation force all of the services in the enterprise, which should be focused on business logic, to handle a higher degree of complexity just to get started. This, of course, further aggravates the service complexity challenge outlined above.

SOA solutions enable the dynamic integration and collaboration of multiple services. This requires service management — and this can be very challenging. In a real sense, the integration of services forms new, more complex services. We examine three aspects of service management:

- Service abstractions
- Service registration
- Service dynamics

Each represents a major aspect of service management.

Service abstractions describe the services. The richer and more complete the abstractions, the more easily the service is found and properly used. Most SOA services exist as low-level or simple abstractions. This low level forces the programmer to take up the slack and really understand the service beyond the simple abstraction descriptions. For example, say a software developer determines the need for a service, like a catalog lookup. Given a high-level abstraction like "catalog lookup" the developer would, in an ideal SOA world, find a service and be able to integrate it easily into an application at that high level. However, as we in the real world know, to take advantage of the catalog lookup service, a SOA service too often requires inspecting complex APIs defined in Web Service Description Language (WSDL), which identifies particular end point names, the numbers and types of variables, and return values for each supported method. Even those detailed WSDL descriptions, however, do not describe how the parameters serve the method, the purpose of the method, or any expectations with regard to errors and performance. This forces you, the programmer, to invert your development process in an unnatural way. Instead of finding a service that matches the needs of your application, you must design your application to match the requirements of the services. In other words, the services don't really serve your specific need, they serve their own needs in their own particular way. You must call and use the service according to the service's usage patterns and not your own. Of course, all of this is moot if you design all the services, or if all of the assumptions are shared by every service, etc — but that is extraordinarily limiting in terms of the services that your application can exploit, and is really contrary to the goal of large-scale SOA. The key is inverting this perspective and enabling a service to align with your system's needs. Thus, your SOA system accesses the services or aggregation of services at the level of abstraction that makes sense to your needs. This is somewhat analogous to views for a relational database. This service view must also accommodate similar services, failed services, and so on. Ideally, you just want your overall system to operate correctly and not have to change drastically to accommodate the specific syntax or other dependencies of a particular service implementation.

Service registration remains a real challenge for SOA. How can you compose the best SOA without full knowledge of what services are available and how to interact with them? The Universal Description, Discovery, and Integration (UDDI) standard has fallen short of realizing its larger goals. The inherent complexity of using XML to represent such rich knowledge has produced an unwieldy solution that befuddles many developers. Many SOA solutions now register their services in hand-carved proprietary solutions or use no UDDI implementation at all, and some even try to use a Wiki. This is a direct blow to the scale of SOA efforts. A quick look at programmableweb.com illustrates this problem. This site lists thousands of web services, yet only a handful are used to support almost all the mashups and the vast majority of mashups use only two or fewer services (with a 40% chance that one of these is Google Maps). Sadly, most offered services are not used in any mashups. This situation could be the result of inadequate services but it is more likely due to the inability of developers to find and utilize the services they want.

For SOA to work properly, a service registration system must clearly describe and search a vast array of different service offerings available in the given SOA. These requirements demand a solution that is more than just a common set of syntaxes. Explicit syntax agreements are important, for without them nothing would work. However, determining a common syntax (or set of syntaxes) only allows exact matches or at best a basic category search. This works fine for services that you control (and name) but does not stand up to the challenge of integrating services that originate and evolve from many sources. Actually, for services you create, the registry is not even that important — you already know the descriptions. However, large-scale integration needs much more than a common syntax — it needs a common semantics. Capturing the information about what a service is and what it does and communicating that information effectively in an automated way remains a key challenge for large, dynamic SOA implementations.

Service abstractions and registration contain another inherent challenge — dynamics. Over time, services change — that is a simple rule, and they usually change for the better. Traditional SOA techniques often create brittle connections to services that may break on the slightest change to the service. Again, this is no trouble if you develop and control the services, but again, that is con-

trary to SOA's goal. This inhibits the use of services to two types: those that you control or those that are so big they will never (or at least very rarely) change. This set describes only a small fraction of the available services, and requires that SOA applications have regular, and often significant, maintenance. Without better ways to deal with the dynamics of web services, the growth of an SOA solution will remain stunted and more costly.

These SOA struggles result in a reduced awareness of what is available, a limited capacity to compose services, and a diminished exploitation of valuable services due to uncontrolled changes. In an ideal world, SOA could seamlessly link services around the globe, but these factors significantly threaten that vision.

Semantics at Your Service

Semantic technologies capture meaning and employ it to ease communication between services. This meaning can be used to give richer descriptions at all levels of the SOA, from the underlying data to individual service descriptions to aggregate service descriptions. Semantic technologies can strengthen the SOA mission by absorbing and leveraging higher levels of complexity at every level. For example, semantic technology can organize data and processing to better reflect a person or a company — the meaning forms complex, dynamic concepts beyond other methods.

In the case of computers, the use of the term semantics can easily be confused with human understanding, but make no mistake: computers using semantic technologies do not understand anything. Computers simply process information — but if the information is represented in such a way that it is imbued with more of it's meaning, then computers can act upon that information. For example, a series of statements and data structures in a programming language may capture the approval process of a loan application. In a sense, the programming code captures the meaning behind the approval and denial of a loan. A programmer carved the business meaning directly into code. However, the computer system executing that code has no conception of the higher level task it is accomplishing — it has no way of representing the idea of a loan application. Likewise a database programmer can design database tables and columns to achieve similar results, thereby encoding the business logic into the data representation. The meaning is in there — but not well organized or abstracted. The fact remains that it might be theoretically possible to extract the semantics of loan approval from the code and associated databases, but it is not cohesive or easy to identify. You can't easily find, organize, and distinguish the meaning parts from other coding parts, the meaning get tightly coupled with the program logic and database schemata. Where do you go if the loan procedure changes? Is the meaning consistent? When an error occurs, where is the culprit? These questions quickly haunt complex SOA solutions, and result in systems that that tend to scatter meaning and lose their ability to adapt.

Luckily, other technological forms of semantic representation exist that offer a range of expressivity or meaning capture that better serves these challenges. Now you might think the right choice of semantic technology for your applications would be the most expressive and sophisticated one available, such as first order logic or even richer forms. However, oftentimes the more complex a representation is, the more difficult it is to use. The harder a technology is to use, the less widely it is adopted. And less adoption means failure when SOA solutions attempt to span the globe. Large scale SOA is a critical mass solution. What is needed is a flexible approach that balances complex expressivity with ease-of-use participation. Albert Einstein famously said that "Everything should be made as simple as possible, but not simpler." A prominent Semantic Web researcher, Dr. Jim Hendler, was quoted as saying something that could be thought of as a corollary: "A little semantics goes a long way" (see http://www.cs.rpi.edu/~hendler/LittleSemanticsWeb.html). This level of information representation that is richer than traditional data models but still designed for Web-scale performance is a perfect fit for Semantic Web technologies. The Semantic Web represents meaning as a directed graph of binary relationships between resources. That is, one resource is related to another resource through a directed link called a predicate. A resource can represent anything, and need not be a web page or service. With respect to services, Semantic Web relationships can be thought of as falling into three relationship categories: instance relationships, structure relationships, and constraint relationships.

1. Instance relationships such as 'John is friends with Mary'. These simply describe an association between two resources, also called instances. In this case, John and Mary are linked by the predicate is friends with. Without any additional information about the significance of the is friends with predicate, there is no implied structure or meaning beyond merely associating two resources with each other.
2. Structural relationships such as 'John is a Person' and a 'Person is a Living Thing'. These provide structure to the instance data by assigning the resource John to be a member of the Person class and that the Person class is a subclass of the Living Thing class. Note that the relationship between Person and Living Thing is quite apart from the John instance. This allows quite a bit of flexibility and multiple perspectives on the same instance relationships.
3. Constraint relationships such as 'Persons can only have friends with other Persons'. Essentially these relationships establish rules that better define the various classes. This allows a more fine grain capture of how to define a particular class such as Person.

The last two types of relationships enable tools called reasoners, to infer additional relationships or entailments regarding the provided statements. For example, a reasoner could infer that John is a Living Thing from statements in 2 above and that Mary is a Person from the statements in 1 and 3 above. This is very powerful because we see that the data itself generates additional data — no human is required.

Several other traits of the Semantic Web are also important to keep in mind. First and foremost is that it is an accepted, open, W3C standard. This protects your investment in a semantic data

model while also encouraging wide participation from a community of researchers and professionals to develop tools, methods, and even additional semantically-enabled data sources. Secondly, the Semantic Web is run-time dynamic. Semantic Web solutions can change structure and constraints at any time. This flexibility allows solutions to maintain constant alignment with the changing goals of a SOA solution or service. Lastly, the structure of the Semantic Web is completely independent from the instance data. Thus, structure can evolve independently from the instance data and multiple structures can simultaneously exercise the same instance data. The structure and constraints semantics emanate from the same language constructs as the instance data. There is no difference between information model underlying the three relationship categories. In contrast, relational databases have significant language differences between the structure or database schema and the instances or rows. This feature of Semantic Web systems allows a solution to explore the Semantic Web through structure, constraint, and instance patterns all expressed in the same language.

A collection of these semantic statements containing instances, structure, and constraints offer the ability to create and manage rich, complex semantics. We can apply these semantics directly to the challenges of SOA. Simply put the Semantic Web cohesively forms meaning above and beyond the syntax offered by SOA and helps form a cohesive approach to meaning throughout SOA. So, how does this help?

- **Knowledge-based SOA Services:** Services that directly participate in a SOA can use semantic technologies to take advantage of the inference and enhanced information representation to ferret out patterns and relationships more efficiently than with other data techniques. These results can serve the SOA in two ways — indirectly by simply providing an answer while keeping the semantics embedded or directly providing the actual relationship graph or meaning for other knowledge-based services to consume. This extends the data capabilities of the SOA by direct incorporation of knowledge services. Simply put, semantic services provided by the Semantic Web are perfectly compatible with SOA.
- **Flexible and Adaptive SOA Service Composition:** SOA services can contribute to a larger solution in both a specific and an abstract form. The specifics are the various methods and data that form an exchange between services. The abstraction is the high-level business utility of the service — the concept above all the technology. The Semantic Web can unite both forms and keep them in step. Thus, a service can select the technical specifics to maintain a conceptual, dynamic business priority. This also allows abstract service composition that maintains the specifics needed to get the job done without being bound to those details. Semantic service composition can achieve higher-level business sequencing by chaining together services that follow the business meaning and thus allow run time changes. It also offers promise for automatic composition where services seek out additional required services. Additionally, service composition could improve performance and availability by chaining together similar services and making resource allocations based on the current business priorities. The capturing of the composition in one standard form allows easy changes and easy recognition of any faults.
- **Service and Data Integration:** Since the semantic concepts can represent many various formats and forms of technology, it serves to more gracefully integrate services and data. Integration requires varying degrees of equivalence (a type of relationship) from making two different XML equivalent to making complex procedures equivalent. Additionally, integration is often thought of as unifying two technologies but it is also unifies people to technologies. Semantics can build the customized perspectives necessary for effective, customized human interactions.
- **Improved Data Exploitation:** Semantic reasoning can infer missing pieces, expand conceptual perspectives, and identify critical flaws. This amplifies and extends the information to produce more useful results. Reasoning can identify commonality such as the Living Thing example above. Reasoning can detect simple and complex data inconsistencies and conflicts. This can help maintain high integrity across the SOA. Additionally, the nature of semantics allows a decoupling of structural information from instance information. This permits multiple interpretations of data and variable levels of integrity and quality. Although a SOA could maintain an absolute truth it doesn't need to — this is especially critical when dealing with large systems that might never agree on anything that is absolute.

We now help illustrate those semantic enhancements in four examples. Two examples illustrate SOA-izing your Semantics by adding semantic services to a SOA solution and two illustrate Semantic-izing your SOA through adding semantic capability to fundamental SOA design constructs.

SOA-izing Your Semantics

SOA-izing your Semantic Web applications makes them available to the wide, powerful ecosystem established through SOAs. These semantic applications, exposed as standard services allow organizations to bring the benefits of semantics into their enterprise-wide SOA solutions using familiar and compatible SOA interfaces. Consumers of the service need not even be aware that semantic technologies are employed in the implementation of the service. The first example simply embeds the Semantic Web into its service and then communicates with SOA through traditional SOA interfaces. This is certainly not earth-shattering but it is important to point out how this simple integration offers semantic benefits to the SOA world. The second example forms a SPARQL endpoint, an emerging standard, into a standard SOA web service. This opens up a semantic knowledgebase directly to the SOA allowing deep interrogation of the underlying knowledgebase from other semantic services.

The two together help illustrate the value in contributing the power of knowledgebases, reasoners, and dynamic, semantic representations to a SOA. This enables the SOA solution to include semantics at the service level to best accomplish its mission. These examples illustrate techniques to address the challenges of information complexity and service dynamics. The underlying semantics enable rich information expressions and interrogation to meet the challenge of information complexity. The information dynamics inherent to the Semantic Web allow run-time adjustments to

the semantic data, its semantic structure and constraints.

Embedded Semantic Web SOA Service

Embedding semantics into a SOA service is straight forward. The service establishes its traditional SOA interface and registration. This exposes, to varying degrees, the value in forming, manipulating, and querying a complex knowledgebase but by maintaining a service layer, the subscribing client need not know anything about the Semantic Web. This provides the benefits of Semantics without adding any complexity to other SOA services.

This example shows two key parts of the embedded semantic service; the service declaration and service implementation that contains the Semantic Web constructs. The latter illustrates the knowledgebase composition. This maintains a complete decoupling between the Semantic Web and the other SOA services.

Our example provides a semantic service that accesses the catalogs of several suppliers. The Semantic Web technologies are ideally suited for such integration. The Semantic Web technologies describe the contents of each vendor's catalog for easy integration. The translation from various data forms to the Semantic Web RDF is often easily done using XSL transformation or some equivalent. RDF, the de-facto language of the Semantic Web, expresses relationships among concepts and allows for the transparent integration of multiple catalogs.

Figure 1 contains two basic catalogs from two suppliers. The first supplier's vocabulary describes Products, of which some are Books. Products names and prices are described using the hasName and hasPrice predicates, respectively. The second supplier describes Items, of which some are Journals and some are Magazines. Their names and prices are similarly described, but with different predicates: title and pricePerIssue. Finally, Figure 1 shows the simple query service and associated business object to represent a catalog item with a given price.

Using just a few lines of the Web Ontology Language (OWL), you can describe the relationships between the concepts from the first supplier's information model and those from the second supplier's.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix one: <http://example.com/supplierOne#> .
@prefix two: <http://example.com/supplierTwo#> .
one:Product owl:equivalentClass two:Item .
one:hasPrice owl:equivalentProperty two:pricePerIssue .
one:hasName owl:equivalentProperty two:title.
```

The two different suppliers use different vocabularies to encode their catalogs, and the fact that supplier two has the notion of a pricePerIssue suggests that there might be some significant differences between their wares. However, for the purposes of a price check into the catalogs, those differences can be ignored, and the classes and properties above can be thought of as equivalent. Then, using just a few lines of Java code, you can combine the three models — the data from the two suppliers, and those OWL statements that serve to align the data models:

```
supplierOneModel =
    loadModel("resources/supplierOne.tur", "TURTLE");
 supplierTwoModel =
    loadModel("resources/supplierTwo.tur", "TURTLE");
 jointModel =
    loadModel("resources/alignment.tur", "TURTLE");

jointModel.add(supplierOneModel);
jointModel.add(supplierTwoModel);
```

After the models have been added to the joint model, a query can be written that exclusively uses the vocabulary of the first supplier which will still return results from the second supplier's catalog. This is because the equivalences defined in the OWL above allow a reasoner to infer that all Items from the second supplier's catalog are equally validly referred to as Products, like those from the first supplier's. When executed against the model, the query returns The Cat in the Hat and Where the Sidewalk Ends, but also Dr. Dobb's, because all three cost less than $12.

```
PREFIX one: <http://example.com/supplierOne#>
SELECT DISTINCT ?name ?price
WHERE {
[] a one:Product ;
     one:hasName ?name ;
     one:hasPrice ?price
FILTER(?price < 12.00) .
}
```

SPARQL Endpoint SOA Service

Rather than use semantic technologies behind the scenes as with the first example, a service could alternatively choose to expose them to other semantic-enabled services. This allows SOA services to work together to form a solution while taking advantage of the semantics.

The Semantic Web community has been interested in the complementary role that semantic technologies can play in SOA systems for some time. The World Wide Web Consortium (W3C)-recommended query language for RDF is called SPARQL, and the community is working towards standard artifacts for easy communication with SPARQL endpoints.

The next example illustrates how easy it is to expose semantic data as a SPARQL endpoint. We modify the previous example to accept arbitrary SPARQL SELECT queries and return the results according to the XML ResultSet schema that the SPARQL protocol uses. Assuming the same scenario as in the first example, exposing the combined catalogs as a SPARQL endpoint can give more visibility into the data by clients that are aware of semantic technolo-

Figure 1: Embedded semantic example.

gies. Here's an excerpt from the implementation the service class:

```
@WebService
public class SPARQL {
   private final Object _lock = new Object();
   private final ResultSetFormatter _resultSetFormatter =
     new ResultSetFormatter();
   private boolean _initialized = false;
   private Model _model = null;
   private final String SUPPLIER_ONE_FILE =
     "resources/supplierOne.tur";
private final String SUPPLIER_TWO_FILE =
           "resources/supplierTwo.tur";
   private final String SUPPLIER_ALIGNMENT_FILE =
             "resources/alignment.tur";
   private Model loadModel(String fileName, String format) {
   // model loading code, removed only because it is
    // very similar to the code shown from the first example...
   }
   private void loadModel() {
     _model = loadModel(SUPPLIER_ALIGNMENT_FILE, "TURTLE");
     _model.add(loadModel(SUPPLIER_ONE_FILE, "TURTLE"));
     _model.add(loadModel(SUPPLIER_TWO_FILE, "TURTLE"));
   }
   private void initialize() {
     if(!_initialized) {
       synchronized(_lock) {
         if(!_initialized) {
           loadModel();
           _initialized = true;
         }
       }
     }
   }
   public String query(String query) {
     initialize();
     String toReturn;
     QueryExecution qe = QueryExecutionFactory.create(
              query, _model);
     ResultSet results = qe.execSelect();
     toReturn = _resultSetFormatter.asXMLString(results);
     return toReturn;
   }
}
```

The SPARQL protocol specification defines an XML result set format which is returned by this code. This example depends on the Jena Semantic Web Framework (http://jena.sourceforge.net/), an open source implementation of many of the Semantic Web technologies that has come to be very popular in the Semantic Web community. One of Jena's components is ARQ, an implementation of the SPARQL query language, and ARQ provides the classes for executing queries and for serializing their results into the XML result set format. Below is the WSDL generated for this query service:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name='SPARQLService'
targetNamespace='http://service.example2.ssoa.ddj.com/'
xmlns='http://schemas.xmlsoap.org/wsdl/'
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
xmlns:tns='http://service.example2.ssoa.ddj.com/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <types>
  <xs:schema
targetNamespace='http://service.example2.ssoa.ddj.com/'
version='1.0' xmlns:tns='http://service.example2.ssoa.ddj.com/'
xmlns:xs='http://www.w3.org/2001/XMLSchema'>
    <xs:element name='query' type='tns:query'/>
    <xs:element name='queryResponse' type='tns:queryResponse'/>
    <xs:complexType name='query'>
     <xs:sequence>
      <xs:element minOccurs='0' name='arg0' type='xs:string'/>
     </xs:sequence>
    </xs:complexType>
    <xs:complexType name='queryResponse'>
     <xs:sequence>
      <xs:element minOccurs='0' name='return' type='xs:string'/>
     </xs:sequence>
    </xs:complexType>
  </xs:schema>
 </types>
 <message name='SPARQL_query'>
  <part element='tns:query' name='query'/>
 </message>
 <message name='SPARQL_queryResponse'>
  <part element='tns:queryResponse' name='queryResponse'/>
 </message>
 <portType name='SPARQL'>
  <operation name='query' parameterOrder='query'>
   <input message='tns:SPARQL_query'/>
   <output message='tns:SPARQL_queryResponse'/>
  </operation>
 </portType>
 <binding name='SPARQLBinding' type='tns:SPARQL'>
  <soap:binding style='document'
transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='query'>
   <soap:operation soapAction=''/>
   <input>
    <soap:body use='literal'/>
   </input>
   <output>
    <soap:body use='literal'/>
   </output>
  </operation>
 </binding>
 <service name='SPARQLService'>
  <port binding='tns:SPARQLBinding' name='SPARQLPort'>
   <soap:address location='REPLACE_WITH_ACTUAL_URL'/>
  </port>
 </service>
</definitions>
```

Being able to expose new, richer descriptions of data and the inferences that can be produces from them via standard service protocols is a big advantage, and helps to show SOA solutions can be enhanced with a little semantic technology.

Semantic-izing Your SOA

Whereas SOA-izing your semantics brings semantically enriched services to the SOA, semantic-izing your SOA improves the fundamental infrastructure capabilities of SOA. This assists the overall management and composition of the SOA itself. We examine two such additions — a semantic facade that decouples normal SOA services from a larger, more complex and dynamic SOA service, and a semantic registry that improves on the syntax-oriented offerings by UDDI or other types of implementations.

Semantic Facade SOA Service

A semantic facade (the standard figure of which is shown in Figure 2; see http://c2.com/cgi/wiki?FacadePattern) offers a method to build robust, complex abstractions from groupings of regular SOA services. The facade might be used to provide high availability by intelligently switching between similar services, align multiple,

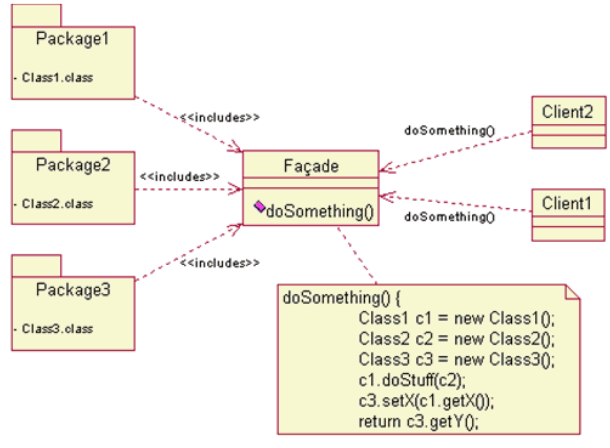


Figure 2 contains the standard Facade pattern.

similar services, or to build complex sequences of services. The semantic facade is made possible by the use of semantic technologies and the inferencing provided by a knowledgebase and associated reasoners. A semantic facade can be used to encode and effect the more efficient use of existing services based on the semantics contained in the facade layer. The capability to semantically unite services aides in managing complexity on all three fronts: service complexity, information complexity, and service management complexity:

Now the Semantic facade adds a SOA interface layer to interact with both the client and the services (here listed as Package 1 through 3). The doSomething method enables the incorporation of semantics to manage the interactions between client and the services. Figure 3 displays the logical architecture of the Semantic Facade.

The Semantic facadebrings the power of semantics to orchestrate complex service aggregations.

Semantic Registry SOA Service

Existing registries, and their shortcomings, are the Achilles' heel of today's SOAs. No awareness equals no use, and current implementations of service registries make it all too easy for services to fall through the cracks into oblivion. Of course awareness of services from back channels external to SOA can be used on a one-off basis, but this is not a sustainable solution for building a large, dynamic SOA-based application.

Awareness exists on several levels; syntax, functionality, dependencies, assumptions, and sequences. Syntax awareness provides the technical details to exchange information. Functionality awareness provides the purpose of the exchange. Dependencies and assumptions provide the background to exercise the service successfully. Finally sequence awareness provides the logical steps of multiple processes to achieve a larger goal. UDDI and similar SOA services 9 (even LDAP) provide only basic awareness — allowing insight into syntax, some keywords that describe functionality possibly linked to a taxonomy, and some ancillary information. This is critical information but clearly they would benefit from some additional, rich knowledge about the service and its relationships to information and other services. Semantics can go beyond a taxonomy to find similar services and partnering services. Semantics can maintain alignment to the services as they evolve. Additionally, the complexity of UDDI has limited its implementation and adoption. Semantics can help in managing the complexity of UDDI while also providing a richer description of the service.

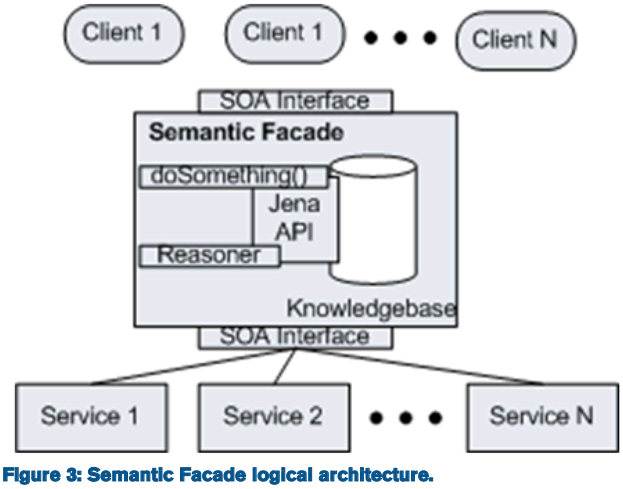


Figure 3: Semantic Facade logical architecture.

Rich semantics would even allow negotiation of services and advance composition. Areas that would enable the self-formation of a service sequence to serve higher level abstractions. This requires advanced, complex service descriptions in addition to the logic to interface properly.

Awareness can be inherent to SOA, where the SOA itself is aware of services. This awareness can improve management for the SOA services. In this case, SOAs can identify busy and non-busy services, leading to better optimization and improved resource utilization. A semantic registry could also provide governance — a key enabler of advanced, large-scale SOA implementations. A Semantic Registry need not replace any existing syntax-based registry but rather enrich and strengthen the existing registry through semantics. Several potential semantic standards exist. See OWL-S or SAWSDL to see these standardization attempts at adding semantic information to SOA services.

Summary

The Semantic Web offers a bridge to the larger goals possible for a SOA and the many available networked services. It provides the conceptual abstractions necessary to collapse the various technologies into more useful computing artifacts and concepts. Additionally, it provides software the dynamism to keep pace with changes inherent to large scale computing especially when using decentralized, heterogeneous SOA components.

The flexibility of the Semantic Web can help to move SOA from relatively small, single-enterprise efforts to massive Web-scale systems that leverage both the services and information available through the Internet. It also provides a powerful set of tools for communicating and manipulating application data, structure and constraints, to align with many different perspectives.

We outlined four examples to illustrate methods where semantic gracefully strengthens SOA. The first two focused on the ease to add semantic capabilities to the SOA service portfolio. The remaining two demonstrated ways to improve SOA itself through improved composition and registration. These examples illustrate ways to better manage complexity at the service, information, and management levels. This complexity management expands the scope and reach of a SOA solution. Using a richer representation for data and service descriptions can empower higher levels of the SOA stack to effectively tackle the ever larger and more complex challenges that the future will inevitably bring.

— John Hebeler is a division scientist at BBN Technologies. Andrew Perez-Lopez is a Software Engineer at BBN Technologies.

The Coreinfo 2.0 Utility

Understanding manycore complexity in Windows

by Gaston Hillar

Windows Server 2008 R2 and Windows 7 (64-bits version) offer new NUMA (Non-Uniform Memory Access) support. Therefore, it is very important for Windows developers to understand the differences found in the complex underlying multicore and manycore hardware. Coreinfo (http://technet.microsoft.com/en-us/sysinternals/cc835722.aspx) is a simple — yet powerful — command-line utility that shows you very useful information about the processors, their organization and the cache topology.

Mark Russinovich, a member of Windows Sysinternals team, has made the new version v2.0 of Coreinfo available for download.

This command-line utility runs on most modern Windows versions and displays information about the mapping between logical cores (logical processors or hardware threads) and the physical cores. Besides, it shows information about the NUMA nodes, groups, sockets and all the cache levels. This information is very important to understand the underlying hardware. When you benchmark multicore performance, the great differences between many multicore architectures can make it really difficult to tune the application for a specific architecture. Using this command-line utility, you can easily save the information about the underlying hardware before running your benchmarks and performance tests.

The new version supports Windows Server 2008 R2 systems with more than 64 logical processors (logical cores or hardware threads). Besides, it is also compatible with IA-64 architectures. You don't need to run an installer. You can unzip the executable file and run it from the command-line.

The utility uses the *GetLogicalProcessorInformation* Windows API function (see http://msdn.microsoft.com/en-us/library/ms683194%28VS.85%29.aspx) to obtain all the information displayed on the screen. Therefore, you can also obtain this information in your applications to tune performance according to the underlying hardware architecture. In fact, if you plan to create applications targeting manycore systems with multiple NUMA nodes, you'll have to take into account the detailed cache topology if you want to exploit the underlying hardware.

The results of running Coreinfo v2.0 on an Intel Atom N270 powered netbook are the following:

```
Logical to Physical Processor Map:
** Physical Processor 0 (Hyperthreaded)
Logical Processor to Socket Map:
** Socket 0
Logical Processor to NUMA Node Map:
** NUMA Node 0
Logical Processor to Cache Map:
** Data Cache 0, Level 1, 24 KB, Assoc 6, LineSize 64
** Instruction Cache 0, Level 1, 32 KB, Assoc 8, LineSize 64
** Unified Cache 0, Level 2, 512 KB, Assoc 8, LineSize 64
```

There is just one physical core. However, as this CPU offers Hyper-Threading technology, Coreinfo tells you it is Hyperthreaded.

The results of running Coreinfo v2.0 on an Intel Core 2 Duo P8600 powered notebook are the following:

```
Logical to Physical Processor Map:
*- Physical Processor 0
-* Physical Processor 1
Logical Processor to Socket Map:
** Socket 0
Logical Processor to NUMA Node Map:
** NUMA Node 0
Logical Processor to Cache Map:
*- Data Cache 0, Level 1, 32 KB, Assoc 8, LineSize 64
*- Instruction Cache 0, Level 1, 32 KB, Assoc 8, LineSize 64
-* Data Cache 1, Level 1, 32 KB, Assoc 8, LineSize 64
-* Instruction Cache 1, Level 1, 32 KB, Assoc 8, LineSize 64
** Unified Cache 0, Level 2, 3 MB, Assoc 12, LineSize 64
```

Coreinfo uses an asterisk "*" to represent a mapping. In this case, there are two physical cores and two logical cores as there isn't Hyper-Threading technology. Besides, there is a unified 3 MB Level 2 cache memory. Both physical cores share this cache, therefore, Coreinfo shows two asterisks "**" on the left side of the last line. This means that the cache is mapped to both processors:

```
*- =Physical Processor 0
-*=Physical Processor 1
```

Therefore, ** means Physical Processor 0 and Physical Processor 1.

The results of running Coreinfo v2.0 on an Intel Core 2 Quad Q6600 powered workstation are the following:

```
Logical to Physical Processor Map:
*--- Physical Processor 0
-*-- Physical Processor 1
--*- Physical Processor 2
---* Physical Processor 3
Logical Processor to Socket Map:
**** Socket 0
Logical Processor to NUMA Node Map:
**** NUMA Node 0
Logical Processor to Cache Map:
*--- Data Cache 0, Level 1, 32 KB, Assoc 8, LineSize 64
*--- Instruction Cache 0, Level 1, 32 KB, Assoc 8, LineSize 64
-*-- Data Cache 1, Level 1, 32 KB, Assoc 8, LineSize 64
-*-- Instruction Cache 1, Level 1, 32 KB, Assoc 8, LineSize 64
**-- Unified Cache 0, Level 2, 4 MB, Assoc 16, LineSize 64
--*- Data Cache 2, Level 1, 32 KB, Assoc 8, LineSize 64
--*- Instruction Cache 2, Level 1, 32 KB, Assoc 8, LineSize 64
---* Data Cache 3, Level 1, 32 KB, Assoc 8, LineSize 64
---* Instruction Cache 3, Level 1, 32 KB, Assoc 8, LineSize 64
--** Unified Cache 1, Level 2, 4 MB, Assoc 16, LineSize 64
Logical Processor to Group Map:
**** Group 0
```

In this case, there are four physical cores and four logical cores as there isn't Hyper-Threading technology. Besides, there are two unified 4 MB Level 2 cache memories. Each pair of physical cores

share this cache, therefore, Coreinfo shows asterisks to identify the processors mapped to each cache:

```
*---=Physical Processor 0
-*--=Physical Processor 1
--*-=Physical Processor 2
---*=Physical Processor 3
```

Therefore, **-- means Physical Processor 0 and Physical Processor 1, and --** means Physical Processor 2 and Physical Processor 3.

These are the two lines that display the information about each unified cache mapped to each pair of physical processors:

```
**-- Unified Cache 0, Level 2, 4 MB, Assoc 16, LineSize 64
--** Unified Cache 1, Level 2, 4 MB, Assoc 16, LineSize 64
```

In the aforementioned examples, there is just one NUMA node. Some of the results of running Coreinfo v2.0 on a server powered by two quad-core AMD Opteron 2379 HE microprocessors with a NUMA architecture are the following:

```
Logical to Physical Processor Map:
*------- Physical Processor 0
-*------ Physical Processor 1
--*----- Physical Processor 2
---*---- Physical Processor 3
----*--- Physical Processor 4
-----*-- Physical Processor 5
------*- Physical Processor 6
-------* Physical Processor 7
Logical Processor to Socket Map:
****---- Socket 0
----**** Socket 1
Logical Processor to NUMA Node Map:
****---- NUMA Node 0
----**** NUMA Node 1
```

In this case, Coreinfo shows very useful mapping information related to NUMA nodes.

As it is a command-line utility, it is very simple to run it and redirect its output to a text file. For example:

```
coreinfo > cpudetails.txt
```

Saves all the information to the cpudetails.txt file.

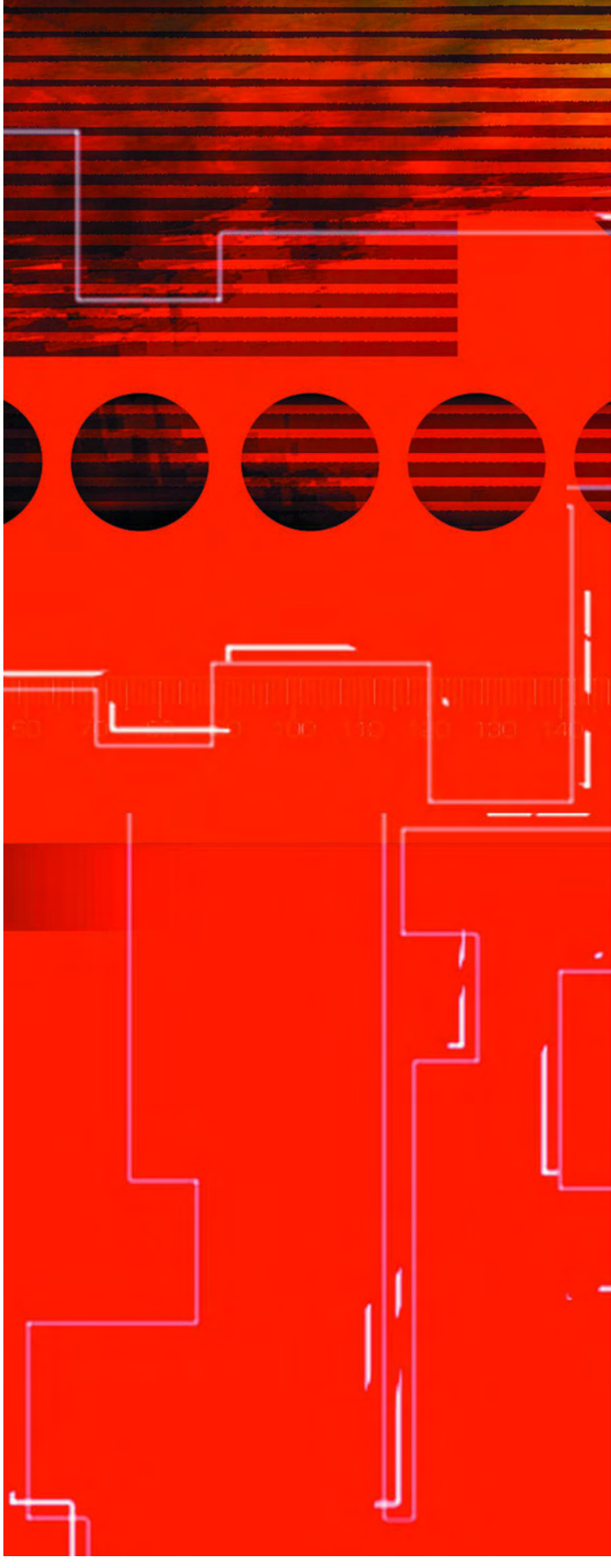
The application offers many parameters to select the information to dump:

- **-c** Dump information on cores.
- **-g** Dump information on groups.
- **-l** Dump information on caches.
- **-n** Dump information on NUMA nodes.
- **-s** Dump information on sockets.

You can download Coreinfo v2.0 at http://www.ddj.com/windows/220900822.

— Gaston Hillar is the author of C# 2008 and 2005 Threaded Programming: Beginner's Guide.

Return to Table of Contents



Combining Code Development, Modeling, and Simulation with Eclipse

A "best of both worlds" solution for open source development

by Paul Urban

Embedded systems and software developers juggle three critical challenges: rising design complexity, increasingly narrow time-to-market windows, and doing more with existing resources. To overcome these difficult issues, many development teams use the Eclipse open source platform for software development (http://www.eclipse.org/). One advantage of the Eclipse workspace is that it provides a highly customizable environment that can be tailored to the needs of individual software developers.

While working in Eclipse, many developers are also using model-driven development (MDD) to obtain automation, visualization, and abstraction capabilities to help improve team communication and design quality. MDD is becoming a mainstream technique to assist productivity and team collaboration. When your team effectively combines MDD and an Eclipse environment, you can create a powerful, customizable development environment that helps you improve overall team effectiveness.

There have been major hurdles for engineers in transitioning to a complete MDD environment with Unified Modeling Language (UML), such as giving up control over the structure of the code and learning to design at the model level. With the recent advances in MDD technology, however, engineers can now successfully use Eclipse C/C++ Development Tools (CDT; see http://www.eclipse.org/cdt/) and Java Development Tools (JDT; see http://www.eclipse.org/jdt/) to work at the code level and in a MDD environment. This approach allows users to obtain the benefits of

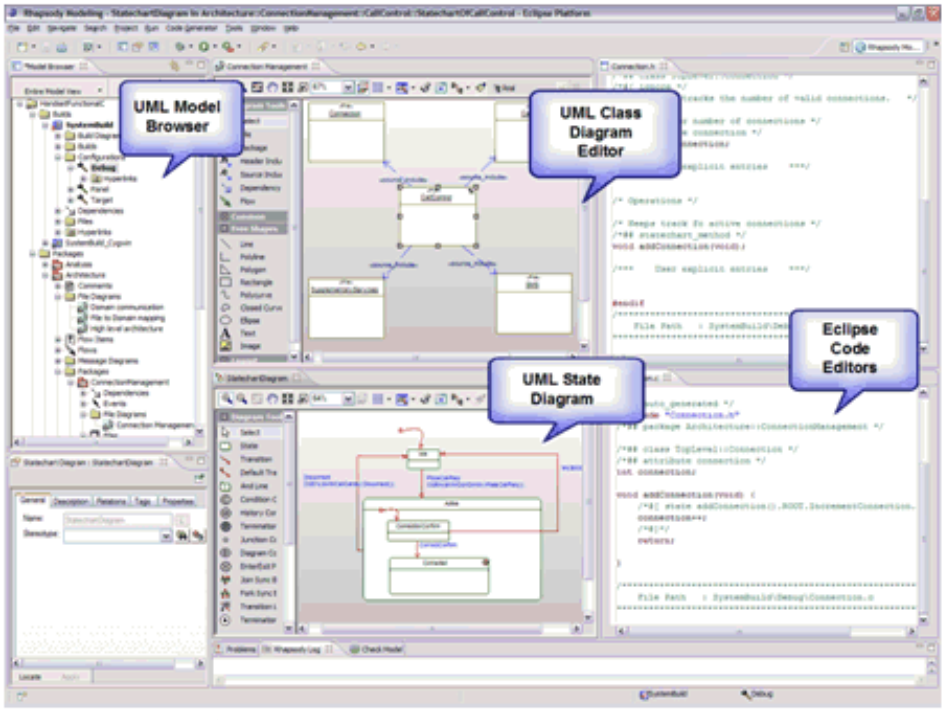


Figure 1: An example of a MDD environment embedded within the Eclipse IDE.

abstraction and automation provided by a modeling, simulation and code generation solution with C, C++ and Java development capabilities — helping users get a "best of both worlds" workflow.

In the past, MDD environments and IDE forced users to work in one or the other with no substantive interaction between the two "worlds." A developer that wanted to work in the model faced the tedious task of writing the code themselves in an IDE and then the developer would have to maintain both the models and the code since they would easily lapse out of synch with each other. Early MDD solutions — many of which still exist today — that generated code produced black box code, which is often unreadable and lacks user control of the results. Getting the black box code to look right and operate as intended was difficult at best; if the user resorted to changing the code, it was a given that the model and the code would be out of synch, leading to a major maintenance issue. Writing the code from scratch was even worse, because the model was rarely updated to reflect any changes implemented in the code. With the model and code out of synch, the value of the reusability, communication and consistency of the model was lost. Now, with a strong Eclipse IDE and a UML-based MDD environment that also offers code generation with automated synchronization of the model and code, organizations have the help they need to develop software in a combined model-based and open-source environment.

Streamlining the Workflow

The power of the combined Eclipse IDE and MDD environment is that this integration allows users to work at multiple levels. Now teams can work at the code level in the same natural workflow as before, and at the model level, with a host of benefits derived from working in a MDD framework. These benefits include transitioning to a MDD environment at the user's own pace, with reduced risks from changing to a MDD framework. Since developers can continue to work at the code level, there is a short learning curve with the solution and almost zero ramp-up time. As an additional benefit, the MDD environment automatically produces code documentation and test scenarios, easily streamlining this process.

Four important technologies make the bi-directional combination between an Eclipse-based, code-centric workflow and an MDD, model-centric workflow a reality:

- Reverse engineering of existing code into a graphical model
- Round-tripping of changes made to the code being developed back into the model
- Automatic generation of new code from the model
- Dynamic model code associativity (DMCA), which helps ensure that changes made to either the code or the model are kept in sync.

By employing these technologies, the code is automatically visualized in the modeling environment and code changes are automatically updated within the model, but the code remains in the same format.

Generating the code from the model automates your development process and avoids manually typing code. Code generated from the model accurately reflects the design, and the model can be used as a source of documentation for the code. It is possible to generate structural code such as classes, operations, and variables, but it's also possible to generate the full behavior from UML state diagrams with tools such as IBM Rational Rhapsody. The generated code can be viewed and edited within the Eclipse editors.

This combination is particularly effective when challenges to meeting project requirements arise. You can use the Eclipse IDE to write or modify the code and have changes automatically appear in the MDD environment, helping ensure that algorithms meet special characteristics including timing, safety, and security. In situations where code size is important or direct interaction with the hardware is required, this workflow is particularly effective. Additionally, a scenario where the existing code must be included in the project, such as a wish to reuse legacy code or a need to integrate third-party code, the combined Eclipse IDE and MDD environment helps provide a smart solution.

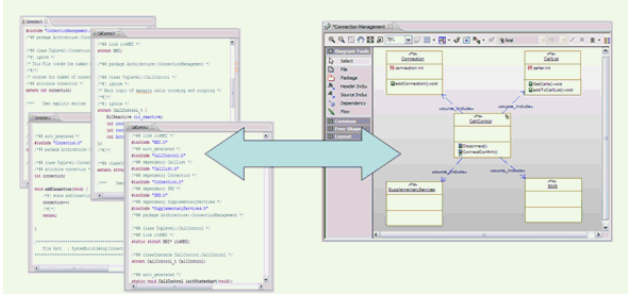


Figure 2: Model and code synchronization, reverse engineering, roundtripping and code generation help enable combining code-centric and model-centric workflowsEclipse IDE.

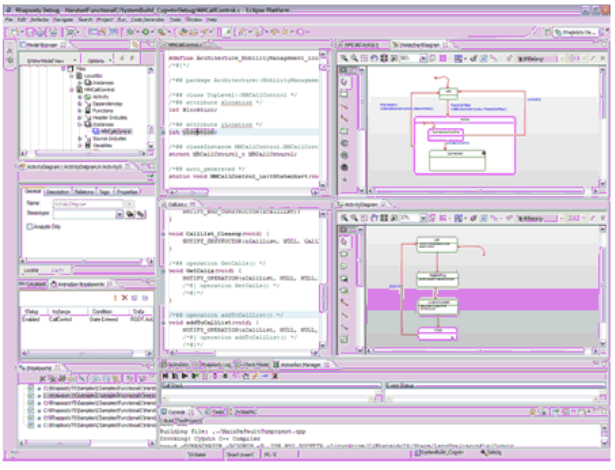


Figure 3: Simultaneous model and code level debugging can be performed highlighting model behavior and stepping through code execution

Incorporating graphical modeling into the project allows engineers and developers to abstract away from the code in order to clearly see how all the elements of the project operate together. With the ability to simulate the model and the code on the host computer during the development process, rooting out problems is done quickly at the beginning of the process, when the costs and time needed to fix these issues is at a minimum. Additionally, this allows development and testing to begin before the target hardware is available. By automatically generating the code for a majority of the new capabilities and reverse engineering any existing code into the model, engineers can shorten the development time through automation. Perhaps the most valuable feature is the self-documenting workflow. The ability to generate designs of the code being developed makes design intent much clearer to each stakeholder.

Eclipse users also leverage tools and processes within Eclipse for configuration management and team collaboration. Fortunately, users have the ability to continue to leverage their current processes on the code and model exactly the way it is done today with an extension that provides differencing and merging of multiple branches at the graphical model level. Using Eclipse to maintain and manage the source code with base-aware graphical differencing and merging, the MDD and Eclipse combination helps enable teams to collaborate in real time and work in parallel.

Obtaining maximum productivity benefits requires that the navigation from the code to the model and the model to the code is seamless and automated. The MDD and Eclipse integration achieves this by allowing users to select some code in Eclipse and automatically find and open the corresponding model element (operating in reverse, too), or by simply selecting a model element and the corresponding code is then located.

Embracing a bi-directional, code-centric and model-centric Eclipse-based approach can help your organization rapidly produce high-quality software deliverables — without adding resources — while continuing to meet challenging deadlines. Users can embrace a powerful environment when they use the Eclipse IDE and MDD integrated solution that automates software delivery, enables early design validation and maintains consistency across the product lifecycle.

— Paul Urban has more than 20 years experience in the embedded and real-time systems industry developing systems, software, and hardware. Currently, Paul works for IBM Rational as a Senior Marketing Manager for Systems and Medical Devices. He has worked with the IBM Rational Rhapsody MDD environment since 1995 in various roles including application engineer, consultant, and in product marketing. Prior to his career at IBM, Urban developed custom hardware for high-performance computing applications.

Return to Table of Contents

The Agile Social Contract

The two keys are commitment and a disciplined path for Agile rollout

by Ryan Martens

Are you preparing your organization to "go Agile"? Have you personally committed yourself to an Agile initiative? Or are you planning to have the teams figure it out and report their successes to you?

Agile software development has been around for a while. It is probably fair to say that it has crossed the chasm and is truly mainstream, at least as far as being accepted in the IT lexicon. That may be why you are now cautiously ready to make your move. But what does it really mean to "go Agile"? What books have you read? Which case studies gave you the guidance to move forward with your initiative? Do you really understand what your teams will be doing? Do you, in turn, know what YOU will be doing? What are the most important actions to take in adopting Agile? In fact, do you have anything at all to do in the adoption since you've heard it is just a development group activity with engineering practices?

Through Rally's work over the years helping organizations large and small adopt Agile, we've come to figure out some fundamental components of what success looks like. Yes, there are roadblocks and challenges. And Agile is not a silver bullet. Yet we believe Agile can bring increased productivity, higher quality, increased ROI, faster customer feedback, and higher morale. How does this happen? What is at the core of this success? Two simple tenets: commitment and a disciplined path for Agile rollout.

What I Mean by Commitment

We can understand how commitment works at the team level. Agile teams commit to delivering a set of features from a prioritized backlog every timeboxed iteration. They then make a daily commitment on what they will do toward the successful completion of their iteration commitment. At the end of the iteration, the team demos what it was able to complete of its commitment. After review and retrospection, the Agile team then moves to its next iteration, planning what its commitment will be for the next timebox. This team commitment is at the heart of Agile

Where do you fit in with this Agile work? You too must have an Agile commitment. At a high level, transition commitment is often called "executive championship," a "funded business case" or "executive buy-in." To me, these are soft commitments; they are pushed down the organization where the real commitment is enforced. In Agile, the opposite is true. What is really required for Agile success is a commitment I call "The Agile Social Contract." As the lead of an Agile rollout, you create this Agile social contract with the stakeholders and the teams, to answer the question "What is in if for us, and what is your commitment to us?"

The Agile Social Contract

This notion of a Social Contract is not mine. Isreal Gat is my mentor on the topic and forms the background of its use. It was a pleasure of mine to meet and work for Israel Gat at BMC back in 2005. I learned about his Social Contract in his office at BMC, but you can read about it on his blog in two posts "A Social Contract for Agile" and "Addition to the Social Contract." What you notice quickly about his contract is that it addresses the true elephant in the room. "If I do this for you, am I going to lose my job?" To me, this declarion of commitment hits at the source of much of the fear, uncertainty and doubt that accompanies any organizational initiative. And, as in Israel's case, it is all the more important to offer this commitment given the economic challenges occurring right now. The Agile Social Contract can't answer the employment question directly, but it does spell out the issues and commitment with empathy and consequences.

To paraphrase Israel:

> *We have to become more competitive, and Agile is the approach that does this by increasing value delivery, quality and reducing costs. It cuts cost by making the operations cheaper. If we do not do this, none of us will have a job. You might still lose your job to a cheaper labor source. But the chances are lower with good Agile skills. So, I am going to invest in you with training and professional development to increase your skills in an area that is very marketable. In return, I want you to support this effort with everything you have.*

This economic imperative may not be the setting that everyone is facing with their Agile rollout. But this type of contract is one that gets everyone rowing together. As Israel noted, this is leading from within. What if your organization is doing well and not feeling this type of economic downturn pressure to adopt Agile? I would suggest that you either find a strategic imperative that you enable through Agile. Or resign yourself to a long boat ride of trying to gain true commitment and real Agile adoption. With the commitment you offer via an Agile Social Contract such as Israel's executive level one, your teams can offer their commitment at the team level. One level of commitment sustains the other. This is key to your Agile success

A Disciplined Path for Agile Rollout

With a clear Agile social contract, the entire organization can follow a very simple, step-wise adoption process to successfully adopting agility beyond the team level. This process and its success are in your hands. Your Agile Social Contract binds you to commit to all it entails. In return, you will reap the benefits of Agile you had contracted to your organization, the stakeholders, and the teams.

Jean Tabaka and I have written and spoken at the Agile conference about Flow-Pull-Innovate, as a Lean model for successful Agile rollout. It is a simple structure that guides your scaling and maturing of Agile team to teams of teams, to the entire organization. Our five-step process works incrementally and iteratively to mature your organization's level of agility, level of discipline and realization of benefits. These steps are discrete transition states for any team, program or organization. Organizations that understand these steps are very effective at developing strategies to move states quickly and efficiently. That is, they inspect and adapt without wasteful thrashing and churn. These firms plan and experiment with changes in the tools, techniques, methods and infrastructures that prove and enable these states at a small scale before moving to the entire organization.

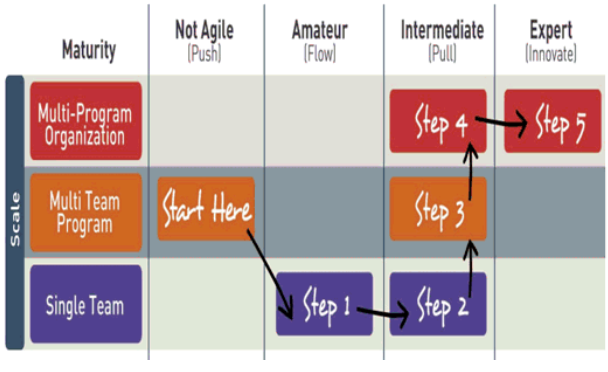


Figure 1: Pull Innovate Model for Agile Maturing and Scaling.

Setting the Agile Rollout in Motion

Realistically assessing the current capabilities for leading teams and programs is the first step in creating a roadmap of priorities for your organziation. For instance, without a clear rollout plan, groups of distributed teams can quickly start running in all directions. That equates to waste of thrashing. An effective Agile rollout map such as our five-step model provides critical focus and scope control for the entire adoption process; one step at time!

What is your role in all of this? You have created and communicated your Agile Social Contract. You are ready to follow the Flow Pull Innovate model for Agile maturing and scaling. Now what?

First, hold an Agile Rollout Planning meeting where you drive for the highest level of commitment to move to the next transition step within an agreed upon timebox. This meeting aligns with other standard Agile meetings, such as an Iteration Planning meeting, in that it drives and is driven by:

- Priorities
- Estimates
- Commitments
- Acceptance
- Adaptation

In this meeting, you first declare your vision for the organization, re-stating your Agile Social Contract for the entire organization. You and your stakeholders as the Steering Committee determine what your specific goals are for the Agile rollout given the vision. You gain consensus around these Agile adoption goals. You also gain commitment to use the Flow Pull Innovate model for your rollout. In addition, you define your roles in the overall rollout and you select the initial teams, the pilot teams, that you will support in the rollout. Finally, in this kick-off of the Agile rollout, you and your stakeholders commit to what you will complete in the Organizational Implementation backlog before the next meeting.

The Organizational Implementation Backlog

The Organizational Implementation Backlog is a clear indication of your commitment to the Agile rollout. This backlog prioritizes the work that you and the stakeholders vouch to do in order to support the teams as stated in your Agile Social Contract. The items,

also referred to as stories, have a rough costing and a sense of benefit/value that helps guide its ranking and the effort of estimate to drive the item to acceptance/completion. This means that each item must also have clearly articulated success criteria. In the Agile Rollout meeting, or prior to it, the backlog items need to be groomed. That means that they have been roughly prototyped, roughly costed and presented with clear acceptance criteria.

Going Forward

In each subsequent meeting with the Steering committee (or Rollout Team), you will follow the same set of similar steps as the initial meeting:

1. Initial presentation including a clear assessment of the current state, a review of the vision and roadmap as well as a review of the progress made.
2. Given the theme of getting to the next step, you present your prioritized list of Organizational Implementation backlog items.
3. The team debates the design, shape, estimates, feasibility and effectiveness of these ranked backlog items.
4. The team then selects the items to work on in this cycle's timebox and reviews the criteria they will use to inspect each item's progress or acceptance.
5. The team commits to get these items done in the cycle.
6. A communication plan is developed to share the plans with the rest of the organization.
7. A Steering Committee that is prepared for this meeting and that has engaged a meeting facilitator can complete this work in two to four hours. In addition to the work stated above, you may need to revisit your Agile Social Contract. It might take a few cycles before you have enough experiences and data to complete your final draft the Agile Social Contract. Once you have it however, it becomes a simple step of commitment along the Agile transition path.

Can you see the simplicity of Agile Adoption when you apply appropriate commitment and structure? A truly effective Agile Social Contract that creates true trust and commitment requires clarity and discipline. With the transparency of a clearly communicated Agile Social Contract, you will establish a strong leadership mechanism that aligns all the stakeholders and teams within your Agile adoption. Of course Enterprise-scale agile adoptions take place in a larger context of the business and market. As Israel Gat stated in his personal Agile Social Contract, we cannot guarantee lifetime employment in this globally competitive world. But, by making a clear commitment to win-win agreements, we can change the conversation into a motivating one versus a de-motivating one. Don't try to scale Agile without a real and personal commitment or without a clear rollout structure. It does not work any better than an Agile team that does not have the discipline to commit.

—Ryan is founder and CTO of Rally Software. He can be contacted at www.rallydev.com.

Return to Table of Contents

Elimination of Text Corruption in XML

Feeds have become a standard way of sharing content on the web

by Subramanian Narayanan and Ashish Arora

Feeds have become a standard way of sharing content on the web. Formats like RSS and Atom provide the feeds that we use at Yahoo! to syndicate content from the many content partners we work with.

When processing XML data acquired from our partners/feed providers, we sometimes find the markup text or HTML text (in particular character entity reference, ISO characters, or non Unicode characters) to be corrupted. If this data is passed on to the Front-end as is, it results in a bad experience for the user. To avoid this situation, we use an algorithm to process the content to eliminate data corruption.

Recognizing Types of Data Corruption

The following data sets in an XML document will result in a bad user experience:

- Double Encoded text (for example, **'** Yahoo rocks! **'**)
- Character entity reference (for example, **—**)
- Windows characters in the range of 128-159 (for example, **—**)
  * ANSI Hex characters (for example, **\x85**)

Here are ways on how these affect can the user experience.

Double Encoded Text. When the XML containing the double encoded text is rendered in the front end, we have the following result:

- Input XML Subbu**'** book
- Expected result Subbu's book
- Front-end page Subbu**'** book

where **'** is a numeric character references corresponding to an apostrophe.

Character Entity References. XML parsers recognize only five symbols -- **& " ' < >**. All other valid character entity references (like — é) are not recognizable by XML parsers and the Front-end systems currently face difficulty in rendering these character entity reference. In some cases, the document gets truncated abruptly because of these character entity references. These often occur because they are valid in HTML. However, when they are provided as XML -- not CData -- the XML parser is not able to parse them.

Windows Characters In the Range of 128-159. ASCII characters in the range of 128-159 don't render well in most browsers:

- Input XML Subbu **—** book
- Front-end page Subbu**[]** book

[] above represents the square that browsers show when they don't have an appropriate character set.

ANSI Hex Characters. The Front-end machines do not parse the ANSI Hex Characters properly. We need to normalize these data sets to valid Unicode characters to comply with our globalization standard (all in Unicode) requirement.

Not As Simple As It Seems

Dealing with this problem isn't always entirely straightforward. There are a couple of considerations that create challenges to implementing a fix for these encoding issues.

First, the complete set of characters that needs to be corrected contains around 400 individual

characters. We need a single, uniform and scalable solution to tackle the entire data set. It's also difficult to detect and eliminate double encoded characters. For example:

```
* N&amp;#39;V --> Here &#39; is double encoded.
* Arun &amp;Subbu --> Here &amp;Subbu is not double encoded.
```

It's important that any system we build recognizes the difference between the first and second example above. As such the system will need to be able to detect when an encoded ampersand is followed by a character entity.

Building a System To Fix Text Corruption

Use a normalized data set and map all characters to it. For all the categories in the data set (character entity references, Windows characters in the range of 128-159, ANSI Hex characters), we need to have a unique mapping data set for uniform conversion. The mapping set we use is that of numeric character references. For example:

```
* &copy; -> &#169; (html symbol to numeric character reference)
* &#151 -> &#8212; (ascii to numeric character references)
* \x97 -> &#8212; (hex to numeric character references)
```

Why did we choose to use numeric character references? Any XML parser can parse the numeric character references. When parsed, they are rendered without character corruption. They are also viable HTML characters which means that they can be easily re-used further down the pipeline without further translation.

Detecting and removing double encoding. The double encoding is detected by the following regular expressions:

```
* &amp;(\\w*?);
* &amp;(#\\d*?);
* &amp;(#[x|X][0-9a-fA-F]*?);
* &#38;(\\w*?);
* &#38;(#\\d*?);
* &#38;(#[x|X][0-9a-fA-F]*?);
```

The matched pattern will detect patterns like **&** or **&**. However, it will also detect other patterns like &Guru;. Hence, before we do the replacement, we check if the matched pattern is a valid XML entity. Let's take two example strings:

- `It&#39; my book`

  The pattern **'** is matched by the regex **&(#\\d*?);**. The **&** is removed from the matched pattern resulting in **'**, which is detected as being a valid XML entity; hence, it replaces **'**.

  Result - **It'** my book

- `Arun&Guru; are my friends`

  Here, **&Guru** is matched by the pattern **&#amp;(\\w*?);**. **&** is replaced by **&** and results in **&Guru;**. But since **&Guru;** is not a valid XML entity, it does not replace **&Guru**.

  Result - Arun**&Guru;** are my friends

How is XML entity validation done on the matched pattern? The matched pattern is wrapped inside an XML, which contains the DTDs for XML entities in its doctype declaration. The wrapped XML is parsed for validating the matched pattern to determine if this is an XML entity or not.

Conclusion

When taking content from third-party sites it's easy to get corruptions of the text that break the way the text is rendered to the user. We've presented the two steps you need to implement in your systems to stop the common problems of invalid XML characters and double encoding. By implementing these changes to your system, you will create a much more robust ingestion pipeline for your content.

—Ashish Arora is an engineering graduate from NIT, Trichy in India. At Yahoo! he builds platforms for the Yahoo! Properties. Subramanian Narayanan is an engineering graduate in computer science from PSG Tech, Coimbatore, India. Subramanian is part of the feed processing platform team at Yahoo! and works on distributed processing using Hadoop.

Note

The character map in full is list below:

```
{ "characterMap": { " ": " ", "&iexcl;": "&#161;",
"&cent;": "&#162;", "&pound;": "&#163;", "&curren;": "&#164;",
"&yen;": "&#165;", "&brvbar;": "&#166;", "&sect;": "&#167;",
"&uml;": "&#168;", "&copy;": "&#169;", "&ordf;": "&#170;",
"&laquo;": "&#171;", "&not;": "&#172;", "&shy;": "&#173;",
"&reg;": "&#174;", "&macr;": "&#175;", "&deg;": "&#176;",
"&plusmn;": "&#177;", "&sup2;": "&#178;", "&sup3;": "&#179;",
"&acute;": "&#180;", "&micro;": "&#181;", "&para;": "&#182;",
"&middot;": "&#183;", "&cedil;": "&#184;", "&sup1;": "&#185;",
"&ordm;": "&#186;", "&raquo;": "&#187;", "&frac14;": "&#188;",
"&frac12;": "&#189;", "&frac34;": "&#190;", "&iquest;":
"&#191;", "&Agrave;": "&#192;", "&Aacute;": "&#193;", "&Acirc;":
"&#194;", "&Atilde;": "&#195;", "&Auml;": "&#196;", "&Aring;":
"&#197;", "&AElig;": "&#198;", "&Ccedil;": "&#199;", "&Egrave;":
"&#200;", "&Eacute;": "&#201;", "&Ecirc;": "&#202;", "&Euml;":
"&#203;", "&Igrave;": "&#204;", "&Iacute;": "&#205;", "&Icirc;":
"&#206;", "&Iuml;": "&#207;", "&ETH;": "&#208;", "&Ntilde;":
"&#209;", "&Ograve;": "&#210;", "&Oacute;": "&#211;", "&Ocirc;":
"&#212;", "&Otilde;": "&#213;", "&Ouml;": "&#214;", "&times;":
"&#215;", "&Oslash;": "&#216;", "&Ugrave;": "&#217;",
"&Uacute;": "&#218;", "&Ucirc;": "&#219;", "&Uuml;": "&#220;",
"&Yacute;": "&#221;", "&THORN;": "&#222;", "&szlig;": "&#223;",
"&agrave;": "&#224;", "&aacute;": "&#225;", "&acirc;": "&#226;",
"&atilde;": "&#227;", "&auml;": "&#228;", "&aring;": "&#229;",
"&aelig;": "&#230;", "&ccedil;": "&#231;", "&egrave;": "&#232;",
"&eacute;": "&#233;", "&ecirc;": "&#234;", "&euml;": "&#235;",
"&igrave;": "&#236;", "&iacute;": "&#237;", "&icirc;": "&#238;",
"&iuml;": "&#239;", "&eth;": "&#240;", "&ntilde;": "&#241;",
"&ograve;": "&#242;", "&oacute;": "&#243;", "&ocirc;": "&#244;",
"&otilde;": "&#245;", "&ouml;": "&#246;", "&divide;": "&#247;",
"&oslash;": "&#248;", "&ugrave;": "&#249;", "&uacute;":
"&#250;", "&ucirc;": "&#251;", "&uuml;": "&#252;", "&yacute;":
"&#253;", "&thorn;": "&#254;", "&yuml;": "&#255;", "&fnof;":
"&#402;", "&Alpha;": "&#913;", "&Beta;": "&#914;", "&Gamma;":
"&#915;", "&Delta;": "&#916;", "&Epsilon;": "&#917;", "&Zeta;":
"&#918;", "&Eta;": "&#919;", "&Theta;": "&#920;", "&Iota;":
"&#921;", "&Kappa;": "&#922;", "&Lambda;": "&#923;", "&Mu;":
"&#924;", "&Nu;": "&#925;", "&Xi;": "&#926;", "&Omicron;":
"&#927;", "&Pi;": "&#928;", "&Rho;": "&#929;", "&Sigma;":
"&#931;", "&Tau;": "&#932;", "&Upsilon;": "&#933;", "&Phi;":
"&#934;", "&Chi;": "&#935;", "&Psi;": "&#936;", "&Omega;":
"&#937;", "&alpha;": "&#945;", "&beta;": "&#946;", "&gamma;":
"&#947;", "&delta;": "&#948;", "&epsilon;": "&#949;", "&zeta;":
"&#950;", "&eta;": "&#951;", "&theta;": "&#952;", "&iota;":
"&#953;", "&kappa;": "&#954;", "&lambda;": "&#955;", "&mu;":
"&#956;", "&nu;": "&#957;", "&xi;": "&#958;", "&omicron;":
"&#959;", "&pi;": "&#960;", "&rho;": "&#961;", "&sigmaf;":
"&#962;", "&sigma;": "&#963;", "&tau;": "&#964;", "&upsilon;":
"&#965;", "&phi;": "&#966;", "&chi;": "&#967;", "&psi;":
```

```
"&#968;", "&omega;": "&#969;", "&thetasym;": "&#977;",
"&upsih;": "&#978;", "&piv;": "&#982;", "&bull;": "&#8226;",
"&hellip;": "&#8230;", "&prime;": "&#8242;", "&Prime;":
"&#8243;", "&oline;": "&#8254;", "&frasl;": "&#8260;",
"&weierp;": "&#8472;", "&image;": "&#8465;", "&real;":
"&#8476;", "&trade;": "&#8482;", "&alefsym;": "&#8501;",
"&larr;": "&#8592;", "&uarr;": "&#8593;", "&rarr;": "&#8594;",
"&darr;": "&#8595;", "&harr;": "&#8596;", "&crarr;": "&#8629;",
"&lArr;": "&#8656;", "&uArr;": "&#8657;", "&rArr;": "&#8658;",
"&dArr;": "&#8659;", "&hArr;": "&#8660;", "&forall;": "&#8704;",
"&part;": "&#8706;", "&exist;": "&#8707;", "&empty;": "&#8709;",
"&nabla;": "&#8711;", "&isin;": "&#8712;", "&notin;": "&#8713;",
"&ni;": "&#8715;", "&prod;": "&#8719;", "&sum;": "&#8721;",
"&minus;": "&#8722;", "&lowast;": "&#8727;", "&radic;":
"&#8730;", "&prop;": "&#8733;", "&infin;": "&#8734;", "&ang;":
"&#8736;", "&and;": "&#8743;", "&or;": "&#8744;", "&cap;":
"&#8745;", "&cup;": "&#8746;", "&int;": "&#8747;", "&there4;":
"&#8756;", "&sim;": "&#8764;", "&cong;": "&#8773;", "&asymp;":
"&#8776;", "&ne;": "&#8800;", "&equiv;": "&#8801;", "&le;":
"&#8804;", "&ge;": "&#8805;", "&sub;": "&#8834;", "&sup;":
"&#8835;", "&nsub;": "&#8836;", "&sube;": "&#8838;", "&supe;":
"&#8839;", "&oplus;": "&#8853;", "&otimes;": "&#8855;",
"&perp;": "&#8869;", "&sdot;": "&#8901;", "&lceil;": "&#8968;",
"&rceil;": "&#8969;", "&lfloor;": "&#8970;", "&rfloor;":
"&#8971;", "&lang;": "&#9001;", "&rang;": "&#9002;", "&loz;":
"&#9674;", "&spades;": "&#9824;", "&clubs;": "&#9827;",
"&hearts;": "&#9829;", "&diams;": "&#9830;", "&quot;": "&#34;",
"&amp;": "&#38;", "&lt;": "&#60;", "&gt;": "&#62;", "&OElig;":
"&#338;", "&oelig;": "&#339;", "&Scaron;": "&#352;", "&scaron;":
"&#353;", "&Yuml;": "&#376;", "&circ;": "&#710;", "&tilde;":
"&#732;", " ": " ", " ": " ", " ":
" ", "&zwnj;": "&#8204;", "&zwj;": "&#8205;", "&lrm;":
"&#8206;", "&rlm;": "&#8207;", "&ndash;": "&#8211;", "&mdash;":
"&#8212;", "&lsquo;": "&#8216;", "&rsquo;": "&#8217;",
"&sbquo;": "&#8218;", "&ldquo;": "&#8220;", "&rdquo;":
"&#8221;", "&bdquo;": "&#8222;", "&dagger;": "&#8224;",
"&Dagger;": "&#8225;", "&permil;": "&#8240;", "&lsaquo;":
"&#8249;", "&rsaquo;": "&#8250;", "&euro;": "&#8364;", "&#x21;":
"&#33;", "&#x22;": "&#34;", "&#x23;": "&#35;", "&#x24;":
"&#36;", "&#x25;": "&#37;", "&#x26;": "&#38;", "&#x27;":
"&#39;", "&#x28;": "&#40;", "&#x29;": "&#41;", "&#x2A;":
"&#42;", "&#x2B;": "&#43;", "&#x2C;": "&#44;", "&#x2D;":
"&#45;", "&#x2E;": "&#46;", "&#x2F;": "&#47;", "&#x30;":
"&#48;", "&#x31;": "&#49;", "&#x32;": "&#50;", "&#x33;":
"&#51;", "&#x34;": "&#52;", "&#x35;": "&#53;", "&#x36;":
"&#54;", "&#x37;": "&#55;", "&#x38;": "&#56;", "&#x39;":
"&#57;", "&#x3A;": "&#58;", "&#x3B;": "&#59;", "&#x3C;":
"&#60;", "&#x3D;": "&#61;", "&#x3E;": "&#62;", "&#x3F;":
"&#63;", "&#x40;": "&#64;", "&#x41;": "&#65;", "&#x42;":
"&#66;", "&#x43;": "&#67;", "&#x44;": "&#68;", "&#x45;":
"&#69;", "&#x46;": "&#70;", "&#x47;": "&#71;", "&#x48;":
"&#72;", "&#x49;": "&#73;", "&#x4A;": "&#74;", "&#x4B;":
"&#75;", "&#x4C;": "&#76;", "&#x4D;": "&#77;", "&#x4E;":
"&#78;", "&#x4F;": "&#79;", "&#x50;": "&#80;", "&#x51;":
"&#81;", "&#x52;": "&#82;", "&#x53;": "&#83;", "&#x54;":
"&#84;", "&#x55;": "&#85;", "&#x56;": "&#86;", "&#x57;":
"&#87;", "&#x58;": "&#88;", "&#x59;": "&#89;", "&#x5A;":
"&#90;", "&#x5B;": "&#91;", "&#x5C;": "&#92;", "&#x5D;":
"&#93;", "&#x5E;": "&#94;", "&#x5F;": "&#95;", "&#x60;":
"&#96;", "&#x61;": "&#97;", "&#x62;": "&#98;", "&#x63;":
"&#99;", "&#x64;": "&#100;", "&#x65;": "&#101;", "&#x66;":
"&#102;", "&#x67;": "&#103;", "&#x68;": "&#104;", "&#x69;":
"&#105;", "&#x6A;": "&#106;", "&#x6B;": "&#107;", "&#x6C;":
"&#108;", "&#x6D;": "&#109;", "&#x6E;": "&#110;", "&#x6F;":
"&#111;", "&#x70;": "&#112;", "&#x71;": "&#113;", "&#x72;":
"&#114;", "&#x73;": "&#115;", "&#x74;": "&#116;", "&#x75;":
"&#117;", "&#x76;": "&#118;", "&#x77;": "&#119;", "&#x78;":
"&#120;", "&#x79;": "&#121;", "&#x7A;": "&#122;", "&#x7B;":
"&#123;", "&#x7C;": "&#124;", "&#x7D;": "&#125;", "&#x7E;":
"&#126;", "&#x20AC;": "&#8364;", "&#x201A;": "&#8218;",
"&#x0192;": "&#402;", "&#x201E;": "&#8222;", "&#x2026;":
"&#8230;", "&#x2020;": "&#8224;", "&#x2021;": "&#8225;",
"&#x02C6;": "&#710;", "&#x2030;": "&#8240;", "&#x0160;":
"&#352;", "&#x2039;": "&#8249;", "&#x0152;": "&#338;",
"&#x017D;": "&#381;", "&#x2018;": "&#8216;", "&#x2019;":
"&#8217;", "&#x201C;": "&#8220;", "&#x201D;": "&#8221;",
"&#x2022;": "&#8226;", "&#x2013;": "&#8211;", "&#x2014;":
"&#8212;", "&#x02DC;": "&#732;", "&#x2122;": "&#8482;",
"&#x0161;": "&#353;", "&#x203A;": "&#8250;", "&#x0153;":
"&#339;", "&#x017E;": "&#382;", "&#x0178;": "&#376;", " ":
" ", "&#xA1;": "&#161;", "&#xA2;": "&#162;", "&#xA3;":
"&#163;", "&#xA4;": "&#164;", "&#xA5;": "&#165;", "&#xA6;":
"&#166;", "&#xA7;": "&#167;", "&#xA8;": "&#168;", "&#xA9;":
"&#169;", "&#xAA;": "&#170;", "&#xAB;": "&#171;", "&#xAC;":
"&#172;", "&#xAD;": "&#173;", "&#xAE;": "&#174;", "&#xAF;":
"&#175;", "&#xB0;": "&#176;", "&#xB1;": "&#177;", "&#xB2;":
"&#178;", "&#xB3;": "&#179;", "&#xB4;": "&#180;", "&#xB5;":
"&#181;", "&#xB6;": "&#182;", "&#xB7;": "&#183;", "&#xB8;":
"&#184;", "&#xB9;": "&#185;", "&#xBA;": "&#186;", "&#xBB;":
"&#187;", "&#xBC;": "&#188;", "&#xBD;": "&#189;", "&#xBE;":
"&#190;", "&#xBF;": "&#191;", "&#xC0;": "&#192;", "&#xC1;":
"&#193;", "&#xC2;": "&#194;", "&#xC3;": "&#195;", "&#xC4;":
"&#196;", "&#xC5;": "&#197;", "&#xC6;": "&#198;", "&#xC7;":
"&#199;", "&#xC8;": "&#200;", "&#xC9;": "&#201;", "&#xCA;":
"&#202;", "&#xCB;": "&#203;", "&#xCC;": "&#204;", "&#xCD;":
"&#205;", "&#xCE;": "&#206;", "&#xCF;": "&#207;", "&#xD0;":
"&#208;", "&#xD1;": "&#209;", "&#xD2;": "&#210;", "&#xD3;":
"&#211;", "&#xD4;": "&#212;", "&#xD5;": "&#213;", "&#xD6;":
"&#214;", "&#xD7;": "&#215;", "&#xD8;": "&#216;", "&#xD9;":
"&#217;", "&#xDA;": "&#218;", "&#xDB;": "&#219;", "&#xDC;":
"&#220;", "&#xDD;": "&#221;", "&#xDE;": "&#222;", "&#xDF;":
"&#223;", "&#xE0;": "&#224;", "&#xE1;": "&#225;", "&#xE2;":
"&#226;", "&#xE3;": "&#227;", "&#xE4;": "&#228;", "&#xE5;":
"&#229;", "&#xE6;": "&#230;", "&#xE7;": "&#231;", "&#xE8;":
"&#232;", "&#xE9;": "&#233;", "&#xEA;": "&#234;", "&#xEB;":
"&#235;", "&#xEC;": "&#236;", "&#xED;": "&#237;", "&#xEE;":
"&#238;", "&#xEF;": "&#239;", "&#xF0;": "&#240;", "&#xF1;":
"&#241;", "&#xF2;": "&#242;", "&#xF3;": "&#243;", "&#xF4;":
"&#244;", "&#xF5;": "&#245;", "&#xF6;": "&#246;", "&#xF7;":
"&#247;", "&#xF8;": "&#248;", "&#xF9;": "&#249;", "&#xFA;":
"&#250;", "&#xFB;": "&#251;", "&#xFC;": "&#252;", "&#xFD;":
"&#253;", "&#xFE;": "&#254;", "&#xFF;": "&#255;", "\\x21":
"&#33;", "\\x22": "&#34;", "\\x23": "&#35;", "\\x24": "&#36;",
"\\x25": "&#37;", "\\x26": "&#38;", "\\x27": "&#39;", "\\x28":
"&#40;", "\\x29": "&#41;", "\\x2A": "&#42;", "\\x2B": "&#43;",
"\\x2C": "&#44;", "\\x2D": "&#45;", "\\x2E": "&#46;", "\\x2F":
"&#47;", "\\x30": "&#48;", "\\x31": "&#49;", "\\x32": "&#50;",
"\\x33": "&#51;", "\\x34": "&#52;", "\\x35": "&#53;", "\\x36":
"&#54;", "\\x37": "&#55;", "\\x38": "&#56;", "\\x39": "&#57;",
"\\x3A": "&#58;", "\\x3B": "&#59;", "\\x3C": "&#60;", "\\x3D":
"&#61;", "\\x3E": "&#62;", "\\x3F": "&#63;", "\\x40": "&#64;",
"\\x41": "&#65;", "\\x42": "&#66;", "\\x43": "&#67;", "\\x44":
"&#68;", "\\x45": "&#69;", "\\x46": "&#70;", "\\x47": "&#71;",
"\\x48": "&#72;", "\\x49": "&#73;", "\\x4A": "&#74;", "\\x4B":
"&#75;", "\\x4C": "&#76;", "\\x4D": "&#77;", "\\x4E": "&#78;",
"\\x4F": "&#79;", "\\x50": "&#80;", "\\x51": "&#81;", "\\x52":
"&#82;", "\\x53": "&#83;", "\\x54": "&#84;", "\\x55": "&#85;",
"\\x56": "&#86;", "\\x57": "&#87;", "\\x58": "&#88;", "\\x59":
"&#89;", "\\x5A": "&#90;", "\\x5B": "&#91;", "\\x5C": "&#92;",
"\\x5D": "&#93;", "\\x5E": "&#94;", "\\x5F": "&#95;", "\\x60":
"&#96;", "\\x61": "&#97;", "\\x62": "&#98;", "\\x63": "&#99;",
"\\x64": "&#100;", "\\x65": "&#101;", "\\x66": "&#102;",
"\\x67": "&#103;", "\\x68": "&#104;", "\\x69": "&#105;",
"\\x6A": "&#106;", "\\x6B": "&#107;", "\\x6C": "&#108;",
"\\x6D": "&#109;", "\\x6E": "&#110;", "\\x6F": "&#111;",
"\\x70": "&#112;", "\\x71": "&#113;", "\\x72": "&#114;",
"\\x73": "&#115;", "\\x74": "&#116;", "\\x75": "&#117;",
"\\x76": "&#118;", "\\x77": "&#119;", "\\x78": "&#120;",
"\\x79": "&#121;", "\\x7A": "&#122;", "\\x7B": "&#123;",
"\\x7C": "&#124;", "\\x7D": "&#125;", "\\x7E": "&#126;",
"\\x80": "&#8364;", "\\x82": "&#8218;", "\\x83": "&#402;",
"\\x84": "&#8222;", "\\x85": "&#8230;", "\\x86": "&#8224;",
"\\x87": "&#8225;", "\\x88": "&#710;", "\\x89": "&#8240;",
"\\x8A": "&#352;", "\\x8B": "&#8249;", "\\x8C": "&#338;",
"\\x8E": "&#381;", "\\x91": "&#8216;", "\\x92": "&#8217;",
"\\x93": "&#8220;", "\\x94": "&#8221;", "\\x95": "&#8226;",
"\\x96": "&#8211;", "\\x97": "&#8212;", "\\x98": "&#732;",
"\\x99": "&#8482;", "\\x9A": "&#353;", "\\x9B": "&#8250;",
"\\x9C": "&#339;", "\\x9E": "&#382;", "\\x9F": "&#376;",
"\\xA1": "&#161;", "\\xA2": "&#162;", "\\xA3": "&#163;",
"\\xA4": "&#164;", "\\xA5": "&#165;", "\\xA6": "&#166;",
"\\xA7": "&#167;", "\\xA8": "&#168;", "\\xA9": "&#169;",
"\\xAA": "&#170;", "\\xAB": "&#171;", "\\xAC": "&#172;",
"\\xAD": "&#173;", "\\xAE": "&#174;", "\\xAF": "&#175;",
"\\xB0": "&#176;", "\\xB1": "&#177;", "\\xB2": "&#178;",
"\\xB3": "&#179;", "\\xB4": "&#180;", "\\xB5": "&#181;",
"\\xB6": "&#182;", "\\xB7": "&#183;", "\\xB8": "&#184;",
"\\xB9": "&#185;", "\\xBA": "&#186;", "\\xBB": "&#187;",
"\\xBC": "&#188;", "\\xBD": "&#189;", "\\xBE": "&#190;",
"\\xBF": "&#191;", "\\xC0": "&#192;", "\\xC1": "&#193;",
"\\xC2": "&#194;", "\\xC3": "&#195;", "\\xC4": "&#196;",
"\\xC5": "&#197;", "\\xC6": "&#198;", "\\xC7": "&#199;",
"\\xC8": "&#200;", "\\xC9": "&#201;", "\\xCA": "&#202;",
"\\xCB": "&#203;", "\\xCC": "&#204;", "\\xCD": "&#205;",
"\\xCE": "&#206;", "\\xCF": "&#207;", "\\xD0": "&#208;",
"\\xD1": "&#209;", "\\xD2": "&#210;", "\\xD3": "&#211;",
"\\xD4": "&#212;", "\\xD5": "&#213;", "\\xD6": "&#214;",
"\\xD7": "&#215;", "\\xD8": "&#216;", "\\xD9": "&#217;",
"\\xDA": "&#218;", "\\xDB": "&#219;", "\\xDC": "&#220;",
"\\xDD": "&#221;", "\\xDE": "&#222;", "\\xDF": "&#223;",
"\\xE0": "&#224;", "\\xE1": "&#225;", "\\xE2": "&#226;",
"\\xE3": "&#227;", "\\xE4": "&#228;", "\\xE5": "&#229;",
"\\xE6": "&#230;", "\\xE7": "&#231;", "\\xE8": "&#232;",
"\\xE9": "&#233;", "\\xEA": "&#234;", "\\xEB": "&#235;",
"\\xEC": "&#236;", "\\xED": "&#237;", "\\xEE": "&#238;",
"\\xEF": "&#239;", "\\xF0": "&#240;", "\\xF1": "&#241;",
"\\xF2": "&#242;", "\\xF3": "&#243;", "\\xF4": "&#244;",
"\\xF5": "&#245;", "\\xF6": "&#246;", "\\xF7": "&#247;",
"\\xF8": "&#248;", "\\xF9": "&#249;", "\\xFA": "&#250;",
"\\xFB": "&#251;", "\\xFC": "&#252;", "\\xFD": "&#253;",
"\\xFE": "&#254;", "\\xFF": "&#255;", "&#128;": "&#8364;",
"&#130;": "&#8218;", "&#131;": "&#402;", "&#132;": "&#82222;",
"&#133;": "&#8230;", "&#134;": "&#8224;", "&#135;": "&#8225;",
"&#136;": "&#710;", "&#137;": "&#8240;", "&#138;": "&#352;",
"&#139;": "&#8249;", "&#140;": "&#338;", "&#142;": "&#381;",
"&#145;": "&#8216;", "&#146;": "&#8217;", "&#147;": "&#8220;",
"&#148;": "&#8221;", "&#149;": "&#8226;", "&#150;": "&#8211;",
"&#151;": "&#8212;", "&#152;": "&#732;", "&#153;": "&#8482;",
"&#154;": "&#353;", "&#155;": "&#8250;", "&#156;": "&#339;",
"&#158;": "&#382;", "&#159;": "&#376;", }}
```

Return to Table of Contents

A Build System for Complex Projects: Part 4

Generating a full-fledged Visual Studio build system for a non-trivial system involves multiple projects

by Gigi Sayfan

This is the fourth article in a series of articles that explore an innovative build system for complicated projects. Part 1 (http://www.ddj.com/architect/218400678) and Part 2 (http://www.ddj.com/architect/219000123) discussed build systems in general and the internals of the ideal build system that can integrate with existing build systems. Part 3 (http://www.ddj.com/architect/220100417) discussed in detail how to the ideal build system works with the NetBeans IDE and can generate its build files. This article will do the same for Microsoft Visual Studio.

Generating the Visual Studio Build System

As you recall, Isaac the development manager became a true Invisible Build System (ibs) convert after seeing ibs in action. He gave Bob the mandate to use ibs on Windows to build the company's top-secret project: "Hello World - Enterprise Platinum Edition". The Windows developers of the company sweat by Visual Studio. Visual C++ supports a makefile-like build environment via the NMAKE tool, but it is not very common. Visual Studio provides both an IDE-based build environment for C/C++ projects as well as several alternatives for automated builds from the command line (vcbuild.exe, Visual Studio automation and extensibility object model, direct invocation of devenv.exe). The build files shared by all these approaches (except NMAKE) are the project and the solution. There is one project file per logical project and it encapsulates all the information about a project (same as the Makefile + nbproject directory of NetBeans). The solution is a collection projects and it corresponds the project group of NetBeans.

Figure 1 shows the Visual Studio IDE with the various Hello World projects organized in folders (apps, dlls, hw, and test).

The Visual Studio Build System Anatomy

The Visual Studio build files are considerably simpler then NetBeans. There is a single project file, which is a pretty straight forward XML file and there is a solution file, which uses (unfortunately) a proprietary text format. Figure 2 shows the project properties page for the hello_world application. There many many options and settings in GUI and most of them have default values. The project file contains only the settings that differ from the defaults (and settings that don't have defaults and must be specified). The format of the .vcproj file is documented at http://msdn.microsoft.com/en-us/library/2208a1f2%28VS.71%29.aspx.

Project File (.vcproj)

The entire build information for a project is stored in a single file. Let's examine the project file for the main hello_world application. I'll analyze it section by section. It all starts with an XML 1.0 tag to indicate it is an XML file and then there is a *VisualStudioProject* element with various attributes. The important ones are the project type (Visual C++), the version (9.00 for VC++ 2008), the name ("hello_world") and the ProjectGUID, which uniquely identifies this project.

```
<?xml version="1.0" encoding="UTF-8"?>
<VisualStudioProject
  ProjectType="Visual C++"
  Version="9.00"
  Name="hello_world"
  ProjectGUID="{88AD54BE-4316-4DFB-965E-4369A2910DF8}"
  RootNamespace="hello_world"
  Keyword="Win32Proj"
  TargetFrameworkVersion="0">
```

The next section is the platforms section, which determines the target platforms. In this case just Win32:

```
<Platforms>
        <Platform
                Name="Win32"
        />
*</Platforms>
```

There is an empty *ToolFiles* element. This element can point to custom build rules files. It is straightforward to use custom build rules from the IDE, but doing it programmatically is not documented very well (try http://reader.feedshow.com/show_items-feed=74de1c4a6c2c74a16b8d7eb75cd0f393 if you must). This capability can be added easily to ibs in a cross-platform way via a callback mechanism where ibs will call back a provided function before/after building each project.

```
<ToolFiles> </ToolFiles>
```

The *Configurations* element is a collection of *Configuration* elements. Each configuration element contains a list of *Tool* elements where each tool is a program that participates in the build process. The most common and important ones are the compiler and linker. Here is the *Debug* configuration. The *Release* configuration is very similar:

```
<Configurations>
        <Configuration
                Name="Debug|Win32"
                OutputDirectory="Debug"
                IntermediateDirectory="Debug"
                ConfigurationType="1"
                >
                <Tool
                Name="VCPreBuildEventTool"
                />
                <Tool
                        Name="VCCustomBuildTool"
                />
                <Tool
                        Name="VCXMLDataGeneratorTool"
                        />
                        <Tool
                Name="VCWebServiceProxyGeneratorTool"
                        />
                        <Tool
                                Name="VCMIDLTool"
                        />
                        <Tool
                                Name="VCCLCompilerTool"
                                Optimization="0"
 AdditionalIncludeDirectories=".;../..;../../../
   3rd_party/include/win32/"
      PreprocessorDefinitions="WIN32;_DEBUG;_CONSOLE"
                                MinimalRebuild="true"
                                BasicRuntimeChecks="3"
                                RuntimeLibrary="1"
                                UsePrecompiledHeader="0"
                                WarningLevel="3"
                Detect64BitPortabilityProblems="true"
                                DebugInformationFormat="4"
                                />
                                <Tool
                Name="VCManagedResourceCompilerTool"
                                />
                                <Tool
                                Name="VCResourceCompilerTool"
                                />
                                <Tool
                                        Name="VCPreLinkEventTool"
                                />
                                <Tool
                                        Name="VCLinkerTool"
                                        LinkIncremental="2"
 AdditionalLibraryDirectories="..\..\..\3rd_party\lib\win32"
                                IgnoreAllDefaultLibraries="false"
                                IgnoreDefaultLibraryNames=""
                                GenerateDebugInformation="true"
                                        SubSystem="1"
                                        TargetMachine="1"
                                />
                                <Tool
                                        Name="VCALinkTool"
                                />
                                <Tool
                                        Name="VCManifestTool"
                                />
                                <Tool
                                        Name="VCXDCMakeTool"
                                />
                                <Tool
                                        Name="VCBscMakeTool"
                                />
                                <Tool
                                        Name="VCFxCopTool"
                                />
                                <Tool
                                        Name="VCAppVerifierTool"
                                />
                                <Tool
                                Name="VCPostBuildEventTool"
                                />
                </Configuration>
                <Configuration
```

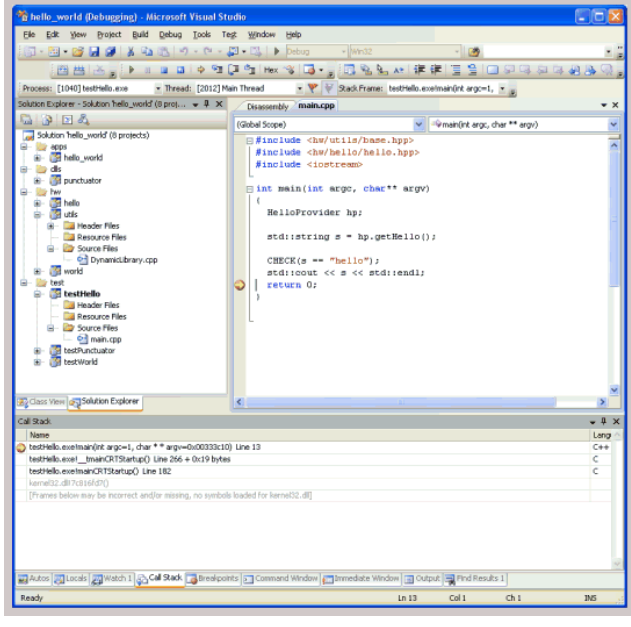


Figure 1: Visual Studio IDE.

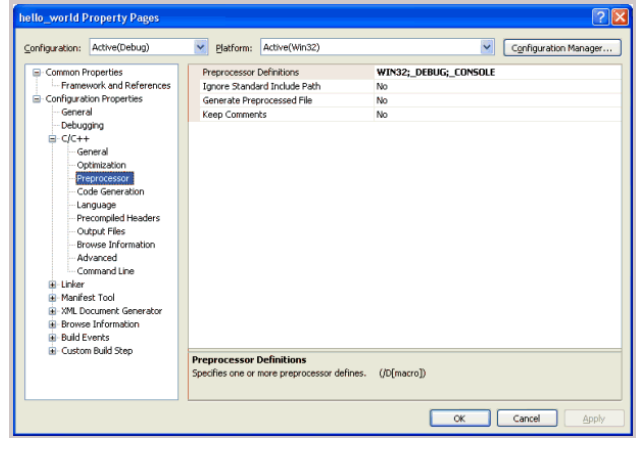


Figure 2: Project properties.

```
                Name="Release|Win32"
```

```
                        OutputDirectory="Release"
                        IntermediateDirectory="Release"
                        ConfigurationType="1"
                        >
                        ...
                </Configuration>
        </Configurations>
```

The *References* element may contain references to other projects that the current project depends on. The referenced projects will be built before the current project. This is mostly critical for static libraries that need to be linked into an executable or DLL. But, there is also an alternative way of specifying dependencies through the solution file. The same project may belong to multiple solutions (possibly with different references/dependencies). Using the *References* element in the .vcproj file is not as flexible, but keeps the dependencies with the rest of the project metadata. In this case, I chose to capture the dependencies in the solutions file, so the references element is empty.

```
<References>
</References>
```

The *Files* element simply contains all the project files. There are several filters like *Header Files, Resource Files* and *Source Files*. The filters are mostly important for user interface purposes because the different file types are grouped into folders based on the filter. For building purposes the compiler needs to know what extension to use for source files, because these are the files that are actually compiled (header files are always *#include*d by some source file). Files can be specified using relative path or absolute path. It is almost always better to use relative paths, so the same project file can be used in different locations by different users. Also, with ibs every project file resides under the project directory.

```
  <Files>
    <Filter
      Name="Header Files"
      Filter="h;hpp;hxx;hm;inl;inc;xsd"
      UniqueIdentifier="{93995380-89BD-4b04-88EB-625FBE52EBFB}"
      >
      <File
        RelativePath=".\another_file.hpp"
        >
      </File>
    </Filter>
    <Filter
      Name="Resource Files"

Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx"
      UniqueIdentifier="{67DA6AB6-F800-4c08-8B7A-83BB121AAD01}"
      >
    </Filter>
    <Filter
      Name="Source Files"
      Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx"
      UniqueIdentifier="{4FC737F1-C7A5-4376-A066-2A32D752A2FF}"
      >
      <File
        RelativePath=".\another_file.cpp"
        >
      </File>
      <File
        RelativePath=".\main.cpp"
        >
      </File>
    </Filter>
  </Files>
```

The *Globals* element allows definition of global objects. I'm not sure what are they and how they are supposed to be used. I never had the need for any global object. ibs simply generates an empty *Globals* element:

```
<Globals>
</Globals>
```

Finally, the closing tag of the .vcproj file:

```
</VisualStudioProject>
```

Solution File(.sln)

The solution file organizes all the projects in folders and optionally stores dependencies too. At the solution level, dependencies are more than just build dependencies. As you recall build dependencies are always executables or dynamic libraries that link against static libraries. But, there are other dependencies, too. If you have an executable *E* that loads a dynamic library *D* you want to make sure that *D* is up to date when you test *E*, so for testing purposes you may want to add a dependency of *E* on *D*. This will cause *D* to be built before *E* is built (although the order doesn't matter) and you can be confident that you test the same version of *E* and *D*.

Back to the solution file, the format is proprietary but fairly simple. The main concept is the "Project", which can be either a Visual Studio project (captured in a .vcproj file in the case of ibs) or a virtual folder that contains a number of other projects. The folders are not file system folders, but are used in the Visual Studio IDE for organizational purposes. Folders can be nested and can contain other folders or actual projects. Each project (either a real project or a folder) has a GUID (globally unique identifier) associated with it. The .sln file is using the GUIDs to refer to projects. The reason is that the solution may contain projects with identical names and it is easier to distinguish between them by GUID then by absolute path to a project file, which may not work in case of relative paths.

The file format consists of project elements that includes the project dependencies if any followed by a few global sections that determine the folder nesting and what projects participate in the build.

The dependencies are specified as GUIDs. The path to the project file is specified if there is a project file. Here is the project section of the "testHello" test project:

```
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "testHello",
     "test\testHello\testHello.vcproj",
     "{5F46CA1C-88BC-4E19-BB65-8686B453441D}"
  ProjectSection(ProjectDependencies) = postProject
     {D0736B61-D7AE-4B50-99FF-1AC604AF83D1} =
       {D0736B61-D7AE-4B50-99FF-1AC604AF83D1}
     {7A0F57A3-00B8-4879-BBE1-318E3FC0A526} =
       {7A0F57A3-00B8-4879-BBE1-318E3FC0A526}
  EndProjectSection
EndProject
```

In case of a folder, the folder name is used instead of a path to the project file and there are no dependencies:

```
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") =
   "test", "test", "{64EECA44-A21D-4FB3-8C2C-3D3B81EE2F3C}"
EndProject
```

The *Global* part of the file contains multiple global sections. The configurations section contains the available configurations

(by default *Debug* and *Release*) and which ones should be built. It is divided into two global sections marked *preSolution* and *postSolution*.

```
GlobalSection(SolutionConfigurationPlatforms) = preSolution
  Debug|Win32 = Debug|Win32
  Release|Win32 = Release|Win32
EndGlobalSection
GlobalSection(ProjectConfigurationPlatforms) = postSolution
  {5F46CA1C-88BC-4E19-BB65-8686B453441D}.Debug|Win32.ActiveCfg
    = Debug|Win32
  {5F46CA1C-88BC-4E19-BB65-8686B453441D}.Debug|Win32.Build.0 =
    Debug|Win32
  {5F46CA1C-88BC-4E19-BB65-8686B453441D}.Release|Win32.ActiveCfg =
    Release|Win32
  {5F46CA1C-88BC-4E19-BB65-8686B453441D}.Release|Win32.Build.0 =
    Release|Win32
  ...

EndGlobalSection
```

Next there is a little section that determines if in the IDE the solution itself will have a node in the tree or if it's just going to be a list of projects/folders:

```
GlobalSection(SolutionProperties) = preSolution
          HideSolutionNode = FALSE
EndGlobalSection
```

The last section specifies the nesting of the projects inside folders. It is a clever way to specify arbitrarily nested structure in a linear format. Both the parent and the child are specified using their GUIDs, so it's pretty difficult to figure out what project is in what folder. Of course, this file is not intended for direct viewing:

```
GlobalSection(NestedProjects) = preSolution
  {5F46CA1C-88BC-4E19-BB65-8686B453441D} =
     {64EECA44-A21D-4FB3-8C2C-3D3B81EE2F3C}
  {48933983-2311-4966-A33E-06B47FE88B6A} =
     {64EECA44-A21D-4FB3-8C2C-3D3B81EE2F3C}
  {6D1F69E3-575B-4BB9-8B5F-D295916A2C3B} =
     {64EECA44-A21D-4FB3-8C2C-3D3B81EE2F3C}
  {B5183A0D-18E4-4288-8DB0-60183460677E} =
     {8A47B373-446F-42A7-83BB-EFED3017940F}
  {88AD54BE-4316-4DFB-965E-4369A2910DF8} =
     {55B446A3-CAA3-4EFB-BA53-2232048BF417}
  {D0736B61-D7AE-4B50-99FF-1AC604AF83D1} =
     {0C7A17A3-B93D-43AE-A166-1DDAAE8E2B46}
  {7A0F57A3-00B8-4879-BBE1-318E3FC0A526} =
     {0C7A17A3-B93D-43AE-A166-1DDAAE8E2B46}
  {C99C4A67-8323-4CD7-B049-354E019994C8} =
     {0C7A17A3-B93D-43AE-A166-1DDAAE8E2B46}
EndGlobalSection
```

Let's try and follow one such nesting relation:

```
{7A0F57A3-00B8-4879-BBE1-318E3FC0A526} =
    {0C7A17A3-B93D-43AE-A166-1DDAAE8E2B46}
```

The first GUID belongs to the *utils* project.

```
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") =
   "utils", "hw\utils\utils.vcproj",
     "{7A0F57A3-00B8-4879-BBE1-318E3FC0A526}"
EndProject
```

By the way, the GUID in *Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}")* denotes the project type (static library in this case). The project's GUID is the one following the project file path.

The second GUID belongs to the *hw* folder:

```
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") =
   "hw", "hw", "{0C7A17A3-B93D-43AE-A166-1DDAAE8E2B46}"
EndProject
```

So, the nesting relation says that the *utils* project is contained in the *hw* folder.

The Visual Studio Helper

The VC++ 2008 *Helper* class is responsible for the VC++ specific code used to generate the .vcproj file for every project. It is equivalent to the NetBeans 6 *Helper* class. The generic build_system_generator.py script is using this helper to generate the .vcproj file and the solution (.sln) file. Let's take a closer look at this class. The first thing it does is import some useful system modules and then import the *BaseHelper* and *Template* classes from the build_system_generator module (as well as the '*title*' function for debugging purposes):

```
#!/usr/bin/env python
import os
import sys
import glob
import string
import uuid
from pprint import pprint as pp

sys.path.insert(0,
os.path.join(os.path.abspath(os.path.dirname(__file__)), '../'))
from build_system_generator import (BaseHelper,
                                    Template,
                                    title)
```

Then, there are a couple of utility functions for handling GUIDs. The *make_guid()* function simply creates a new GUID. Conveniently enough, Python has a module called uuid that can generate GUIDs (and much more). Handy modules like uuid are exactly why Python earned the "Batteries Included" reputation. In any other language, you would have to go and hunt for a 3rd party library (or even worse... implement GUID generation yourself), download it, test it, integrate it into your code and your deployment/packaging script and hope it's not too buggy.

```
def make_guid():
  title()
  return '{' + str(uuid.uuid4()) + '}'
```

The *get_guid()* function extracts the the GUID of a project from an existing .vcproj file or creates a new one if the file doesn't exist.

```
def get_guid(filename):
  title(additional=filename)
  if os.path.isfile(filename):
    lines = open(filename).readlines()
    guid_line = lines[5]
    assert 'ProjectGUID=' in guid_line
    g = guid_line.split('=')[1][1:-2]
  else:
    g = make_guid()

  return g.upper()
```

The *Helper* class itself subclasses *BaseHelper* to benefit from all its common functionality. The *__init__()* method initializes the templates dir and sets the path separator to a back slash. This is not strictly necessary. Windows can actually work with back and forward slashes and even mix them in the same path. This is a valid path on Windows: "c:/a/b\c\d". However, for esthetic and readability purposes it is best to have a uniform convention and on

Windows the back slash is more prevalent. The separator is used when constructing paths.

```
class Helper(BaseHelper):
  """VC++ 2008 helper
  """
  def __init__(self, templates_dir):
    BaseHelper.__init__(self, templates_dir)
    self.sep = '\\'
```

The *get_templates()* method is pretty simple and returns a list containing a single *Template* object with the template type (program, static library or dynamic library), the path and the relative path of the .vcproj file.

```
  def get_templates(self, template_type):
    """Get all the template files associated with a particular
         template type

    require multiple build files per template type

    @param template_type - 'program', 'dynamic_lib' or 'static_lib'
    @return a Template object
    """
    result = []
    vcproj_file = os.path.join(self.templates_dir,
                               template_type,
                               '%s.vcproj' % template_type)
    assert os.path.isfile(vcproj_file)
    relative_path = '%s.vcproj'
    template = Template(vcproj_file, relative_path, template_type)
    return [template]
```

Often there will be just one template file, but some build systems

The *prepare_substitution_dict()* is the essential method that prepares the values that the generic *ibs* uses to populate the template for the .vcproj file. It is much simpler than the corresponding *NetBeans* method because it needs to generate just one file and not six and also the dynamic information that needs to be substituted in is concentrated in a few places in a uniform way. There are only three placeholders: GUID, *SourceFiles* and *HeaderFiles*. All the other information is encoded in the templates. Here is signature:

```
  def prepare_substitution_dict(self,
                                project_name,
                                project_type,
                                project_file_template,
                                project_dir,
                                libs_dir,
                                dependencies,
                                source_files,
                                header_files,
                                platform):
```

The *prepare_substitution_dict()* method uses a nested function called *make_files_section()* to prepare the *SourceFiles* and *HeaderFiles* lists. This function sorts the file lists too (using a case insensitive custom compare function). Note the recursive nature of this operation to create the files section a mini-template is populated with the file's relative path for each file. The result is an XML fragment that can later be embedded directly in the .vcproj file:

```
    def make_files_section(file_list):
      def icase_cmp(s1, s2):
        return cmp(s1.lower(), s2.lower())
      file_template = """\
\t\t\t<File
\t\t\t\tRelativePath=".\%s"
\t\t\t\t>
\t\t\t7<</File>"""

      if file_list == []:
        return ''
      file_list = sorted(file_list, icase_cmp)
      files = [file_template % os.path.basename(f) for f in file_list]
      return '\n'.join(files) + '\n'
```

The code of *prepare_substitution_dict()* itself is trivial. It prepares the filename and then gets the GUID from the *get_guid()* function and the *SourceFiles* and *HeaderFiles* from the nested *make_files_section()* function and just populates the result *dict*:

```
    filename = os.path.join(project_dir, project_name + '.vcproj')
    return dict(GUID=get_guid(filename),
                SourceFiles=make_files_section(source_files),
                HeaderFiles=make_files_section(header_files))
```

The *generate_workspace_files()* is much more complicated in Visual Studio. It generates the solution file for the entire system. I'll walk you through it because there is a lot going on and it could be hard to figure it out just by staring at the code. It takes as input the solution name, the root path and a list of Project objects and starts iterating over all the sub-directories under the root path using Python's excellent *os.walk()* function that returns a 3-tuple for each directory under the root path that includes the current directory, its sub-directories and its files. That allows complete iteration of every file and directory. The Visual Studio Helper class supports the notion of folders. As always ibs uses convention over configuration. The convention is that a project must be a direct sub-directory of a folder. So, to figure out the folders automatically all the sub-directories are iterated and whenever a directory that contains a .vcproj file is found its parent must be a folder. Here is the code to iterate over all the sub-directories.

```
def generate_workspace_files(self, solution_name, root_path, projects):
    """Generate a VC++ 2008 solution file

    """
    title()
    folders = {}
    for d, subdirs, files in os.walk(root_path):
      if os.path.dirname(d) != root_path:
        continue
      folder_projects = []
      for s in subdirs:
        ...
```

The project list is provided so non-project directories are skipped. The path to the .vcproj file is constructed and the project GUID is extracted. The correct paths to the dependencies of the current project are computed. Finally a SolutionItem object is constructed that contains all the relevant information of the project and the appended to the list of *folder_projects*.

```
        project_dir = os.path.join(d, s)
        if not project_dir in projects:
          continue
        vcproj_filename = os.path.join(project_dir,
                                       os.path.basename(s) + '.vcproj')
        assert os.path.isfile(vcproj_filename)

        guid = get_guid(vcproj_filename)

        # Get the directories of of all the dependency projects
        proj_dependencies = projects[project_dir].dependencies

        # Get the GUIDs of all the dependency projects
        dependencies = []
        for dep in proj_dependencies:
          basename = os.path.basename(dep)
          dep_path = os.path.join(dep, basename + '.vcproj')
          dependencies.append(get_guid(dep_path))

        si = SolutionItem(item_type=project_type,
```

```
                        name=s,
                    path=vcproj_filename,
                guid=guid,
                dependencies=dependencies,
                projects=[])
        folder_projects.append(si)
```

If the *folder_projects* list is not empty then a folder *SolutionItem* is created that contains all the folder's project. The guid for a folder is just a dummy '?'. After all the folder objects are constructed the *make_solution()* function is called, which actually generates the .sln file from all the information collected so far and the .sln file is saved to disk.

```
    guid = '?'
    if folder_projects != []:
      name = os.path.basename(d)
      folder = SolutionItem(name=name,
                            item_type=folder_type,
                            path=None,
                            guid=guid,
                            dependencies=[],
                            projects=folder_projects)
      folders[os.path.basename(d)] = folder

gen_solution = make_solution(root_path, folders)
solution_filename = os.path.join(root_path, solution_name + '.sln')
open(solution_filename, 'w').write(gen_solution)
```

The *make_solution()* uses several mini-templates to construct different parts of the .sln file. Here are the templates. The names pretty much explain the purpose of each template. The templates use the same principle as the project templates and are just segments of text with placeholder for substitution values that the *make_solution()* function populates with the proper values and weave together:

```
# A project template has a header and a list of project sections
# such as ProjectDependencies. The ProjectDependencies
# duplicate the dependency information in .vcproj files in VS2005.
# For generating a solution that contains only C++ projects, no other
# project section is needed.
project_template_with_dependencies = """\
Project("${TypeGUID}") = "${Name}", "${Filename}", "${GUID}"
          ProjectSection(ProjectDependencies) = postProject
${ProjectDependencies}
          EndProjectSection
EndProject
"""
project_template_without_dependencies = """\
Project("${TypeGUID}") = "${Name}", "${Filename}", "${GUID}"
EndProject
"""
project_configuration_platform_template = """\
\t\t${GUID}.Debug|Win32.ActiveCfg = Debug|Win32
\t\t${GUID}.Debug|Win32.Build.0 = Debug|Win32
\t\t${GUID}.Release|Win32.ActiveCfg = Release|Win32
\t\t${GUID}.Release|Win32.Build.0 = Release|Win32
"""
# This is the solution template for VS 2008
# The template arguments are:
#
# Projects
# ProjectConfigurationPlatforms
# NestedProjects
#
solution_template = """
Microsoft Visual Studio Solution File, Format Version 10.00
# Visual Studio 2008
${Projects}
Global
\tGlobalSection(SolutionConfigurationPlatforms) = preSolution
\t\tDebug|Win32 = Debug|Win32
\t\tRelease|Win32 = Release|Win32
\tEndGlobalSection
\tGlobalSection(ProjectConfigurationPlatforms) = postSolution
${Configurations}
\tEndGlobalSection
\tGlobalSection(SolutionProperties) = preSolution
\t\tHideSolutionNode = FALSE
\tEndGlobalSection
\tGlobalSection(NestedProjects) = preSolution
${NestedProjects}
\tEndGlobalSection
EndGlobal
"""
```

Whoever designed the Visual Studio build system was big on GUIDs. Almost every object is identified by a GUID including the types of various solution items like folders and projects:

```
# Guids for regular project and solution folder
project_type = '{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}'
folder_type = '{2150E333-8FDC-42A3-9474-1A3956D46DE8}'
```

The *SolutionItem* class itself is really just a named tuple, but since ibs is not limited to Python 2.6 and up (when named tuples were introduced into the language) I use a dedicated class:

```
class SolutionItem(object):
  """Represents a solution folder or project

  The set of solution projects contain all the information
  necessary to generate a solution file.

  name - the name of the project/folder
  type - folder_type or project_type
  path - the relative path from the root dir to the .vcproj file
    for projects,
          same as name for folders
  guid - the GUID of the project/folder
  dependencies - A list of project guids the project depends on.
    It is empty for folders and projects with no dependencies.

  projects - list of projects hosted by the folder. It is empty
    for projects.
  """
  def __init__(self, item_type, name, path, guid, dependencies,
      projects):
    title()
    self.name = name
    self.type = item_type
    self.path = path
    self.guid = guid
    self.dependencies = dependencies
    self.projects = projects
```

The *make_solution()* takes the source directory and the folders list to generate the solution file using a bunch of nested functions.

```
def make_solution(source_dir, folders):
  """Return a string representing the .sln file

  It uses a lot of nested functions to make the different parts
  of a solution file:
   - make_project_dependencies
   - make_projects
   - make_configurations
   - make nested_projects

  @param folders - a dictionary whose keys are VS folders and
    the values
    are the projects each folder contains. Each project must be
    an object that
    has a directory path (relative to the root dir), a guid and a
    list of dependencies (each dependency is another projects).
This directory
    should contain a .vcproj file whose name matches the
    directory name.
  @param projects - a list of projects that don't have a folder
    and are contained
    directly by the solution node.
  """
```

The *get_existing_folders()* nested function takes and existing .sln file and extracts the GUIDs of every project in it. It returns a dictionary of project names and GUIDs that can be used to regenerate a .sln file with identical GUIDs to the existing ones.

```
  def get_existing_folders(sln_filename):
    title()
    lines = open(sln_filename).readlines()
    results = {}
    for line in lines:
```

```
    if line.startswith('Project("{2150E333-8FDC-42A3-9474-
        1A3956D46DE8}") ='):
      tokens = line.split('"')
      print tokens
      name = tokens[-4]
      guid = tokens[-2]
      results[name] = guid

  return results
```

The *make_project_dependencies()* nested function takes a list of dependency GUIDs of a project and returns the text fragment that is the *ProjectDependencies* sub-section of this project in the .sln file.

```
def make_project_dependencies(dependency_guids):
  title()
  if dependency_guids == []:
    return ''

  result = []
  for g in dependency_guids:
    result.append('\t\t%s = %s' % (g, g))

  result = '\n'.join(result)
  return result
```

The *make_projects()* nested function takes the source directory and the list of projects and generates a text fragment that represents all the projects in the .sln file. It uses the micro-templates defined earlier and the *make_project_dependencies()* function.

```
def make_projects(source_dir, projects):
  title()
  result = ''
  t1 = string.Template(project_template_with_dependencies)
  t2 = string.Template(project_template_without_dependencies)
  for p in projects:
    if p.type == project_type:
      filename = p.path[len(source_dir) + 1:].replace('/', '\\')
    else:
      filename = p.name
 dependency_guids = [get_guid(p.path) for d in p.dependencies]
    guid = get_guid(filename) if p.guid is None else p.guid
    d = dict(TypeGUID=p.type,
             Name=p.name,
             Filename=filename,
             GUID=guid,
 ProjectDependencies=make_project_dependencies(p.dependencies))
    t = t1 if p.dependencies != [] else t2
    s = t.substitute(d)
    result += s

  return result[:-1]
```

The *make_configurations()* function returns a text fragment that represents all the project configuration platforms. It works by iterating over the projects list and populating the *project_configuration_platform* template with each project's GUID.

```
def make_configurations(projects):
  title()
  result = ''
  t = string.Template(project_configuration_platform_template)
  for p in projects:
    d = dict(GUID=p.guid)
    s = t.substitute(d)
    result += s

  return result[:-1]
```

The *make_nested_projects()* function returns a text fragment that represents all the nested projects in the .sln file. It works by iterating over the folders and populating the *nested_project* template with the guids of each nested project and its containing folder. Each folder is an object that has *guid* attribute and a *projects* attribute (which is a list of its contained projects):

```
def make_nested_projects(folders):
  title()
  for f in folders.values():
    assert hasattr(f, 'guid') and type(f.guid) == str
    assert hasattr(f, 'projects') and type(f.projects) in
     (list, tuple)

  result = ''
  nested_project = '\t\t${GUID} = ${FolderGUID}\n'
  t = string.Template(nested_project)
  for folder in folders.values():
    for p in folder.projects:
      d = dict(GUID=p.guid, FolderGUID=folder.guid)
      s = t.substitute(d)
      result += s

  return result[:-1]
```

These were all the nested functions and here is how the containing *make_solution()* function puts them to good use.

```
try:
  sln_filename = glob.glob(os.path.join(source_dir, '*.sln'))[0]
  existing_folders = get_existing_folders(sln_filename)
except:
  existing_folders = []

# Use folders GUIDs from existing .sln file (if there is any)
for name, f in folders.items():
  if name in existing_folders:
    f.guid = existing_folders[name]
  else:
    f.guid = make_guid()

# Prepare a flat list of all projects
all_projects =[]
for f in folders.values():
  all_projects.append(f)
  all_projects += f.projects

# Prepare the substitution dict for the solution template
projects = [p for p in all_projects if p.type == project_type]
all_projects = make_projects(source_dir, all_projects)

configurations = make_configurations(projects)
nested_projects = make_nested_projects(folders)
d = dict(Projects=all_projects,
         Configurations=configurations,
         NestedProjects=nested_projects)

# Create the final solution text by substituting the dict into
  the template
t = string.Template(solution_template)
solution = t.substitute(d)

return solution
```

The Visual Studio Project Templates

The project templates as you recall are the text files with some place holders that ibs populates with the values from the substitution dictionaries to generate the final .vcproj files. There are three different types of projects: static library, dynamic library, and a program. Each project type has its own template.

To create the template files I simply took the .vcproj file for each type of project I created manually and replaced anything that was project-specific (like the source files or list of dependencies) with a place holder. Let's examine one of the template files. Here is the template for a static library. The name of the file is static_lib.vcproj. The template is just an XML file and the place holders are *${Name}*, *${GUID}*, *${HeaderFiles}* and *${SourceFiles}*. Note the element named "*VCCLCompilerTool*." This element contains all information necessary to compile the static library. Static libraries have no link information so there is no linker tool. In general, there are many other tools supported by the Visual Studio .vcproj file format but most of them are not used for building cross platform C++ projects.

```
<?xml version="1.0" encoding="UTF-8"?>
<VisualStudioProject
  ProjectType="Visual C++"
  Version="9.00"
  Name="${Name}"
  ProjectGUID="${GUID}"
  RootNamespace="${Name}"
  Keyword="Win32Proj"
  TargetFrameworkVersion="0"
  >
  <Platforms>
    <Platform
      Name="Win32"
    />
  </Platforms>
  <ToolFiles>
  </ToolFiles>
  <Configurations>
    <Configuration
      Name="Debug|Win32"
      OutputDirectory="Debug"
      IntermediateDirectory="Debug"
      ConfigurationType="4"
      >
      <Tool
        Name="VCCLCompilerTool"
        Optimization="0"
  AdditionalIncludeDirectories=".;../..;../../../3rd_party/
  include/win32/"
        PreprocessorDefinitions="WIN32;_DEBUG;_LIB"
        MinimalRebuild="true"
        BasicRuntimeChecks="3"
        RuntimeLibrary="1"
        UsePrecompiledHeader="0"
        WarningLevel="3"
        Detect64BitPortabilityProblems="true"
        DebugInformationFormat="4"
      />
    </Configuration>
    <Configuration
      Name="Release|Win32"
      OutputDirectory="Release"
      IntermediateDirectory="Release"
      ConfigurationType="4"
      >
      <Tool
        Name="VCCLCompilerTool"
  AdditionalIncludeDirectories=".;../..;../../../3rd_party/
  include/win32/"
        PreprocessorDefinitions="WIN32;NDEBUG;_LIB"
        RuntimeLibrary="0"
        UsePrecompiledHeader="0"
        WarningLevel="3"
        Detect64BitPortabilityProblems="true"
        DebugInformationFormat="3"
      />
    </Configuration>
  </Configurations>
  <References>
  </References>
  <Files>
    <Filter
      Name="Header Files"
      Filter="h;hpp;hxx;hm;inl;inc;xsd"
      UniqueIdentifier="{93995380-89BD-4b04-88EB-625FBE52EBFB}"
      >
${HeaderFiles}                </Filter>
    <Filter
      Name="Resource Files"

Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx"
      UniqueIdentifier="{67DA6AB6-F800-4c08-8B7A-83BB121AAD01}"
      >
    </Filter>
    <Filter
      Name="Source Files"
      Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx"
      UniqueIdentifier="{4FC737F1-C7A5-4376-A066-2A32D752A2FF}"
      >
${SourceFiles}                </Filter>
  </Files>
  <Globals>
  </Globals>
</VisualStudioProject>
```

Testing the Visual Studio-Generated Build System

Bob finished the implementation of the VC++ 2008 component of ibs and tested it on Windows XP, Vista and Windows 7. First, he generated all the Visual Studio build files using the build_system_generator.py script:

```
PS Z:\ibs> python .\build_system_generator.py --
   build_system=VC_2008

--------------------

generate_build_files

--------------------

platform: win32

----------------------

_populate_project_list

----------------------

----

test

----

----

dlls

----

----

apps

----

--

hw

--

-----------------

generate_projects

-----------------

-------------

save_projects

-------------

------------------------

generate_workspace_files

------------------------

apps

dlls

hw

test

-------------

make_solution

-------------

--------------------

get_existing_folders

--------------------

['test',
```

```
 'testHello',
 'testPunctuator',
 'testWorld',
 'dlls',
 'punctuator',
 'apps',
 'hello_world',
 'hw',
 'hello',
 'utils',
 'world']
-------------
make_projects
-------------
--------------------
make_configurations
--------------------
--------------------
make_nested_projects
--------------------
```

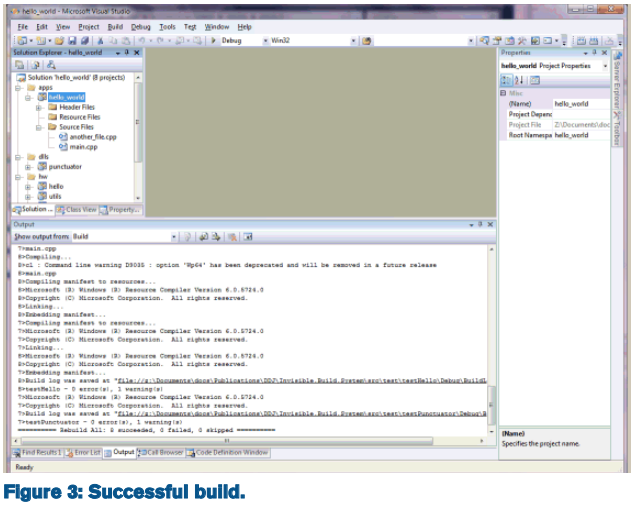


Figure 3: Successful build.

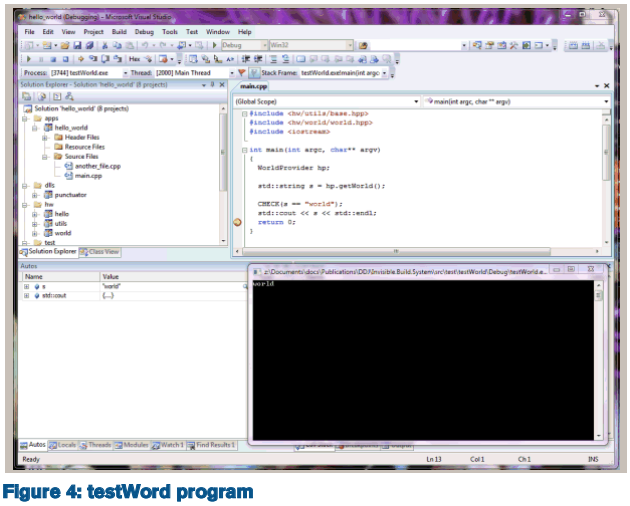


Figure 4: testWord program

Bob verified that the necessary .vcproj and .sln files were created and proceeded to build the solution. He started with a command-line build using the vcbuild.exe program. This program is normally located for Visual Studio 2008 in : "c:\Program Files\Microsoft Visual Studio 9.0\VC\vcpackages".

To build the hello world solution you can just pass the hello_world.sln filename to vcbuild. Here is the short PowerShell snippet Bob ran in the src directory to build hello_world:

```
$vcbuild = "c:\Program Files\Microsoft Visual Studio
    9.0\VC\vcpackages\vcbuild.exe"
& $vcbuild hello_world.sln
```

Isaac barged in as usual and wanted to witness the Windows tests first hand. Bob copied the punctuator.dll from the dlls\punctuator\Debug directory to apps\hello_world\Debug and ran the hello_world.exe application that was built by vcbuild.exe:

```
PS <root dir>\src\apps\hello_world\Debug>
cp..\..\..\dlls\punctuator\Debug\punctuator.dll .

PS <root dir>\src\apps\hello_world\Debug> .\hello_world.exe

hello, world!

Done.
```

Isaac was duly impressed, but wanted to verify that the solution can be built from the Visual Studio IDE too. Bob launched a new instance of Visual Studio and loaded the generated hello_world.sln solution. It then built it successfully (see Figure 3).

Next, Bob ran the testWorld program from within Visual Studio and put a breakpoint to demonstrate that ibs produces code that can be debugged properly (See Figure 4).

Isaac decided that ibs proved itself to be a strong cross-platform build system. Hw wanted to see it deployed and used to build and develop the "Hello World - Enterprise Platinum Edition". Bob was very excited and assured him that ibs is ready to go.

Conclusion

In this article you saw ibs in action, generating a full fledged VC++ 2008 solution for a non-trivial system that involves multiple projects, static libraries, shared libraries, applications and test programs. ibs handled well multiple target Windows operating systems (Windows XP, Vista and 7) and allowed building and testing from the Visual Studio IDE or externally from the command-line (using vcbuild.exe). Bob demonstrated ibs successfully to Isaac his manager and in the next episode, Bob will deploy ibs in the field and will wrestle with real-world issues and requirements.

— Gigi Sayfan specializes in cross-platform object-oriented programming in C/C++/ C#/Python/Java with emphasis on large-scale distributed systems. He is currently trying to build intelligent machines inspired by the brain at Numenta (www.numenta.com).

Return to Table of Contents

Q&A: Semantic Integration: Meeting the Challenge

A semantic integration veteran takes time out to talk

by Jonathan Erickson

Richard Keller is a senior research computer scientist and group lead for the information sharing and integration group at NASA. He recently spoke with Dobb's editor in chief Jonathan Erickson.

Dr. Dobb's: What's the fundamental problem with data integration?

Keller: Because many large organizations generate data in a distributed and organic fashion, they can amass hundreds or thousands of separate, seemingly disconnected data sources in varying formats. Those data sources will remain disconnected unless their contents are properly documented and annotated.

Dr. Dobb's: Will standards help?

Keller: Standards and organizational policies go some way toward supporting data interoperability, but data standards can be difficult to legislate and are onerous and expensive to institute. Also, many companies focus on setting data formatting standards but neglect standards requiring any type of semantic metadata, which are even more important when it comes to robust integration.

Dr. Dobb's: What's the key to semantic integration?

Keller: The key to automated integration is to be rigorous about capturing semantic metadata. If you describe the meaning of the data, then you can automate the process of recognizing connections across data sources and allow them to be married together properly.

Dr. Dobb's: Does the W3C's SPARQL solve any problems?

Keller: SPARQL provides a standardized language for querying ontologies. To the extent that this standard achieves widespread adoption, it will consolidate the marketplace and increase tool interoperability within the semantic web space. But the challenges in semantic integration lie in the construction, not in the querying, of ontologies.

Dr. Dobb's: What's the next big hurdle to achieving semantic integration?

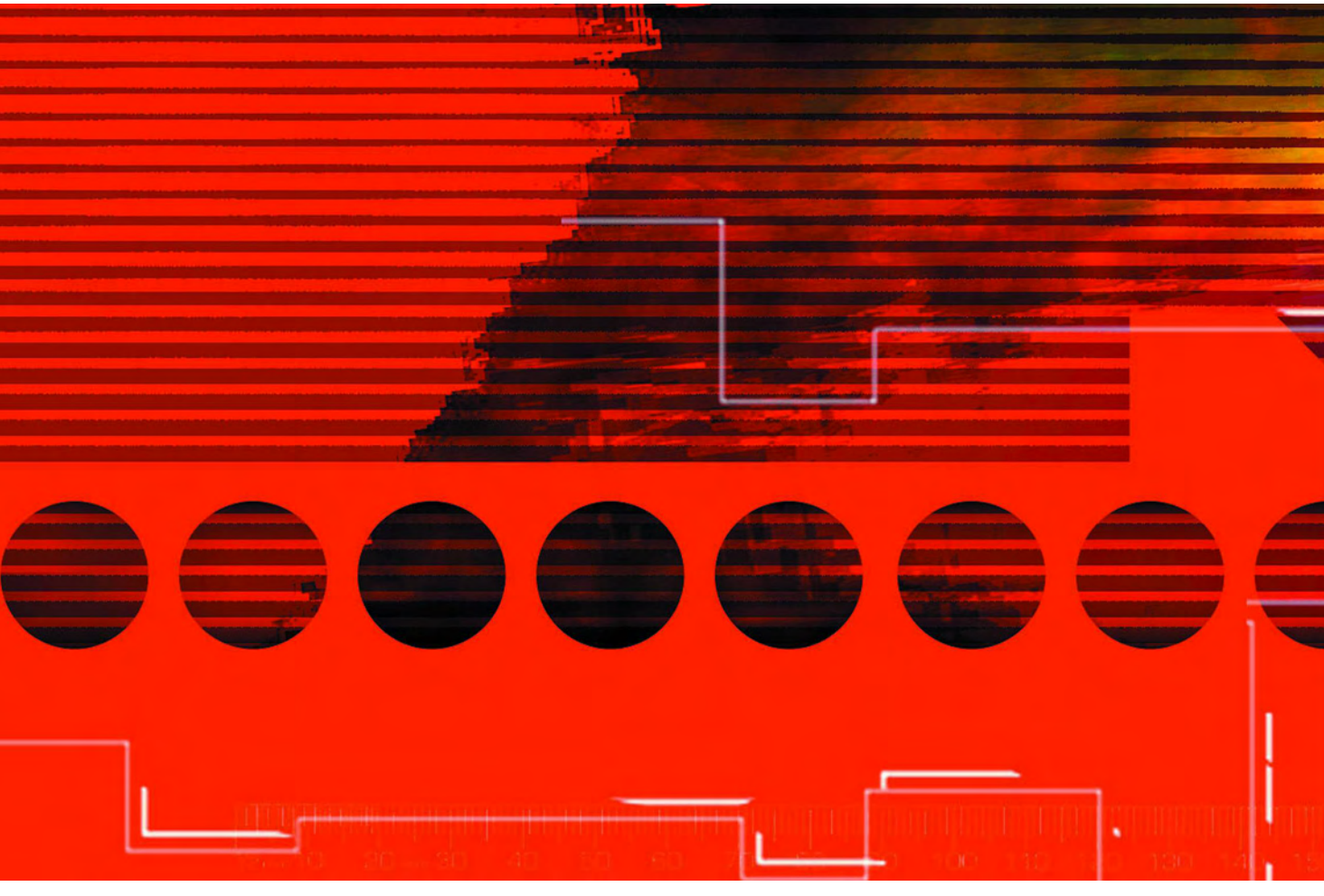
Keller: At the core of semantic integration is the problem of ontology mapping. An ontology map provides information on how to translate the objects, attributes, and relations from one ontology model into those of another. This is the essence of semantic integration; we need to understand how, and under what circumstances, can the data in one data source be combined with data in another. Ontology maps provide the basis for the data translation. When the ontologies for two sources are similar in their conceptual structure, mapping is easy. But when the underlying data models are conceptually disparate, things get complicated. Ontology mapping has been the focus in the semantic web community for a while now, but remains a challenging problem: what form should the maps take, how to automatically generate maps, how to use them efficiently, etc.

More broadly, I think the challenge for making semantic integration work in the marketplace is to make it quicker and easier to specify data semantics. Currently, specifying semantics using ontologies is a somewhat arcane and tedious process. If I create a dataset, then I have to see clearly that there will be significant benefits down the road if I expend the time and effort necessary to provide semantic metadata. Although we are starting to see some good tools on the market to make this process easier, the cost/benefit calculations are not yet sufficiently favorable to support widespread adoption of this approach.

Dr. Dobb's: Can you tell us about the SemanticIntegrator project?
Keller: SemanticIntegrator is a project focused on developing an architecture to support semantic integration of NASA data assets. NASA has accumulated many thousands of datasets generated as part of our work in science, engineering, aeronautics, and space exploration. If NASA could provide a means of connecting and combining disparate, but related datasets, then we would have a powerful capability that might enable scientists and engineers to gain additional insight based on the gestalt of the integrated data. We have used our architecture to demonstrate integration applications in both earth sciences and exploration. And we have demonstrated the utility of this approach for integrating NASA data with data from other federal agencies, such as USDA, EPA, NOAA, etc.

Our approach integrates information sources based on the use of ontologies plus explicit integration rules. For each data source, we develop an ontology that captures the semantics of the underlying data. In addition, we write a software wrapper that exposes the underlying data source as if it were a semantic web resource. Turning the data sources into semantic web resources enables them to be queried using a common, semantic query language such as W3C's SPARQL. The wrapper takes semantic queries as input and dispatches native data source queries (e.g., in traditional SQL) against the actual data sources. In addition to the data source ontologies, an integrating ontology is developed to capture the customer's view of relevant data and relationships across the various sources. To access integrated data, a client application queries the integrated ontology. Using a set of ontology translation rules, this query is mapped into a set of separate queries against the data source ontologies. The results are then translated back into the integrating ontology language and presented to the client application.

Return to Table of Contents



Natural Language Processing with Python

Natural Language Processing with Python
Steven Bird, Ewan Klein and Edward Loper
O'Reilly Media
$44.99

Reviewed by Mike Riley

Python is well known among programmers and system administrators alike to possess powerful libraries ranging from web frameworks and image processing to automated workflows and gaming. A lesser known yet extremely powerful Python library is the Natural Language Toolkit. *Natural Language Processing with Python* demonstrates how to leverage this toolkit to create sophisticated NLP applications.

Ever since I programmed my first interactive application in BASIC on a TRS-80 nearly 25 years ago, I dreamed of fluid, natural conversations with computers, ala various science fiction stories like Arthur C Clark's *2001: A Space Odyssey* and Phillip K Dick's *Do Androids Dream of Electric Sheep.* The computing world has evolved by leaps and bounds since then, though we have yet to attain that elusive vision of ubiquitous, natural conversational interaction with a computer. Text to speech engines are fast approaching the sound and inflection of a convincing human voice and voice recognition has also greatly improved (my Android phone is highly accurate with short phrases - not quite 100%, but far better than what voice recognition was like even a few years ago). Still, an intelligent back-end is required to hook all these technologies together in a cohesive, effortless user experience. While *Natural Language Processing with Python* didn't quite attain this lofty goal, it educated me on the nuances of NLP and the difficult computing problems that need to be resolved before this futuristic vision can become commonplace.

The book starts off with the terms and concepts behind NLP and introduces the free, open source Natural Language Toolkit (NLTK), followed by installing and downloading the NLTK demo book data collection and running some simple Python scripts to show off the NLTK's functions and lexical diversity. A fun exercise is running *nltk.chat.chatbots()*, which shows how NLP can interact with users in a not-quite-there Turing Test sort of way. The next 10 chapters delve into all things NLP, from accessing and processing large bodies of text (both text corpora and raw formats), a quick Python primer oriented toward NLP (complete with Mathplotlib and PyNum data visualization examples) in Chapter 4, using a part-of-speech tagger and automating such tagging via regular expressions, lookups and N-Gram tagging. Text and sequence classification and recognizing textual entailment (ex: predicting the true/false relationships of text within a statement) are covered in Chapter 6. Decision trees, information gain (a measure of "how much more organized the input values become when we divide them up using a given feature"), naive Bayes classifiers ("every feature gets a say in determining which label should be assigned to a given input value"), and other techniques: zero counts, smoothing, maximum entropy classifiers, linguistic pattern modeling, information extraction architecture from unstructured data, chunking, chinking and tag patterns, tree traversal, named entry recognition (NER), relation extraction and more. Chapter 8 covers sentence structure analysis (i.e., dealing with ubiquitous ambiguity), context-free, dependency and weighted grammars, with feature-based grammars discussed in Chapter 9. All this dense background comes together in Chapter 10 by applying an NLP interface to an underlying SQL-structured data source using propositional and first-order logic. Understanding the semantics of English sentences via the Principle of Compositionality, lambda-Calculus, quantified NP's, transitive verbs and discourse representation structures (DRS). The final chapter on managing linguistic data from various sources such as the web, word document files and spreadsheets is demonstrated in a TIMIT (a consortium of Texas Instruments and the Massachusetts Institute of Technology) Corpus, and concluding with an extended welcome to the Open Languages Archive Community (OLAC). The book closes with an Afterword on engaging the reader in the various computational challenges in state-of-the-art NLP systems, the NLTK roadmap and a bold invitation to "build new language technologies to better serve the needs of the information society, and ultimately as a pathway into deeper understanding of the vast riches of human language." Who could turn down such an offer?!

Each chapter concludes with a series of exercises ranging in difficulty; unfortunately, answers to the exercises are nowhere to be found, not even on the book's website. Some of the more public-facing examples of NLP in action are on popular web sites including ask.com and wolframalpha.com. While the authors fail to point readers to such commercial entities that have successfully incorporated the NLTK into their backend data processing applications, such websites no doubt employ the principles discussed in the book.

Natural Language Processing with Python delivers a solid education for any computing professional interested in the complexity and current state of the art in NLP systems. Python programmers will find the book especially Pythonic in the NLTK's implementation and use of NLP principles. While my dream of having an intelligent spoken word conversation with my computer may have to wait for another 25 years of computing evolution, this book helped me understand the complexities of the problem and ways to get closer to the solution.

Return to Table of Contents

Fighting Fragmentation with Mobile Virtualization

Will mobile virtualization prevent fragmentation?

by Steve Subar

Motorola and T-Mobile have announced the launch of a new and innovative Android-based smartphone, the Cliq. This attractive, feature-rich slider handset happens to build on a chipset and firmware that feature OKL4 inside. Definitely cool, but not the focus of this article.

The buzz about this new handset comes from Blur, the social media UI Motorola developed to run on top of Android. Blur aggregates messaging streams and content from Twitter, FaceBook, MySpace, Picasa and a dozen other portals and services. Like many new applications and services, Blur aspires to provide both an end-user experience AND a developer platform.

The dual nature of Blur has prompted cries of Foul and Fragmentation from a number of industry voices, especially from fans and followers of the underlying Android platform. At issue is that instead of augmenting Android, Blur could take on a life of its own, distracting developers and further fragmenting the already splintered smartphone software segment.

The potential for Android fragmentation has existed since Google launched the platform. The search giant and its partners in the Open Handset Alliance (OHA) chose the permissive Apache license to govern Android. Apache offers comfort to manufacturers (OEMs) but worries many that without other strong governance, those OEMs will fork Android to suit their own needs.

I won 't argue the merit of these fears, nor of OHA 's anti-fragmentation pledge. Instead, let 's look at the issue from the OEM point of view. OEMs live in a crowded, noisy, competitive marketplace. To stand out, handset OEMs need to build and market highly differentiated devices with clear, branded added value. However, some of the leading mobile platforms run counter to that need. On one hand, they provide a rich user experience. On the other, they lock OEMs into a narrow look and feel and branding — a phone built with these platforms is first an instance of that software, and second, a product of the OEM. This type of commoditization, akin to the desktop PC marketplace, is anathema to mobile handset design.

So, it is completely comprehensible why OEMs would seek to customize, re-skin and re-brand a platform like Android. However, the further an OEM takes Android from its off-the-shelf architecture and APIs, the more difficult it becomes for ISVs and developers to target that device with the platform SDK and standard software.

I would like to posit an alternative to the trend and temptation to fragment for differentiation: you guessed it — mobile virtualization. By deploying mobile virtualization, OEMs can retain a 100% platform-compliant environment in tandem with a custom-branded and differentiated one, boasting an interface that helps their devices stand out in the mobile crowd. OEMs can deploy a commercial off-the-shelf (COTS) mobile OS like Android, SymbianOS, or Windows Mobile in one virtual machine (VM), and a fully customized vendor-specific interface in another. The differentiating VM could host a complete instance of the customized COTS OS, or just lighter weight code that lets the device strut its stuff, communicating with the COTS OS VM via high-speed hypervisor IPCs.

Mobile virtualization brings the best of both worlds to OEMs, ISVs and developers — a COTS platform and APIs with an opportunity to differentiate mobile products and product families. As a balm for fragmentation, mobile virtualization offers mobile network operators and consumers more opportunities for new services and access to ever-larger software portfolios.

— *Steve Subar is CEO and President of OK Labs.*

Return to Table of Contents

Prefer Structured Lifetimes: Local, Nested, Bounded, Deterministic

What's good for the function and the object is also good for the thread, the task, and the lock

By Herb Sutter

There was a time when it was a novel idea that function calls should obey proper nesting, meaning that the lifetime of a called function should be a proper subset of the lifetime of the function that called it:

```
void f() {
  // …
  g();    // jump to function g here and then…
  // …return from function g and continue here!
  // …
}
```

"Eureka!" said Edsger Dijkstra. "Function *g*'s execution occurs entirely within that of function *f*. Boy, that sure seems easier to reason about than jumping in and out of random subroutines with unstructured *goto*s. I wonder what to call this idea. There seems to be inherent **structure** to it. Hmm, I bet I could build a deterministic and efficient model of 'stack local variables' around it too… and maybe I should write a letter…" (I paraphrase.) [1]

That novel idea begat the discipline of structured programming. This was a huge boon to programming in general, because structured code was naturally localized and bounded so that parts could be reasoned about in isolation, and entire programs became more understandable, predictable, and deterministic. It was also a huge boon to reusability and a direct enabler of reusable software libraries as we know them today, because structured code made it much easier to treat a call tree (here, *f* and *g* and any other functions they might in turn call) as a distinct unit — because now the call graph really could be relied upon to be a tree, not the previously usual plate of "*goto* spaghetti" that was difficult to isolate and disentangle from its original environment. The structuredness that let any call tree be designed, debugged, and delivered as a unit has worked so well, and made our code so much easier to write and understand, that we still apply it rigorously today: In every major language, we just expect that "of course" function calls on the same thread should still logically nest by default, and doing anything else is hardly imaginable.

That's great, but what does it have to do with concurrency?

A Tale of Three Kinds of Lifetimes

In addition to the function lifetimes we've just considered, Table 1 shows three more kinds of lifetimes — of objects, of threads or tasks, and of locks or other exclusive resource access — and for each one lists some structured examples, unstructured examples, and the costs of the unstructured mode.

For familiarity, let's start with object lifetimes (left column). I'll dwell on it a little, because the fundamental issues are the same as in the next two columns even though those more directly involve concurrency.

In the mainstream OO languages, a structured object lifetime begins with the constructor, and ends with the destructor (C++) or *dispose* method (C# and Java) being called before returning from the scope or function in which the object was created. The bounded, nested lifetime means that

	Object lifetimes	Thread/task lifetimes	Lock lifetimes
Structured Examples	Function local vars: • C++ stack scope • C# using blocks • Java dispose pattern By-value parameters By-value nested objects	Recursive/data decomposition: • Subrange per nested stack call • Partitioned sub-ranges, parallel calls Join-before-return, internal parallelism	Scoped locking: • C++ lock_guard • C# lock blocks • Java synchronized blocks Release-before-return
Unstructured Examples	Global objects Heap/lib allocation	Spawn "new thread()" Spawn process	Explicit "lock" call, with no automatic unlock or "finally unlock" call
Unstructured Costs and Drawbacks	Nondeterministic finalization timing Allocation overhead Tracking overhead to not use after delete: • GC • smart pointers Ownership cycles	Nondeterministic execution/join timing Allocation overhead Tracking overhead to not use task after EOT, join task before EOP Ownership/waiting cycles	Nondeterministic lock acq/rel ordering Allocation overhead Tracking overhead to avoid deadlock, priority inversion, ... Waiting/blocking cycles (deadlock)

Table 1: Structured vs. unstructured examples.

cleanup of a structured object is deterministic, which is great because there's no reason to hold onto a resource longer than we need it. The object's cleanup is also typically much faster, both in itself and in its performance impact on the rest of the system. [2] In all of the popular mainstream languages, programmers directly use structured function-local object lifetimes where possible for code clarity and performance:

- In some languages, we get to express the structured lifetime using a language feature, such as stack-based or by-value nested member objects in C++, and *using* blocks in C#.
- In other languages, we use a programming idiom or convention, such as the *try/finally* dispose pattern in Java, and explicit dispose-chaining (to have our object's *dispose* also call *dispose* on other objects exclusively owned by our object, the equivalent of by-value nested member objects) in both C# and Java.

Unstructured, non-local object lifetimes happen with global objects or dynamically allocated objects, which include objects your program may explicitly allocate on the heap and objects that a library you use may allocate on demand on your behalf. Even basic allocation costs more for unstructured, heap-based objects than for structured, stack-based ones. Objects with unstructured lifetimes also require more bookkeeping — either by you such as by using smart pointers, or by the system such as with garbage collection and finalization. Importantly, note that C# and Java GC-time finalization [3] is not the same as disposing, and you can only do a restricted set of things in a finalizer. For example, in object A's finalizer it's not generally safe to use any other finalizable object *B*, because *B* might already have been finalized and no longer be in a usable state. Lest we be tempted to sneer at finalizers, however, note also that C++'s shutdown rules for global/static objects, while somewhat more deterministic, are intricate bordering on arcane and require great care to use reliably. So having an unstructured lifetime really does have wide-ranging consequences to the robustness and determinism of your program, particularly when it's time to release resources or shut down the whole system.

Speaking of shutdown: Have you ever noticed that program shutdown is inherently a deeply mysterious time? Getting orderly shutdown right requires great care, and the major root cause is unstructured lifetimes: the need to carefully clean up objects whose lifetimes are not deterministically nested and that might depend on each other. For example, if we have an open *SQLConnection* object, on the one hand we must be sure to *Close()* or *Dispose()* it before the program exits; but on the other hand, we can't do that while any other part of the program might still need to use it. The system usually does the heavy lifting for us for a few well-known global facilities like console I/O, but we have to worry about this ourselves for everything else.

This isn't to say that unstructured lifetimes shouldn't be used; clearly, they're frequently necessary. But unstructured lifetimes shouldn't be the default, and should be replaced by structured lifetimes wherever possible.

Enter Concurrency

All of the issues and costs associated with unstructured object lifetimes apply in full force to concurrency. And not only do we see "similar" issues and costs arise in the case of unstructured concurrency, but often we see the very same ones.

Threads and tasks have unstructured lifetimes by default on most systems, and therefore by default non-local, non-nested, unbounded, and nondeterministic. That's the root cause of most of threads' major problems. [4] As with unstructured object lifetimes, to manage unstructured thread and task lifetimes we need to perform tracking to make sure we don't try to wait for or communicate with a task after that task has already ended (similar to use-after-delete or use-after-dispose issues with objects); and to join with a thread before the end of the program to avoid risking undefined behavior on nearly all platforms (similar to unstructured "eventual" finalization at shutdown time). Unstructured threads and tasks also incur extra over-heads [5], such as extra blocking when we attempt to join with multiple pieces of work; they are also more likely to have ownership cycles that have to be detected and cleaned up, and similarly for waiting cycles and even deadlock (e.g., when two threads wait for each other's messages).

Structured thread and task lifetimes are certainly possible. You just have to make it happen by applying a dose of discipline when they aren't structured by default: A function that spawns some asynchronous work has to be careful to also join with that work before the function itself returns, so that the lifetime of the spawned work is a subset of the invoking function's lifetime and there are no outstanding tasks, no pending futures, no lazily deferred side effects that the caller will assume are already complete. Otherwise, chaos can result, as we'll see when we analyze an example of this in the next section.

With locks, the stakes are higher still to keep lifetimes bounded — the shorter the better — and deterministic. Deadlocks happen often enough when lock lifetimes all are structured, and being

structured isn't enough by itself to avoid deadlock [6]; but toying with unstructured locking is just asking for a generous helping of latent deadlocks to be sprinkled throughout your application. Unstructured lock lifetimes also incur the other issues common to all unstructured lifetimes, including tracking overhead to manage the locks, and performance overhead from excessive waiting/blocking that can cause throttling and delay even when there isn't an outright deadlock.

Given the stakes, it's no surprise that every major OO language offers direct language support for bounded lock lifetimes where the lock is automatically released at the end of the scope or the function:

- C++0x has *std::lock_guard*, which is used in a way that just leverages C++'s bounded stack-based variable rules.
- C# has *lock* blocks, and can also leverage its *using* blocks with lock types that implement *IDisposable*.
- Java has *synchronized* blocks, and can leverage the *try/finally* dispose pattern for disposable lock objects.

These tools exist for a reason. Strongly prefer using these structured lifetime mechanisms for scoped locking whenever possible, instead of following the dark path of unstructured locking by explicit standalone calls to *Lock* functions without at least a *finally* to *Unlock* at the end of the scope.

Having all that in mind, we'll turn to an example that illustrates a simple but common mistake that has its root in unstructuredness.

An Example, With a Common Initial Mistake

Let's take a fresh look at the same Quicksort example we considered briefly last month. [7] Here is a simplified traditional sequential implementation of quicksort:

```
// Example 1: Sequential quicksort
void Quicksort( Iter first, Iter last ) {
  // If the range is small, another sort is usually faster.
  if( distance( first, last ) < limit ) {
    OtherSort( first, last );
    return;
  }
  // 1. Pick a pivot position somewhere in the middle.
  // Move all elements smaller than the pivot value to
  // the pivot's left, and all larger elements to its right.
  Iter pivot = Partition( first, last );
  // 2. Recurse to sort the subranges.
  Quicksort( first, pivot );
  Quicksort( pivot, last );
}
```

How would you parallelize this function? Here's one attempt, in Example 2 below. It gets several details right: For good performance, it correctly reverts to a synchronous sort for smaller ranges, just like a good synchronous implementation reverts to a non-quicksort for smaller ranges; and it keeps the last chunk of work to run synchronously on its own thread to avoid a needless extra context switch and preserve better locality.

```
// Example 2: Parallelized quicksort, take 1 (flawed)
void Quicksort( Iter first, Iter last ) {
  // If the range is small, synchronous sort is usually faster.
  if( distance( first, last ) < limit ) {
    OtherSort( first, last );
    return;
  }
  // 1. Pick a pivot position somewhere in the middle.
  // Move all elements smaller than the pivot value to
  // the pivot's left, and all larger elements to its right.
  Iter pivot = Partition( first, last );
  // 2. Recurse to sort the subranges. Run the first
  // asynchronously, while this thread continues with
  // the second.
  pool.run(  [=]{ Quicksort( first, pivot ); }  );
  Quicksort( pivot, last );
}
```

Note: *pool.run(x)* just means to create task *x* to run in parallel, such as on a thread pool; and *[=]{ f; }* in C++, or *()=>{ f(); }* in C#, is just a convenient lambda syntax for creating an object that can be executed (in Java, just write a runnable object by hand).

But now back to the main question: What's wrong with this code in Example 2? Think about it for a moment before reading on.

What's Wrong, and How To Make It Right

The code correctly executes the subrange sorts in parallel. Unfortunately, what it doesn't do is encapsulate and localize the concurrency, because this version of the code doesn't guarantee that the subrange sorts will be complete before it returns to the caller. Why not? Because the code failed to join with the spawned work. As illustrated in Figure 1, subtasks can still be running while the caller may already be executing beyond its call to *Quicksort*. The execution is unstructured; that is, unlike normal structured function calls, the execution of subtasks does not nest properly inside the execution of the parent task that launched them, but is unbounded and nondeterministic.

The failure to join actually causes two related but distinct problems:

- **Completion of side effects:** The caller expects that when the function returns the data in the range is sorted.
- **Synchronization:** The caller expects *Quicksort* won't still be accessing the data in the range, which would race with the caller's subsequent uses of that same data.

Remember, to the caller this appears to be a synchronous method. The caller expects all side effects to be completed, and has no obvious way to wait for them to complete if they're not.

Fortunately, the fix is easy: Just make sure that each invocation of *Quicksort* not only spawns the subrange sorting in parallel, but also waits for the spawned work to complete. To illustrate the idea,

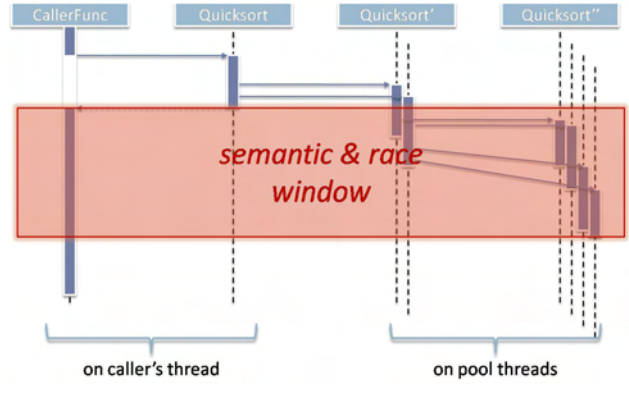


Figure 1: Example 2 execution is unstructured

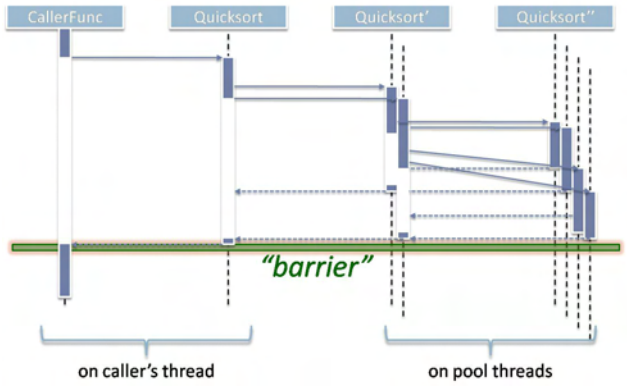


Figure 2: Example 3 execution is structured

the following corrected example will use a *future* type along the lines of futures available in Java and in C++0x, but any kind of task handle that can be waited for will do:

```
// Example 3: Parallelized quicksort, take 2 (fixed)
void Quicksort( Iter first, Iter last ) {
  // If the range is small, synchronous sort is usually faster.
  if( distance( first, last ) < limit ) {
    OtherSort( first, last );
    return;
  }
  // 1. Pick a pivot position somewhere in the middle.
  // Move all elements smaller than the pivot value to
  // the pivot's left, and all larger elements to its right.
  Iter pivot = Partition( first, last );
  // 2. Recurse to sort the subranges. Run the first
  // asynchronously, while this thread continues with
  // the second.
  future fut = pool.run(  [=]{ Quicksort( first, pivot ); }  );
  Quicksort( pivot, last );
  // 3. Join to ensure we don't return until the work is complete.
  fut.wait();
}
```

As illustrated in Figure 2, just making parent tasks wait for their subtasks makes the execution nicely structured, bounded, and deterministic. We've implemented a barrier where everyone including the caller waits before proceeding, and thus returned the desired amount of synchronicity:

- **Completion of side effects:** We guarantee that when the function returns, it has done its full job and the range is sorted.
- **Synchronization:** After the function returns, some part of its work won't still be accessing the data, and so won't invisibly race with the caller's subsequent uses of the data.

Summary

Table 2 illustrates the differences between structured and unstructured lifetimes in the form of a little artsy photo sequence. In the following descriptions, notice how consistently we use synonyms for our four keywords: local, nested, bounded, and deterministic.

- **Structured:** As illustrated on the left side of Table 2, structured lifetimes are like Russian dolls. They nest cleanly, so that each doll fits entirely inside the next larger one. As the program is assembled and executed, each subgroup can be manipulated as a unit. When fully assembled, every individual part is nicely encapsulated and in a well-known place, easy to understand and reason about.

Table 2: Structured vs. unstructured, a graphic treatise

- **Unstructured (initial intent):** When we first buy into unstructured lifetimes, we think we're buying into the middle picture of raw pasta. We start out with a manageable handful of stiff little independent and parallel lifetimes that don't twist or cross very much. Life is good, at least in the initial PowerPoint design…
- **Unstructured (in practice):** Unfortunately, as the final picture shows, we end up with something quite different. Software under maintenance behaves a lot like pasta: Once it's cooked and combined with other ingredients, the unstructured threads don't stay easy to grasp and easy to separate, but rather get intertwined and easier to break. What we actually wind up with ends up being quite slippery — in the worst case, "spaghetti code."

Just because some work is done concurrently under the covers, behind a synchronous API call, doesn't mean there isn't any sequential synchronization with the caller. When a synchronous function returns to the caller, even if it did internal work in parallel it must guarantee that as much work as is needed is fully complete, including all side effects and uses of data.

Where possible, prefer structured lifetimes: ones that are local, nested, bounded, and deterministic. This is true no matter what kind of lifetime we're considering, including object lifetimes, thread or task lifetimes, lock lifetimes, or any other kind. Prefer scoped locking, using RAII lock-owning objects (C++, C# via *using*) or scoped language features (C# *lock*, Java *synchronized*). Prefer scoped tasks, wherever possible, particularly for divide-and-conquer and similar strategies where structuredness is natural. Unstructured lifetimes can be perfectly appropriate, of course, but we should be sure we need them because they always incur at least some cost in each of code complexity, code clarity and maintainability, and run-time performance. Where possible, avoid slippery spaghetti code, which becomes all the worse a nightmare to build and maintain when the lifetime issues are amplified by concurrency.

Acknowledgment

Thanks to Scott Meyers for his valuable feedback on drafts of this article.

Notes

[1] E. Dijkstra. "Go To Statement Considered Harmful" (*Communications of the ACM*, 11(3), March 1968). Available online at search engines everywhere.

[2] Here is one example of why deterministic destruction/dispose matters for performance: In 2003, the microsoft.com website was

using .NET code and suffering performance problems. The team's analysis found that too many mid-life objects were leaking into Generation 2 and this caused frequent full garbage collection cycles, which are expensive. In fact, the system was spending 70% of its execution time in garbage collection. The CLR performance team's suggestion? Change the code to *dispose* as many objects as possible before making server-to-server calls. The result? The system spent approximately 1% of its time in garbage collection after the fix.

[3] C# mistakenly called the finalizer the "destructor," which it is not, and this naming error has caused no end of confusion.

[4] E. Lee. "The Problem with Threads" (*University of California at Berkeley, Electrical Engineering and Computer Sciences Technical Report*, January 2006). Available online at http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf.

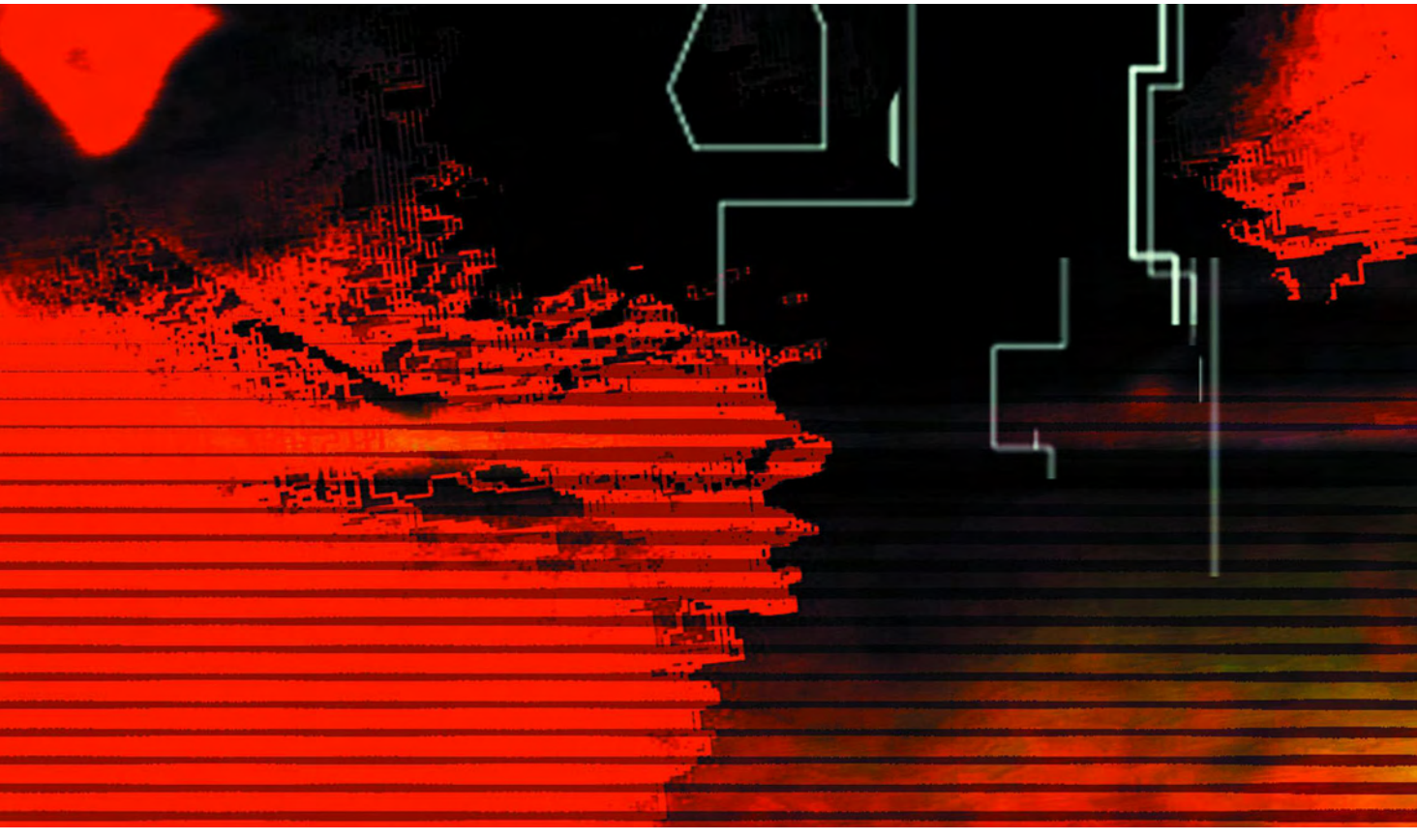
[5] The new parallel task runtimes (e.g., Microsoft's Parallel Patterns Library and Task Parallel Library, Intel's Threading Building Blocks) are heavily optimized for structured task lifetimes, and their interfaces encourage structured tasks with implicit joins by default. The resulting performance and determinism benefits are some of the biggest improvements they offer over older abstractions like thread pools and explicit threads. For example, knowing tasks are structured means that all their associated state can be allocated directly on the stack without any heap allocation. That isn't just a nice performance checkmark, but it's an important piece of a key optimization achievement, namely driving down the cost of unrealized concurrency — the overhead of expressing work as a task (instead of a synchronous function call), in the case when the task is not actually executed on a different thread — to almost nothing (meaning on the same order as the overhead of an ordinary function call).

[6] The key to avoiding deadlock is to acquire locks in a deterministic and globally recognized order. Although releasing locks in reverse order is not as important for avoiding deadlock, it is usually natural. The key thing for keeping lock lifetimes structured is to release any locks the function took at least by the end of the function, which makes code self-contained and easier to reason about. Even when using a technique like hand-over-hand locking where lock release is not in reverse order of lock acquisition, locking should still be scoped so that all locks a function took are released by the time the function returns. See also H. Sutter. "Use Lock Hierarchies to Avoid Deadlock" (*Dr. Dobb's Journal*, 33(1), January 2008). Available online at http://www.ddj.com/hpc-high-performance-computing/204801163.

[7] H. Sutter. "Effective Concurrency: Avoid Exposing Concurrency – Hide It Inside Synchronous Methods" (*Dr. Dobb's Digest*, October 2009). Available online at http://www.ddj.com/go-parallel/article/showArticle.jhtml?articleID=220600388 .

—Herb Sutter is a bestselling author and consultant on software development topics, and a software architect at Microsoft. He can be contacted at www.gotw.ca.

Return to Table of Contents



Two Pomodoros at Foo Bar

"Evening, Larry. A glass of the house Pinot Gris?"

"It is a comfort," journalist Larry Wilde said as he settled onto his customary stool, "to end a long day in a saloon where the bartender knows your preferences."

I poured. Larry watched. The tomato ticked.

"Rough day?" I asked, "Or just a long one?"

"Productive but inconclusive," he said between sips. "I'm researching computer languages for a piece I'm doing on spec for a new online magazine about Silicon Valley lifestyles. Silicon Valley lifestyles. I must say I have my doubts about a publication premised on an oxymoron, but there's not a lot of work out there for a beat journalist."

I tsked sympathetically. I could think of three possible meanings for "beat journalist," two of which fit Larry these days. I enjoyed serving drinks at Foo Bar for the tips I got from working journalists, but fewer and fewer journalists were actually working. "Computer languages, eh?"

"Yes, the question the article asks is, when programming languages grow old, what happens to the programmers using them?"

"Do programming languages really grow old?"

"Oh, yes. Think of Pascal. Once it was all hip and structured and stylish, like Danish Modern furniture. Now it's dated and quaint and inconvenient — "

"Like Danish Modern furniture?"

"Or Ada, which went from being not your father's Oldsmobile to being your grandfather's Edsel."

"Where are you getting these characterizations of languages, anyway?"

"From my Twitter followers. What a marvelous research instrument Twitter is, Michael! You just throw out a question and in minutes you have dozens of answers."

"Yes, but — "

"Or consider Forth. In its youth, Forth was punk, but now it's bluegrass."

"But I don't think these characterizations make any sense."

"Fine, Forth was Ska, but now it's techno. That was another tweet I got. The point is, languages age; they cease to fill the role they once filled, and search for a new role, usually one that involves taking a nap in mid-afternoon. By the way, are you aware that there is a ticking tomato on the bar?"

"It's a Pomodoro," I explained.

"That's simply the Italian for tomato, I believe." He frowned at me but smiled as I refilled his glass.

"The Pomodoro Technique is an agile methodology for a single individual," I explained. "It was invented by Francesco Cirillo and requires the use of a five-dollar kitchen timer. You work for twenty-five minutes and then break for three to five minutes. Each such time period is also called a Pomodoro. If you're being really true to the plan, you use a timer shaped like a tomato."

"So that's your pomodoro. I see. And what is it ticking for?"

"I'm using the Pomodoro Technique to organize my work."

He snorted. "What work? Excuse me, Michael, but all you do here is schmooze with the customers."

Ding!

"Your tomato just dinged."

"Sorry," I said, "I have to stop schmoozing for the next three to five minutes." I set the Pomodoro for five minutes, put my feet up on a nearby stool, my back to the bar, and opened the Merc.

"Ah, I get it. Schmoozing is your work, and you've just done your 25-minute spell of it, so now you get a break. Clever."

"La la la. Can't hear you."

He kept talking and I continued to ignore him. I also ignored the yuppie couple who walked in and sat at the chair next to Larry.

"You'll just have to wait," Larry told them. "He's on Pomodoro break."

A little later the Pomodoro dinged, I reset it for twenty-five minutes, and I asked what I could do for them. After filling their order, I picked up my conversation with Larry, first recapping, in proper Pomodoro Technique, what had been accomplished in the preceding Pomodoro.

"You were telling me about your magazine article, but not making any sense. Continue."

"I was too making sense. Anyway, the point of the article is, what happens to the programmer in such cases?

If you are a long-time Java programmer, you were once cutting-edge and cool. What are you now if you're still using Java? Have you actually changed and become a dinosaur, or are you still the risk-taker you once were, but strapped to a moribund language?"

"Well, people do change."

"Yes, but does the language you use dictate that you change? When it gets old, does it drag you along with it?"

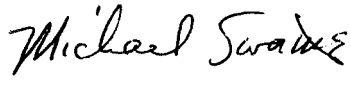
I waited on a couple other customers and pickeed up the conversation again.

"I'm still skeptical about this article of yours, Larry. What other languages are you looking at?"

"I'm asking if Haskell programmers turn into absent-minded professors, or Ruby programmers into grumpy old men. Do you have a programming language that you're particularly interested in these days?"

"Yes," I said. "Subleq. And I think it'll serve me well when I get so senile that I can only remember one instruction."

Ding!



By Michael Swaine

Return to Table of Contents