

# MEN ARE FROM MARS, WOMEN ARE FROM VENUS, WEB SERVICES ARE FROM BETELGEUSE

By Denise P. Kalm & Annie Shum

**“The only way we will really know the condition of Betelgeuse is to wait a few million years.”**

## **Abstract**

Having trouble making robust legacy applications talk to hot new Web applications? Trying to manage business transactions with your partners when everyone is running their proprietary software on different platforms? Simple, flexible interoperability is the “holy grail” and happens to be analogous to the challenges we all face when we try to communicate with the opposite sex. Loosely based on the Mars-Venus theme, we will explain the basics, benefits and challenges of creating or migrating to a Web services standard-based Service-Oriented Architecture (SOA). We will extend that theme to communication with people from other cultures/languages as an analogy to cross-enterprises B2B Web Services.

## **The Problem**

Despite sharing a species (homo sapiens), according to Deborah Tannen, men and women could not be more different when it comes to communication and how they view their place in the world. Even early on, young girls tend to view themselves as an individual in a network – all entities in the network are different, yet equal and all are interdependent. One-to-one communication is a vehicle for achieving connection and degrees of intimacy. On the other hand, young boys immediately recognize themselves as members of a hierarchy, and as children, lower members of the tree. Conversation is designed primarily to establish position in the hierarchy – who has the upper hand? Relationships are viewed as asymmetrical; even close friends need to view themselves as superior to each other to some extent. And importantly, independence is critical. “Women are also concerned with achieving status and avoiding failure, but these are not the goals they are focused on all the time, and they tend to pursue them in the guise of connection,” notes Deborah Tannen. (Fig. 1)

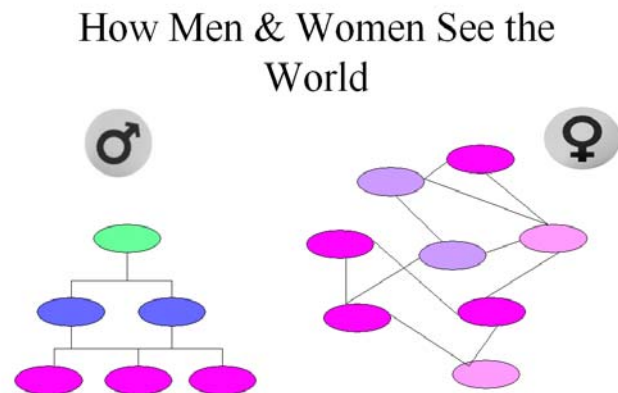


Fig. 1 - With this difference in worldview, few “APIs” exist to facilitate communication

With such different styles, goals and views, it is not surprising that universally successful communication rarely occurs. Moreover, men and women usually have to work very hard to establish “learn as you go” ways to connect, and between each couple, and often, for each situation, these methods are different, complex and error-prone.

She says, “I hate my job.” (I want sympathy and support)

He says, "Then quit." (I see a problem needing to be fixed)

She is unhappy and he can't figure out why his "fix" didn't work.

Sound familiar? Connecting disparate applications, operating systems, databases and any proprietary computer "thing" to another computer "thing" is a challenge with many plausible solutions, but almost none that work all the time, for all players. And the results may not be what you were expecting. If something changes (and it will), so do these communication rules. A common "code" or translator must be capable of making these changes transparent to communicating partners. No matter what words/APIs/transaction codes are used, the message is clear

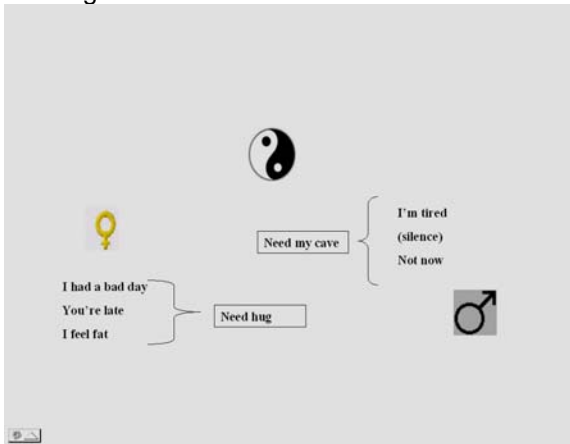


Fig. 2 Code translator

Even with counseling, many couples fall into very narrow, inflexible patterns for relating and when something changes, these highly customized one-to-one methodologies cease to work. When a relationship ends, it is a virtual guarantee that the methods will not work with the new partner, even though the attempt to graft an old solution onto a new problem is inevitable. You have to work out a new method for communicating and this takes time. Change is an inevitable part of life; you have to be agile and able to cope.

There is no difference in the information age. Ideally, businesses want to quickly and readily connect applications within their infrastructure, as well as to partnering companies. In reality, each application, each operating system has its own rule set which

can vary with time, use of the application or other conditions. Between similar applications (such as CICS to CICS), communication is easy, as it is between two men, or two women. But as in our personal lives, communicating only with those most like us is limiting. A business cannot mandate the systems chosen by a partner, and the "luck" of finding partners with like systems is increasingly unlikely. Even within a company, most applications were developed independently – the need to interact is new and was unlikely to have been considered in the original design.

Web Services is all about boundary-less communication with customers as well as business partners in a rapidly changing business world – how can two or more computers exchange information to accomplish a task in a standardized way? Carly Fiorina, chairman and CEO of HP notes, "IT managers are not interested in the fastest box, killer application or the next big technology, but rather on using what they have and making it all work in an environment prone to constant change. Companies that win are those that have an ability to manage change."

### Solutions From the Past

The requirement to communicate between applications is not a new one, but until now, all available solutions were by and large too complex, too vendor-specific, too expensive or all of the above. In 1975, the transportation industry developed EDI (Electronic Data Interconnect), but it was very complex and required private networks. In 1981, IDM (Intelligent Database Machines) founders separated the database from the applications using it – the beginning of client-server. By '83, RPC (remote procedure call) was developed so one computer could request another to perform work. A year later, NFS (network file system) extended this capability to permit file sharing. DCOM (Distributed Component Object Modules) and CORBA/IIOP (Common Object Request Broker Architecture/Internet Inter-Orb Protocol), along with MQ Series (Message Queuing), ORB (Object Request Broker) and others vied for a role as "the solution," but all fell short to achieve boundary-less interoperability. Most were tightly coupled and thus, less flexible. This lack of flexibility often resulted in these options being adopted only by industries in certain niches, but no solution really made

interoperability easy. No industry standard really took off until very recently. (Fig. 3)

RPC Architecture	Payload Parameter Value Format	Endpoint Naming	Wire Protocol	Interfaces
CORBA	Common Data Representation (CDR)	Interoperable Object Reference (IOR)	IIOP (Binary)	Interface Definition Language (IDL)
DCOM	Network Data Representation (NDR)	OBJREF	DCOM (Binary)	Inherited from COM
RMI	Serializable Java Objects	URL	Java Remote Method Protocol (JRMP)	Java Interfaces
Web Services	XML	URL	SOAP (Text-based)	WSDL

Fig. 3 – Distributed Computing Architectures compared

CERN (European Organization for Nuclear Research) created a subset of GML (generalized markup language) called HTML with a goal of trying to simplify the EDI mess (1991). The XML (extensible markup language) was designed – a more approachable GML. In 1998, Microsoft put XML-tagged RPCs into documents, using HTTP; the beginnings of SOAP. IBM joined up in '00, and Microsoft and IBM (along with other vendors) defined WSDL and UDDI; the Websphere Interoperability Organization was formed in '02 to define standards for using standard Web protocols already understood by most organizations. (Fig. 5)

Using traditional integration techniques, a typical application integration process looks a lot like a divorce mediation session. Two parties, speaking through their lawyers – the end result is a thick sheaf of paper, a raft of legal bills and two dissatisfied parties. No one gets exactly what he or she wants for what they are willing to pay. Often, one party (application) ends up making all the compromises and adjustments to “play” with the other. Unbalanced resolutions don’t work well for long, and generally, the requirement to adapt and recode continues as long as the inter-connection is required. Each party has its own API (lawyer) and their own view of the situation (data format). (Fig. 4)

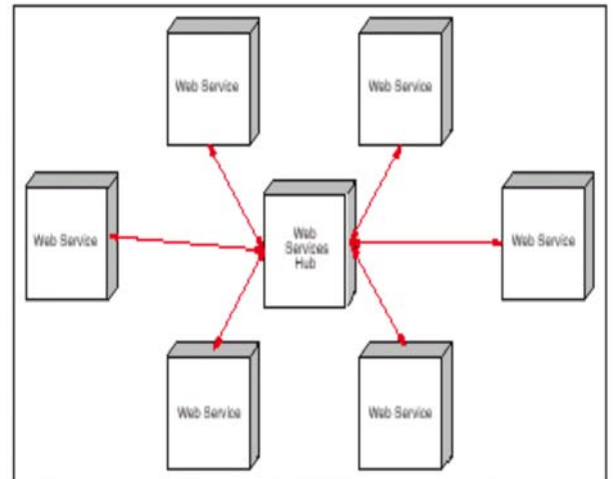


Fig. 5 - Six applications, one API, one data format

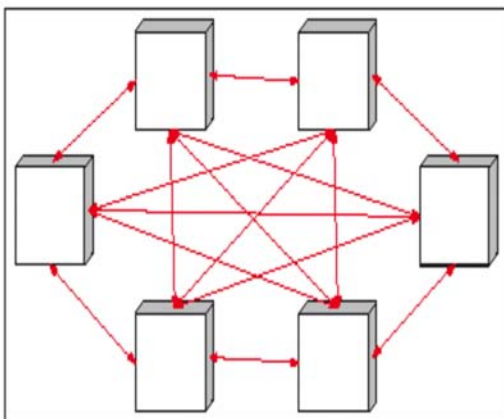


Fig. 4 - Six applications, six APIs and six data formats

Had enough acronyms yet? The key ones needed to understand Web Services and SOA will be explained as we go along.

### Web Services and SOA – The Future

On Betelgeuse, people saw all the mistakes made by the Martians and Venusians and decided that they would take a different approach. (Clearly defining and understanding the problem is key to solving it). Rather than speaking Martian to Venusians or vice versa, they taught all their children Esperanto, so communication between the sexes was as easy as it was among them. In addition, they defined a barter community, where people

from different star systems and galaxies could easily ask for or offer a service, with the Betelgeusians brokering the exchange. A very specific format was defined and everyone agreed to use it, because standardizing opened up the world to even the smallest of small businesses. Everyone used Esperanto in these exchanges, so communications were eased and mistakes were less frequent.

The data processing world must have heard some whisper of the Betelgeuse plan, because Web Services is now the “next, new disruptive technology.” Web Services is looking like the RCA Jack of the Web world; the fabric that allows companies to weave “best of breed” systems together into a network of sharable services. SOA, Service Oriented Architecture, represents a major paradigm shift in how business connects with technology. SOA is a methodology /architecture for linking resources on demand, where resources are made available as services to any participant in the network. A key factor in this is a very clear delineation of three key roles: Service Provider, Service Consumer and Service Broker. Moreover, legacy applications can be wrapped up in a service interface, so SOA is not just applicable to newly designed services. Encapsulation is key to this process – how a service is actually implemented is invisible to service consumers. Using a service should be like driving a car.

“To start a car, you don’t need to know how an internal combustion engine works or even how the starter motor works. You only need to know how to use the interface that the car supplies to start it: Turn the key.” Anne Thomas Manes

By clearly dividing the functions, and engaging standards-based interfaces for these roles, anyone agreeing to the “rules of engagement” can play. (Fig. 6,7)

The 3 basic conceptual roles & operations of SOA: Service Oriented Architecture

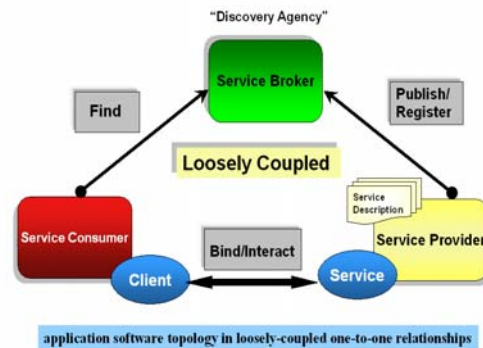


Fig 6 – The 3 Major Roles Interacting in a SOA architecture

**Web Services = Repository + Client + Provider**  
**Web services provide a way for clients to discover and connect to Web-based software services. There are three major aspects to Web services:**

- **A service provider** provides an interface for software that can carry out a specified set of tasks.
- **A service requester** (the client) discovers and invokes a software service to provide a business solution. The requester will commonly invoke a remote procedure call on the service provider, passing parameter data to the provider and receiving a result in reply.
- **A broker** (the repository) manages and publishes the service. Service providers publish their services with the broker. Clients then request access to those services by using the bindings defined by the service provider.

Fig. 7 – Web Services based SOA Roles and Operations

And what are these rules? Web Services based SOA is both a process and a set of protocols designed to connect disparate applications. Loosely defined, Web Services are pieces of structured XML messages that use interfaces defined by the international standards community. In the Web Services model, the provider develops a service, describes it using WSDL (Web Services Description Language) and then publishes it in UDDI (Universal Description, Discovery and Integration – “yellow pages”). The broker is the UDDI. (Note: most current implementations of Web Services do not use

UDDI; it is not needed for internal use or for trusted partners.) Each entry is an XML file, with white pages describing the company offering the service, green pages to describe the service itself and yellow pages, which describe the categories. A Web Service can also be registered with a public or private registry, such as a database, a directory service, or an XML file. Once a service is registered, components that want to call the service may use any type of registry to locate the service and then call the service.

The services themselves are defined in a document called a Type Model or Tmodel. Service consumers can readily search through any of these indexes to find what they need. WSDL is an XML-based metadata description of a service – what does it do? Finally, you invoke Web Services (expose the service) using SOAP (simple object access protocol), which is an XML-based messaging protocol. SOAP builds on XML and common transfer protocols (primarily HTTP, but it also supports other Internet protocols including ftp and SMTP as well as proprietary protocols such as MQ) to facilitate communication. And since XML and the protocols ARE common, Web Services is easier to implement, platform-agnostic and universal. Services and data can be shared without either end needing to know the proprietary internals of the other application. As with the Betelgeusians, voluntary compliance between business partners is assured because of the potential for mutual gain. Web Services is a part of the application, so there is no need for proprietary adaptors, which leads to lower cost of application integration.

Another option you have is whether to have synchronous or asynchronous messaging; the latter is SOAP with attachments, the former is SOAP RPC. Synchronous messaging is almost always a tight point-to-point integration, just like a marriage, where more of the “contract” can be defined. Friends and acquaintances have a more loosely coupled arrangement. (Fig.8)

	Tightly-Coupled	Loosely-Coupled
<b>Interaction</b>	Synchronous	Asynchronous
<b>Messaging Style</b>	RPC	Document
<b>Message Paths</b>	Hard Coded	Routed
<b>Technology Mix</b>	Homogenous	Heterogenous
<b>Data Types</b>	Dependent	Independent
<b>Syntactic Definition</b>	By Convention	Published Schema
<b>Bindings</b>	Fixed and Early	Delayed
<b>Semantic Adaption</b>	By Re-Coding	Via Transformation
<b>Software Objective</b>	Re-Use, Efficiency	Broad Applicability
<b>Consequences</b>	Anticipated	Unintended

Doug Kaye, Loosely-Coupled -- The Missing Pieces of Web Services

Fig. 8 – Coupling Options

Remember Fig. 4? Applying the core Web Services standards for a SOA implementation, SOAP is the messaging API; XML is the common data format, and HTTP is the transport method. As an example, Expedia.com originally had a one-to-one specialized API to each individual partner, but replaced it with Web Services.

For mainframers who wonder how they play in this, IBM has made it much easier for CICS applications to make the switch. CICS TS (Transaction Server) now has support for adaptors, which allow remote applications to invoke any CICS application as a Web Services. Some adaptors support both terminal-oriented (or visual) transactions as well as non-visual, COMMAREA transactions. Adaptors run on the mainframe (better performance and higher reliability) and can bypass screen-scraping altogether. In addition, with this method, you can also get status and error codes, making systems management much easier. Even legacy IMS applications can be converted using gateways. So don't scrap or rewrite your applications – upgrade in place instead. Check out Real Life Success Stories for examples of “upgrading in place” triumphs.

## The Real-life Success Stories

Most successful implementations started small and often involved only integration of in-house applications, a more controlled environment. As with personal relations, learning to understand and communicate with the opposite sex is a necessary prelude to communicating cross-culture and cross-language.

And what could be a better internal use of Web Services than to replace a batch order processing system with XML-based Web Services processing using WebSphere. Cole National (parent of the Things Remembered stores) now lets partners sell their products online on any operating system and send the orders via SOAP to Cole in near real time. Messages are built on WSDL or use a proxy class, which can validate the input parameters and construct a cXML request. To help them continue the migration (where appropriate), a key requirement was to build only reusable services, to keep costs down and to change the organizational model. (Fig. 9)

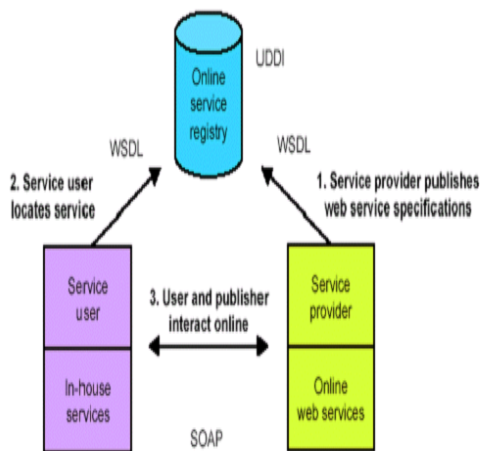


Fig. 9 – A Web Services example

Instant car loan approval? Why not? Ford, GM and DaimlerChrysler use Web Services to implement a dealer portal – no proprietary OS (operating systems) or hardware required. Furthermore, cost came in at 1/10 the cost of any proprietary system, with

none of the restrictions of such a system. Dollar Rent A Car wanted to have an alternative to paying Sabre or Apollo for booking reservations. They looked at the most common traditional EDI models, but decided on Web Services in '00. Using the AgentWare Travel Console to display their inventory, corporate travelers and travel agencies now have access to car reservations through SOAP/XML Web Services.

Merrill Lynch had 23,000 CICS programs running on its mainframes and needed to integrate those programs with other platforms. Creating their own tool set (X4ML), they were able to automatically produce WSDL files by analyzing compiler output. X4ML is accessed through HTTP synchronously or thru MQ Series either sync or async. To boost performance, Merrill also wrote its own XML parser running in CICS. This upped throughput from 19 tps to 239 tps! As it turned out, their homegrown solutions outperformed commercial products.

Let's look at a potentially intrusive "big brother" style, next generation application. Norwich Union (working with IBM and Orange UK) is creating a "Pay as you Drive" insurance program. Data is collected from a telematic device in your car and the premium is based on when, where and how often the car is used. Five thousand volunteers are involved in this pilot; it is a test of whether customers want the flexibility and choice it offers in determining how much their insurance should cost.

There are a variety of diverse examples, but why was it so important to these companies to take the risk? Business agility and responsiveness is key to survival – the market is changing so fast that building old-style applications over a two-year process is a recipe for disaster. Cost is another factor – reusable modules and a model for rapidly bringing in new services reduces the cost. Merrill Lynch's original EAI project was budgeted for \$800K while the Web Services implementation came under \$30K.

By increasing collaboration between customers, partners and suppliers, the potential for benefit for all increases, while giving customers the choice they always dreamed about.

Once you become convinced that SOA and Web Services is the way to go, where do you begin? Few industry analysts suggest a revolutionary, tidal wave approach. In fact, most suggest you pick a few target

applications to begin. Easy ones to start with are ones where the task/service is frequently invoked (worth doing some work on), where it is dynamic (data changes all the time) and where no connection currently exists. If you have a working connection, start with one you need, but don't have.

Steps that have been identified include:

- Assess needs
- Obtain expertise
- Identify a target project
- Test security by roping project off from the outside world
- Involve existing partners
- Focus on the long-term goal – keep code reusable, think of service packets, not lines of business

Other suggestions that appear to work for many are to make sure that what you build is platform-independent and based on a service model, not owned by the line of business. Keep communications open between the business and technical world and talk constantly. Encapsulate existing functionality in Web Services interface that exposes legacy functions as atomic web services. Then, compose atomic services into coarse-grained business services. For purchased software, go with best of breed – don't get locked in.

As Greg Byork (Systinet) says, "I think the biggest mistake that we could conceive of and it's probably one that gets made fairly consistently with new technology is when you've got a hammer, it all looks like a nail. Bite the stuff off in chunks, get definable projects where you believe there could be a rapid return on investment, implement a project and then measure the results."

### **The Challenges**

There are many challenges facing those trying to adopt SOA and Web Services. Not the least of which is that the culture of programming has always been opposed to the culture of reuse. Object programming might have been more successful if it didn't look like a way of eliminating jobs and stifling programming creativity. Even if the culture hurdle can be overcome, there is still the problem of defining a service. If you must code services, you need to know what a service is, and the standards have not yet been defined. And then, there might be

composite services, such as the set of services that would be involved in ordering a book from Amazon.com. One service might be just the credit card verification. One service might be the login and identification of a known user from cookies. But until "services" can be defined, most businesses will still be on the first steps of a real SOA implementation.

Another issue is that in many shops, application implementation reflected more the organizational structure than mirroring logical business function. Every department has their own date routines, as an example. If these kinds of fiefdom issues can be resolved, internal services can be standardized, resulting in immediate savings. Motorola saved more than \$100,000 simply by wrapping credit card validation code into a service; so, another business unit could use the same functionality. Again, this goes to defining a service, so it can be readily identified as reusable. There has also been little progress in bringing IT and the lines of business together. As Steve Ellis, Exec VP, Wells Fargo noted, "You need the business people to 'get' IT and the IT people to 'get' business." Failure to do so will result in redundant code and inefficiencies due to a single line of business-centricity, along with the tendency of techno-geeks to embrace the latest "cool" technology, regardless of its applicability.

In most cases, Web Services implementation will occur as a migration, not a revolution. But this requires absolutely a central commitment to this direction, or again, LOB-owned (line of business) solutions will result. LOBs will need to cede control, as some of the code ceases to be "owned" by them, and management is more at the corporate level. This could represent a return to the glass-house, mainframe style of systems management, which would probably reduce maintenance and management costs. Distribution of applications to business-owned UNIX servers often resulted in fragmented systems management, lack of standards and often, lack of change control. But wresting control back is by no means a trivial concern.

Performance will be a non-trivial issue to most shops, still struggling to ensure Web site performance to a varied and distributed group of end-users. First, there is the network issue – lack of good QOS (quality of service) tuning methods continue to make complex, cross-network, cross-business transactions problematic. In addition, SOAP is another

layer on top of the infrastructure and the XML text-based messages can be verbose and thus incur higher overhead and bandwidth. Layer mapping is required at all layers of the infrastructure; continued development will likely produce faster translation, but it is still a challenging issue. Processing text (XML) messages is slower than processing binary messages, but there are ways to parse XML faster, and more tools will become available as the demand increases.

Security is one of the biggest concerns as company-owned code is exposed to the network as shared services. SAML (security assertion markup language) may help to solve the problem, but verifying users and protecting corporate assets are going to be challenges for a while. And once you feel secure, how do you meter use and manage any chargeback, to recover costs of running an application? Old transaction/job-based chargeback mechanisms won't work anymore, and there will be a need to deal with inter-company chargeback. Paying for systems resources (a challenge that was never really resolved in the DS world) will be an issue that needs a new solution.

Transactional integrity and transaction rollback issues need to be addressed. In addition, outside your corporate firewalls, how do you reliably handle asynchronous messages?

Converting classic mainframe applications into Web Services servers (the 3270 green screen challenge) is an issue many vendors are beginning to confront. Even if the green screens are gone, the applications were designed to feed data to them. If you scrape the 3270 screen and then put a web front end on it (as many companies have done), you end up with more data volume than before, which could cause performance problems ("internet loads"). Browsers can show more than a single screen, but that's how the application was written.

### **Selected Bibliography**

- Tannen, Deborah "You Just Don't Understand – Women and Men in Conversation," William Morrow, 1990
- Gray, John "Men Are From Mars – Women Are From Venus: A Practical Guide for Improving Communication and Getting What You Want in Relationships," Harper-Collins, 1993
- Bloomberg, Jason "Web Services: A New Paradigm for Distributed Computing," The Rational Edge, September, 2001

Finally, holistic management tools need to be available that can view services and integrate systems and network performance. (Fig. 10)

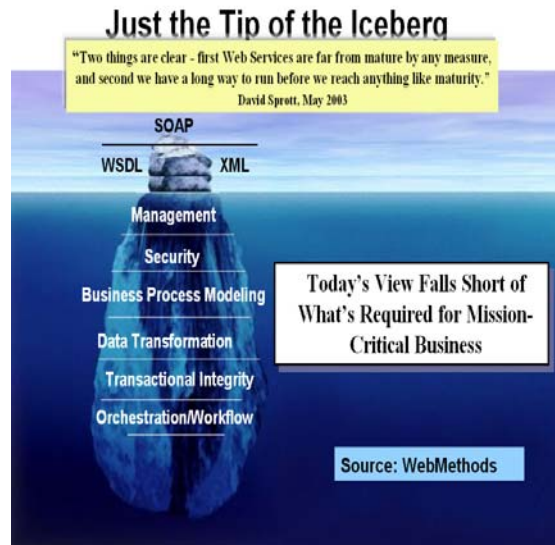


Fig. 10 – Not Ready for Prime Time?

### **Summary**

Unfortunately, the communication between Martians and Venusians is likely to remain problematic for the foreseeable future. But the data processing world won't have to be limited by that. Study the Betelgeusian contract-based world and begin to think in terms of services, not applications.

Web Services is probably in your future, if you plan to stay in business. The key is to start identifying target applications now, and beginning to work with business partners who can see the value of lowering their costs and speeding time-to-market with new and changed applications.

Teubner, Russ & Wilson, Michael, "Integrating Legacy Applications as Web Services," zJournal, April/May 2003

### **Appendix 1 – Useful Links**

- [www.w3c.org](http://www.w3c.org) - WWW consortium for defining web standards
- [www.Web Services-i.org](http://www.Web Services-i.org) - Web Services Interoperability organization
- [www.soapware.org](http://www.soapware.org) - SOAP developers' site
- [www.webservices.org](http://www.webservices.org) - neWeb Services and information
- [www.oasis-open.org](http://www.oasis-open.org) - global e-business standards organization

### **Appendix 2 – What is BETELGEUSE?**

Betelgeuse is the brightest star in Orion and marks the western shoulder of the constellation. Betelgeuse is one of the largest known stars and is probably at least the size of the orbits of Mars or Jupiter around the sun. That's a diameter about 700 times the size of the Sun or 600 million miles. For a star, it has a rather low surface temperature (6000 F compared to the Sun's 10,000 F). The low temperature means that the star will appear orange-red in color. Betelgeuse emits almost 10,000 times as much energy as the Sun. The combination of size and temperature tells astronomers that the star is a kind of star called a red super giant. ( Red super giants are stars that are close to the end of their life.) Probably within the next ten to hundred thousand years Betelgeuse might end its life in a [supernova](#) explosion. Will Web Services last this long?

Source <http://www.astro.uiuc.edu/~kaler/sow/betelgeuse.html>

