
Chapter 4: The Next-Generation Network and IT Systems

Introduction

In Information Technology circles, the discussion is all about Service-Oriented Architecture, Web Services and Grid Computing. A few forward-looking individuals even know about Web 2.0 and the Semantic Web. But take even a cursory look at the IT systems inventory of any established carrier and you will find systems going back to the 1980s and earlier. Carrier systems are rarely retired: instead, new applications and architectures overlay old ones until a kind of geological stratification occurs.

The problem with carrier IT is not so much how to absorb new technology, as to figure out how to use it cost-effectively to deal with the legacy of the past: both the obsolescence of ancient hardware, operating systems, computer languages and applications; and the spaghetti of standalone systems, ad hoc interfaces, and manual workarounds and re-keying of data. We could also include the lack of any common data architecture, schema, naming scheme or record format consistency.

In carriers, we normally distinguish between BSS (Business Support Systems) and OSS (Operations Support Systems). You would find BSS in any large enterprise: these are the systems which support standard business processes such as sales and marketing (CRM), enterprise resource planning and management (ERP/ERM), and billing. OSS is much more tightly focused to telecoms, including the element and network level management systems which configure the boxes, acquire and aggregate status information, and manage faults (trouble-ticketing).

I will start with a diagnostic tour around a typical carrier's IT infrastructure, and its attempts to move forwards. Although this is based largely on my own experience, please be assured that what is about to be described is completely typical.

The state of carrier BSS and OSS

Towards the end of the Internet boom, I was appointed chief architect at a global carrier, with particular responsibilities for information technology systems. During the 1980s and 90s, the carrier had steadily accreted systems. It sometimes seemed that in the three dimensional space comprised of products, processes and networks, every resultant cell had its own, special IT application to make something happen. Perhaps you suspect exaggeration? Figure 1 shows the UK Business Support Systems (BSS) map across products and processes at that time.

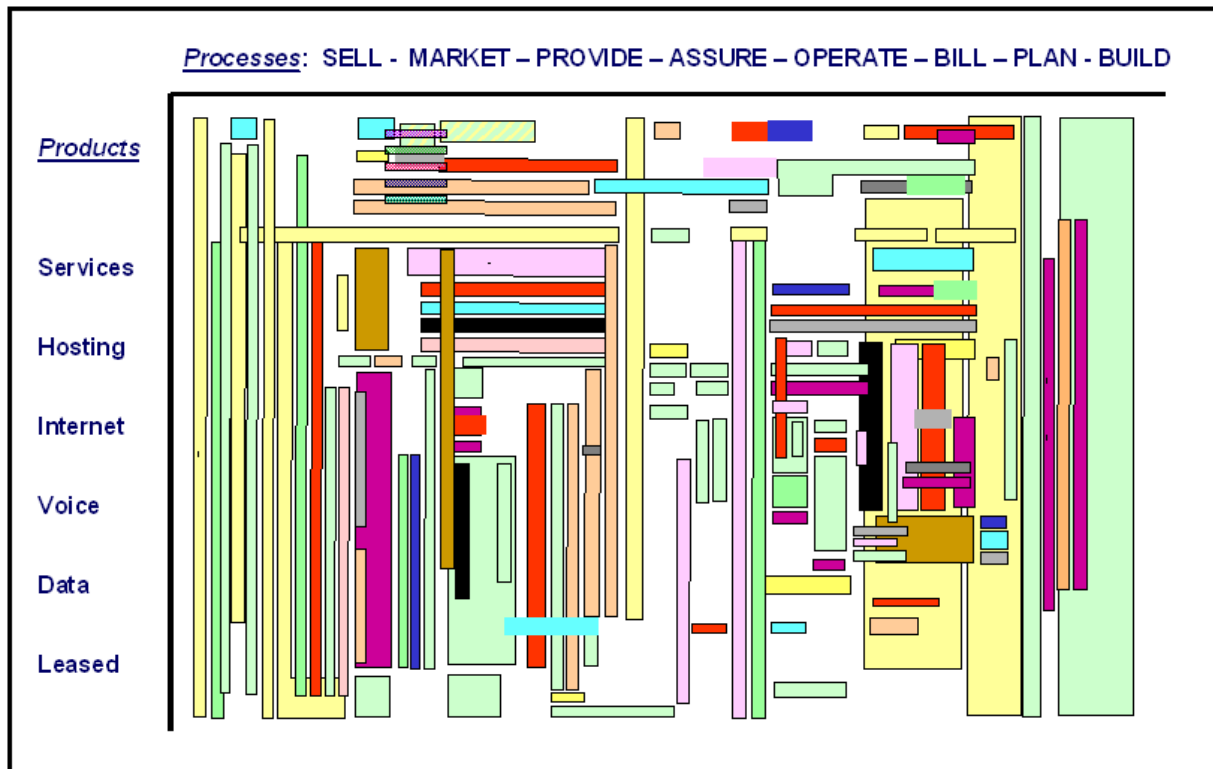


Figure 1. A typical carrier systems map

Products are listed down the left-hand side. Business processes are listed along the top. The boxes are IT systems. To avoid any confidentiality issues, IT system identifying information has been removed and affine transforms applied. Each box is a system, and 90% of the boxes are separate systems (rather than the same system supporting different functions).

The problem is worse still. I have not shown the Operations Support Systems (OSS), which were used to monitor, provision, configure and control network equipment, and to handle alarms. The corresponding OSS diagram contains a further constellation of IT systems amounting to more than half the number of applications documented in figure 1.

Nor were the applications as decoupled as the diagram suggests. Some key applications (such as network inventory management applications) had as many as 40 to 50 interfaces into other applications which needed to use them. These interfaces were entirely custom-built, and had been added over many years in a variety of computer languages and networking paradigms. This proliferation of diverse, complex systems with, in many cases, manual interconnect (requiring data re-keying) had many negative consequences.

- Processes were locked down by the inflexible computer systems.
- Due to the detailed product-specific implementations, systems and process re-use was very difficult. To add a new product one was faced with the unenviable dilemma of either expensively introducing a new suite of IT applications, or engaging in a very expensive rewrite and customisation of existing applications. Some of those applications were written in Fortran!
- There were no economies of scale either in processes or systems.
- Even small changes had to go into the IT pipeline to be fixed. This frequently took months. Departments which viewed such a lack of responsiveness as unacceptable took to recruiting their own secret groups of developers, outside IT’s view or control. There was an IT underground of rogue development and applications.
- IT was generally hated and despised, and became a scapegoat for program failures. This last point particularly irked the IT staff.

I am not being particularly harsh about my employer at that time: all carriers with some history behind them have exactly the same problems. And it was not as if we didn’t know how to design IT systems which solved these problems. The industry had wrestled with inflexible, stovepipe IT throughout the eighties. A consensus had developed around the following approach which is depicted in figure 2.

- Use COTS (Commercial Off-The-Shelf) packages, rather than in-house applications.
- Use enterprise application integration middleware to ‘glue’ applications together.
- Use Internet technologies for maximum flexibility.

In fact one of my counterparts from another carrier had evangelised a similar target architecture around *his* organisation, under the name of ‘the star-ship’ - there *is* a similarity if you look carefully.

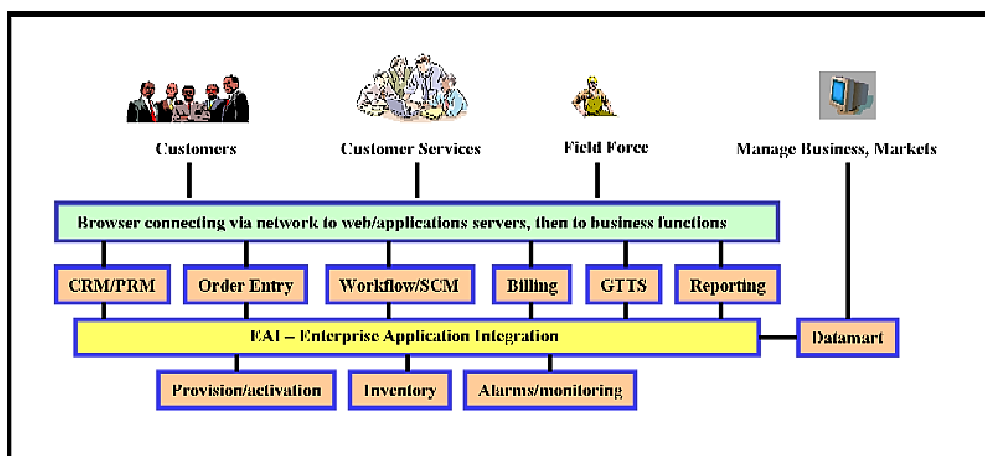


Figure 2. Today’s BSS-OSS architecture

For a 'green field' carrier (i.e. a start-up) there were no issues. Just do it like the star-ship. When we acquired a major US data hosting company, all of three years old, we found it had a modern flow-through automation layer exactly aligned to the model, and sourced entirely from PeopleSoft (now Oracle). We spent months trying to decide if we should break their model by replacing their PeopleSoft financial component with SAP, as used by our extremely influential finance people.

For 'legacy carriers', however, structured entirely around incompatible architectures and systems, and which cannot not be shut down for the purposes of migration, the issue is how to introduce new architecture and technology *at all*.

The consensus to-date has been to migrate from in-house applications to COTS packages and to use EAI - *Enterprise Application Integration* - to tie applications together. EAI is the magic ingredient here. The EAI application, or hub, supports standard interfaces to standard applications - it can exchange messages with Siebel, SAP, popular billing applications etc. It then acts as a post office, accepting messages from one application and delivering them to another, according to configurable business logic scripts. EAI interfaces can also be written for a carrier's proprietary legacy applications, although usually at vast expense (c. millions of dollars). The currently fashionable phrase is to call this architectural concept the Enterprise Service Bus, especially when it supports web services interfaces.

For the last few years, we have been meant to get excited about web services as the real answer to systems modernisation. In the web services model, applications are like 'objects' with public interfaces - 'methods' - which can be invoked to execute a business or network function. Applications publish their web services public interfaces in a UDDI registry (UDDI = Universal Description, Discovery and Integration) which can then be searched by other applications in a 'yellow pages' model.

It is not sufficient just to have all applications promiscuously exposing random interfaces in case someone else wants to access some of their functionality - the exercise must be structured and controlled. A 'Service Oriented Architecture' (SOA) is one which specifies needed functionality and determines which interfaces ought to be published. Legacy applications can also be given web services interfaces (at a price!) so that they can also participate in the Service Oriented Architecture.

Within a web services world, the distinction between the application and the EAI middleware gets eroded. Applications are designed from the very beginning to import and export their specific functionality and communicate bilaterally. It looks more like a meshed world than a hubbed one.

How does an application talk to another using web services? Recall that the web services model is object-oriented, so that a message is sent to a web services interface, and after a computation, a reply message is sent back. This is like method invocation in an object-oriented programming language (e.g. Java) or a procedure call for any reader whose last experience of programming, like the author's, was COBOL. The procedure call is coded in an XML format, and inserted into a SOAP message. (The function of SOAP is to identify the 'method', package the parameters, set-up a transaction-id to coordinate the request and response, to manage security encryption/decryption and to handle any other transaction-related administration - it is like a combination of envelope and routing note packaging a document).

For example, suppose that an application connected to a music download site needs to check with the inventory system of a supplier. From its own database, the application believes the product ID is PQ85a. It therefore sends a SOAP message to the inventory system asking for full product details as shown in Listing 1.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductInfo xmlns="http://catalogue.example.com/d1">
      <productID>PQ85a</productID>
    </getProductInfo>
  </soap:Body>
</soap:Envelope>
```

Listing 1. An example SOAP message request

The Inventory system now does a look-up in its own database and finds the required product details which it sends back in the response shown in Listing 2 (adapted from the Wikipedia discussion of SOAP).

SOAP is normally carried across networks using HTTP, just like HTML messages between browsers and websites. As can be seen, SOAP messages occupy a lot of bytes, and routing messages around networks also appears somewhat challenging. In fact web services has opened up a new application-level network layer, centred around handling SOAP and XML messages which the existing EAI hub vendors have been quick to exploit, along with traditional network equipment suppliers.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductInfoResponse xmlns="http://catalogue.example.com/d1">
      <getProductInfoResult>
        <Category>Classical</Category>
        <productID>PQ85a</productID>
        <description>Goldberg Variations</description>
        <Performer>Glenn Gould</Performer>
        <Composer>J. S. Bach</Composer>
        <price>20.00</price>
      </getProductInfoResult>
    </getProductInfoResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 2. An example SOAP message response

Functions at this layer include access management and security, routing of messages, schema transformation, performance monitoring, load balancing, compression and caching. Devices supporting such functions in hardware or software are called 'content-aware network appliances'. They can be folded into blades running on network switches and routers, as in Cisco's Application-Oriented Networking (AON). IBM, Intel and Citrix are also prominent players. This level of attention has also identified some weaknesses in the existing web services architecture: specifically the current absence of standardised application-level routing protocols between SOAP end-points, paralleling the services provided by IP routing protocols such as OSPF.

I mentioned we were meant to get excited about web services, but over the last few years the impact of web services on carrier IT has been bounded, to say the least. Web Services started as a concept, then developed into an architecture and moved to standardisation. Only then were programming language interfaces and system development kits able to emerge, and their first versions were lamentable. We now have stable and sufficiently sophisticated web services development platforms, but it is taking time to re-engineer traditional COTS packages into the new architecture. Until that is done, re-tooling carriers will continue to be difficult, even ignoring the issues of legacy. And this is the reason why the revolution is progressing on the timescale of a decade, rather than months or years.

Project Ultimate and project Diamond

Shortly after my arrival I was introduced to the showpiece IT project which was going to save the company. I will call it 'Project Diamond'. Project Diamond was the *second* attempt to sort out the log-jam of obsolete systems which had dragged my employer into the morass. Before Project Diamond there had been an initiative I will call 'Project *Ultimate*'.

Project *Ultimate* had been the ultimate big bang. The CIO had frozen all IT spend and had apparently removed his team and himself from our own universe to design and implement *Ultimate* in great secrecy somewhere else. *Ultimate* was not conceptually wrong: its architecture was exactly that of figure 2. The problem with *Ultimate* was that six months had gone by, all departments had backlogs of work which IT had refused to work on, new products could not be introduced, and when finally the great *Ultimate* launch meeting was held, no-one in the audience could understand what on earth was being proposed.

After the CIO had duly relinquished his post, big bang transitions became unfashionable. The new incumbent decided on an incremental approach, hence project *Diamond*. *Diamond* was also an instance of the target carrier architecture of figure 2, but rather than trying to replace all of the existing IT systems at once, it had the incomparably simpler task of just supporting a couple of new products. Even legacy data migration would not be necessary.

The people doing *Diamond* were from a major consultancy, and you could not have met a nicer bunch. They were friendly, hard working and frequently took us to dinner. They made all the decisions themselves, and integrated sensible mainline applications (not all of which were in use in our organisation at the time). *Diamond* delivered a textbook system, roughly on time and at staggering cost. It didn't take long before we noticed that no-one was using it. For *Diamond* to be inserted into the life of the organisation, major products would have to be migrated onto it. This meant that the elegant end-to-end automation system the consultants had built would have to be integrated with tens to hundreds of legacy systems:

- network inventories to check circuit availability
- customer databases for order management
- billing and invoicing systems (of which there were many)
- existing fault management systems
- partner management systems.

And of course, there were many different systems under each of these general headings across the different products and geographies. As an integrated, flow through automation platform, *Diamond* could not deliver its value until these interfaces were in place.

The consultants were prepared to grit their teeth, get stuck into what was evidently shaping up to be a multi-year project, and continue spending our money. Meanwhile, products were being further delayed because they were told not to deploy on legacy systems, but to 'wait for *Diamond*': and the routine maintenance that every department needs was still not being done. Tensions mounted and no IT job looked safe.

Soon we had a new plan. Get the front-end CRM process sorted out and we could at least get the benefits of a standard package in use across the organisation. Flow-through automation would have to wait. I believe this was the first point that reality had intruded into IT for at least a year. Of course, because we were now confronting real departments with real customers and real processes, we started encountering real problems. Locally, the front end-processes, although idiosyncratic, were not bad. They had adjusted to failings in downstream systems and over the years had been customised and tinkered with until all the internal departments were happy. The new, standard system had none of these idiosyncrasies of course. It reflected the vendor's model of some generic industry norm for doing CRM and order entry, and like a new shoe, it pinched. But it did have one major advantage over the status quo - superior, automated reporting to senior sales management. This was enough to get it steam-rolled into operation.

As business conditions worsened, the consultants were let go and development reverted to in-house staff. Slowly the grand transformation plans were abandoned, and incremental development became the norm, focused on where the pain was greatest. Strategic progress stopped altogether, but surprisingly, at the margin, where the pain was worst, things improved.

For example, every month the sales and operational senior managers got together to plan production for the next period. What they wanted to do was simple: review the orders pipeline and network occupancy so that they could schedule capital spending in advance. The idea was, as far as possible, to do just-in-time investment. Of course, the IT systems were incapable of delivering this information. Customer and order information was shredded across dozens of databases and spreadsheet files. The network capacity information was also spread around amongst transmission circuit databases, switch loading statistics and Frame Relay and ATM virtual circuit files mapped to switch occupancy figures.

The COO set up a small team to fix this problem, ignoring IT altogether. The task force identified the relevant databases (there were more than 40) and wrote web scripts in ColdFusion to automate data retrieval. In the case of spreadsheets, they had to write tailored code to map these into a database.

Every month they would access the databases, pull out the information, reformat it, combine and summarise it and then produce formatted web pages and RAG (Red-Amber-Green) reports. Sticking plaster of course it was, but I attended some of the sales-operations review meetings, and they would not have happened without it.

In the end we had teams of web-engineering programmers, who could glue web front-ends onto legacy systems and provide a superficial layer of integration to grateful staff. It was useful. Meanwhile, the heavy engineering of COTS introduction and legacy dismantling went on in the background, to a muted bass tone of user pain, and the gurgling sounds of millions of dollars seeping away.

Is there a better way?

The CIO job in a carrier old enough to have layers of legacy systems is a completely poisoned chalice. The CIO and his staff are often some of the brightest people around. These days, they are as well versed in business realities as in the technical state-of-the-art. Yet change doesn't happen, and careers are destroyed. Why?

Consider Machiavelli's over-familiar observation from Chapter 6 of 'The Prince'. "*And it ought to be remembered that there is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, then to take the lead in the introduction of a new order of things. Because the innovator has for enemies all those who have done well under the old conditions, and lukewarm defenders in those who may do well under the new.*" [1]. A modern IT architecture is indeed a 'new order of things' and it needs a number of separate transformations all to be successfully accomplished, with benefits accruing mostly at the end.

The task has to be modularised - a big-bang cannot work. But if only some of the processes and systems are to be modernised, interfaces have to be built to the remaining legacy systems, otherwise the process-flow breaks. How can such expensive new interfaces be justified, when they only connect to ancient systems which are due to be phased out in their turn anyway?

It will be found that data is shredded across the organisation. I have already alluded to the fact that information about sales to particular customers was spread across 30-40 different databases. And some of these databases were spreadsheets. And the field names were different in the various systems. So it was never quite clear whether the reference was to the same customer and product, and whether we were double counting.

There is no-one to do the work. Most people today in carrier organisations are working ridiculous hours just to do their line jobs. There is no slack to document existing processes, design new ones, configure and tailor the COTS packages from out of the box, run the necessary trials and implement pilot programmes, to educate the staff and to manage the inevitable headcount reductions. It is an uncomfortable truth that most of the desired OPEX improvement resulting from major IT investment comes from the headcount reductions achieved through more efficient automated processes.

I do not know of a single case of a legacy carrier successfully migrating its entire organisation to a successful, state-of-the-art IT system, together with a suite of flow-through processes. As we found from our data centre acquisition, there have been cases of start-ups, particularly in the Internet boom, who were able to build their IT from scratch - and the resulting efficiencies were an existence proof of the virtues of success.

Does this mean that it is hopeless? Almost. Only the gravest crisis can justify the expense and disruption of a comprehensive step-change in processes and systems. The transition to the next-generation network demands sufficient product and network innovation to cause just such a crisis. However, it is still so expensive that an under-capitalised alternative operator may well be unable to meet the bill. The NGN will be the catalyst for a wave of consolidation leading to healthier businesses, charging premium prices, which can afford to invest.

A Common Data Model?

One of my few successes as chief architect was my cancellation of the Common Data Model project. The consultants concerned had managed to convince a number of senior managers that what was needed was a common data model across the organisation. They had prepared a large number of binders when I first encountered the project, detailing the theory of data modelling, class and inheritance models, modelling languages and standardisation efforts. They were all set to spend further millions of dollars walking around our organisation documenting the myriads of names different projects had come up with over the years for what we ought to recognise were the same real-world entity. I went ballistic.

What was wrong was that the consultants had no answer to the question: 'what would change if tomorrow you gave us a completed common data model - what would we do with it?' There was no possible answer, because in the absence of standardised databases and applications adhering to the model, a common data model was just so much useless paper, obsolescing by the second. We had no IT programme at all which was pluggable into the Common Data Model activity, and what's more, the consultants knew it - 'easy money'.

I don't believe that for large carriers a single, complete and consistent data model is possible, although it is a worthy objective. There are just too many different processes and applications, with different data schema requirements. The key thing is to get the data modelling process under control. There must be a schema management procedure in the IT department - part of the gate process by which new IT applications are released to service. Today that means an XML-based schema, and it is possible to translate between different, compatible schema using standard XML tools, linked with the messaging middleware. This kind of distributed scheme is scalable, flexible, and robust against new corporate acquisitions.

At time of writing, I read that IBM is pushing 'Master Data Management' (MDM) [2]. Apparently the centralised MDM database should be updated constantly, so that it can push altered data to applications to ensure everything is always in synch. And of course, this needs a high-bandwidth network and a Service-Oriented Architecture. Naturally, getting to this Nirvana cannot be a big-bang, but must be a multi-year process. Still, Gartner apparently conclude that by 2010, over 70% of Fortune 1000 companies will have implemented MDM programmes. It coyly does not speculate as to whether any of these programmes will have succeeded, or what the other 30% thought they were doing.

A correct strategy for IT transformation

Having described a number of cases where change didn't work, are there any lessons for success? I believe the correct strategy for IT transformation is simply this.

Realise that IT transformation is simply an enabling mechanism for *business transformation* to a new, more efficient and lower-cost business. First commit to the business transformation programme, then commit to the IT modernisation programme as a key enabler.

No business transformation plus IT evolution can be done purely with internal resources. It needs a resourced programme team which has a mandate to engage with the entire organisation and *turn it around*.

No business case can be made for transitioning the entire organisation to the IT state-of-the-art. An audit will show that certain products (such as circuit-switched voice and synchronous transmission) are probably not too bad, although based on obsolete processes and IT, and consequently with a too-high cost base. But since we are going to close down these products and networks anyway as soon as we migrate to IP-based products, the right thing to do is maintenance only: apply process and IT *sticking plaster* where the extra revenue from any marginal improvement clearly and immediately covers the marginal cost.

Finally, there are so many vested interests who seek to slow down, avoid and dilute process and systems change that success will only occur if led from the top. The CEO and C-level managers must not assume this can be delegated down the line. They need to chair the programme board, and it needs to meet often.

In the nutshell, carriers have not been good at re-inventing themselves, and that is why their IT transformation programmes have failed.

An Internet self-service model

Anyone who has used a company with an Internet sales channel like Amazon, Cisco or Dell will ask themselves why carriers cannot provide their customers with a similar interface. I have heard the excuse that the carrier business has a far more complex product line than the relatively simple product models of online retailers. Agreed, a book is not too complex, but Cisco routers and Dell PCs? Please!

The real answer is that an eBusiness architecture turns the existing carrier waterfall-type IT architecture inside out. In the carrier IT systems model we usually see today, the customer and sales person get together to configure and price the order, which then enters the front-end CRM system. The order then burrows deeper and deeper into the BSS and OSS, down to the network boxes, out to the field force and across to other carriers (for buying in private circuits, for example). Deeper in again, the order percolates through to the billing system and various management information systems. Finally by some magic, tens or hundreds of sub-tasks run to completion, often after multiple resubmissions to correct errors, across tens or hundreds of separate systems. The service finally becomes available to the customer. Perhaps more than 60 days have gone by. It's hard enough for the carrier's staff to find out what is going on. As for the customer - not a chance.

How do we get from there to an Amazon point where we appear to be able to point our browser at ‘My Account’ and find out everything - what we ordered, where it is, when it will arrive, all our billing and shipping history, our account and transaction history?

The answer is a customer-centric IT architecture organised around central integrated applications and databases. Technically this is called a three-tier architecture (figure 3) - generalised to n-tier architectures, as further layers of processing differentiate from this simple model.

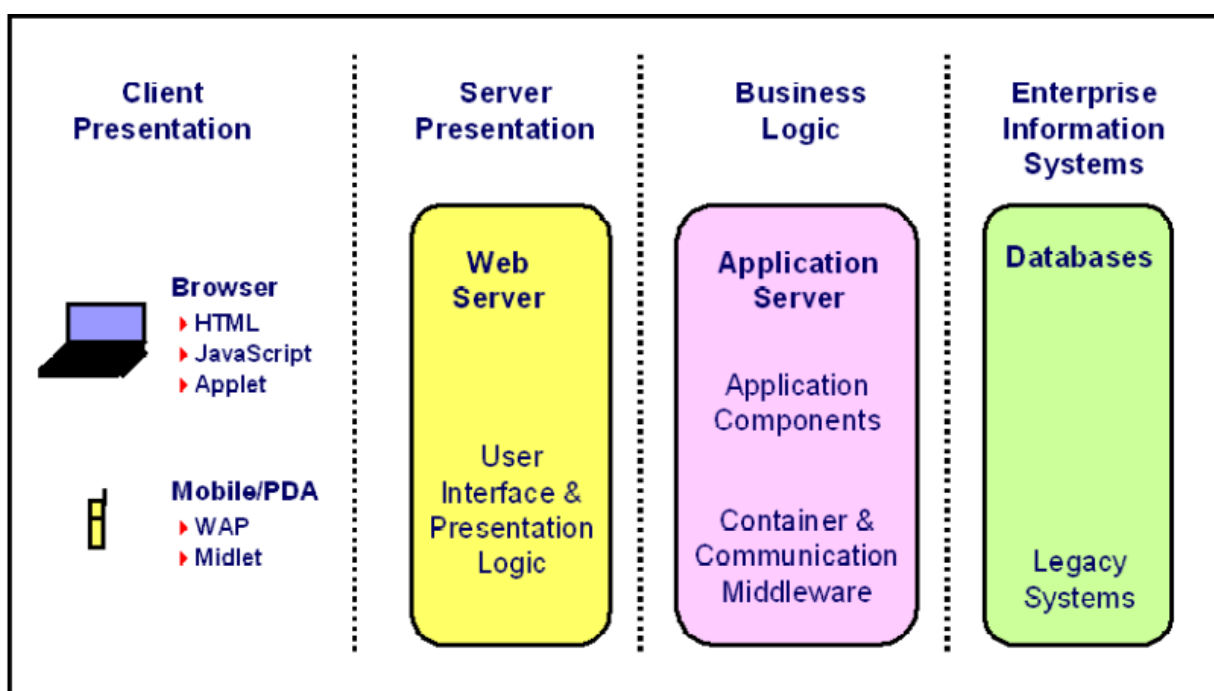


Figure 3. Three tier architecture

In a three-tier architecture, the customer’s browser is pointed at the top-tier, the Internet portal site. The portal’s technical function is to receive HTTP requests from the user, and to return HTML back to the browser, but a portal does a lot more than that. As the service gateway, it proves the first level of service integration. Different parts of the page serve-up different functionality to the customer - windows to diverse functionality encompassing the totality of the underlying process network.

The middle-tier is the domain of application servers, running business logic components. These are the present form of what we used to call applications, and a coherent set of business functions could be

layered onto a number of physically separated but networked components, running on different servers in different data centres. Communication is mediated by web services and SOAP messages.

The third tier comprises databases, or those legacy enterprise information systems which are not yet assimilated to this architecture. A business component will treat the third tier as an information resource.

Familiar instantiations of the three-tier architecture include the Java-based J2EE, and Microsoft's .NET.

The power of the three-tier concept is the separation of concerns. The user experience can be designed autonomously within the top-tier portal. Application logic can be developed without regard to data formats in the middle-tier, which is where enterprise application integration is also addressed, while all the issues of data format, schema development and data modelling, as well as legacy system integration are concerns which can be addressed within the third tier. It has been a real stretch, however, for carriers to adopt this model as fundamental to their IT.

As already observed, vendors of COTS packages are still re-engineering their historically monolithic applications to this new model, while interfacing legacy applications into the tier 3 position of backend enterprise information systems and databases is a hard, grinding and expensive uphill task. But perhaps the most profound reason for inertia is that the dominant 'factory' model of carrier processes is not customer-centric at all. Instead, it is optimised around a division of labour and work-handover within its interior bureaucracy. Pervasive application integration can undermine the current waterfall process-model paradigm, but as the old joke has it, to adopt the new three tier, customer-centric architecture, you would have to really *want* to change (cf. how many psychiatrists does it take to change a light bulb? Just one, but it really has to want to change).

To summarise: the three-tier/n-tier architecture is the right one for today and the only way to achieve reasonable customer service but it has to be *consciously* adopted. The dangers of upgrading to a rather traditional linear COTS + EAI model which has not been modernised as per the Internet architecture above should be obvious, even if it seems easier and feels more comfortable.

Grid Computing

After web services and SOA, I would now like to turn attention to the third major innovation area in IT today, Grid Computing. I first heard about Grid Computing in 2003 at an IBM conference in Florida. Our ever-attentive IBM account manager took great pains to bring it to our attention. I didn't understand it at

all. What was Grid Computing? Was it something to do with autonomic computing, the somewhat unconvincing image of a self-configuring, self-managing and self-healing computer infrastructure which IBM was pushing at the time? Was it some kind of variant of web services? Was it a new IBM operating system?

Even as we grappled with the concept, we were not clear what problem it was trying to solve, or whether we actually had that problem. A few years down the track, and the answers to these questions are becoming a little clearer. Enthusiasts like to explain that the idea of grid computing is that computer power should be a utility, like water or electric power. Just plug your terminal into the computing grid and extract enough computing power to solve your problem. Obviously there would have to be excess computing power 'out there', which you can tap into with the usual concerns for security and cost-efficiency. This seems to me, though, to be the *least likely* application of grid computing. Here are some more plausible scenarios

- You are an enterprise with a diversity of computing resources across your data centres and desktops. You have little idea how CPU, memory and disk resources are being utilised, as you can neither measure them, nor systematically allocate them to your application schedule. But you are sure that your usage is grossly inefficient.
- You are a collaboration between a number of organisations with strong joint computing requirements. You would like a uniform 'virtual' computing environment which allows anyone to submit jobs and get the results, but this has to be implemented across half a dozen different IT organisations with different processes, approvals, security and standards.
- You have invested in a data centre and servers, and you would like to make a business of selling processing to all-comers across the Internet. If only there was a way to set-up, package and bill for this service in a secure way. (Well perhaps this last scenario is not unlike the naive utility model mentioned above!)

To make any of these scenarios work, what is needed is a kind of distributed operating system which can sit on top of a scalable network of distributed computers. Looking down, this new software has to know about the machines it has enrolled, and track their utilisation and capabilities. Looking up, it has to accept jobs from clients, map them optimally onto machines with capacity, monitor execution, handle exceptions, produce billing records, and manage security. Welcome to the world of Grid Computing.

The Global Grid Forum

The world-wide efforts to put grid computing onto the map are being led by the Global Grid Forum (<http://www.ggf.org/>). According to its website, the GGF represents more than 400 organizations and is the leading global standards organisation in grid computing. A tutorial on grid computing is available at CERN, the particle physics research establishment where the web was originally developed (<http://gridcafe.web.cern.ch/gridcafe/>). The Global Grid Forum was formally set up in 2001, and organises conferences, supports working groups and produces technical papers. It sponsors the Open Grid Services Architecture (OGSA) which is the global standard for grid computing.

Grid infrastructure reuses many of the standards and interfaces of web services (for example, for security). The major extension required for grid computing is the management of state. While web services are stateless, grid computing has to explicitly manage the state of computer resources. It has therefore introduced the WS-Resource standard.

Many organisations experimenting with early grid implementations are using the Globus Toolkit, (<http://www.globus.org/>) which implements the OGSA. At the time of writing, the Globus Toolkit is at version 4.x. and has received excellent reviews. For more information on Grid Computing see [3].

Using a Grid infrastructure

Like any operating system, a grid infrastructure will manage secure access, and allocate permitted jobs to available resources. Functions in existing distributed middleware architectures such as CORBA - distributed fault-tolerance and parallelism of execution, for example, will also be supported. The utility of grid computing in enterprise data centres is obvious. Whether it will permit existing unused desktop computing resources to be brought into the virtual server space is less clear - the IT staff have far less control over these machines, even as to whether they are switched on or not. Desktop resources have also to be shared with the allocated user, who may well object if the system is slowed through executing high-priority back-office functions.

Grid computing may make extranets easier to set up, but despite the emphasis on ‘virtual organisations’ in the grid computing literature, commercial applications still seem a stretch. Most of the existing work is with large scale ‘big science’ projects. Likewise, the concept of a public Internet grid, where jobs can be executed on a networked cluster for a fee, seems a long way off. Most applications run perfectly well either on the client machine, or on an enterprise cluster which seems to the user to be a remote enterprise machine accessed via the browser and HTTP (even though it could well be executing on an enterprise

grid in implementation). Enterprise grids are being promoted by the Enterprise Grid Alliance (<http://www.gridalliance.org/>) but progress still seems slow.

So the conclusion on grid computing is that it will certainly do for distributed computing infrastructure what web services aims to do for enterprise application integration. And like web services, it will probably be a while in arriving in a form which enterprises and carriers would actually want to deploy internally.

Web 2.0

Considering this chapter so far, I am rather aware of 'big systems chauvinism'. Much discussion about back-end systems, not much about users, particularly carrier staff. There is a category of IT known as *desktop support*, which deals with PCs and horizontal productivity applications such as email. They also run the IT help-desk. Although it will offend them to say so, I have to say that from a CIO perspective, this whole area is not very exciting. When it works well, nobody cares, and when it breaks, it's nothing but grief.

Part of the reason for the lack of excitement is the plateau of functionality. We seem to have got to Microsoft Office quite early, perhaps in the mid-nineties. That gave us email, contacts and calendar, a spreadsheet, word-processor and a presentations package. Since then, possibly excepting tools such as diagramming systems for engineers and desktop databases for analysts, it has been hard to identify any new, truly pervasive horizontal applications.

This has been frustrating for people who wanted to move up a gear into collaborative tools. The Intranet has managed to move some traditionally paper-based forms onto screens, but seems in general to have disappointed. It is hard for staff to publish to enterprise Intranets, and hard to find stuff once it's there. But the Internet continues to be a source of technologies both scalable and usable. It's time to think about Web 2.0.

A few years back, I was briefly enthusiastic about the semantic web. This continues to be a multi-year effort from the world-wide web consortium to make web content semantically clean and precise, and independent of any underlying application. The idea is that when you search on, for example, the word 'rock', the resources out there on the Internet should know whether they pertain to geology, music, an actor or repetitive movement. To do this resources need to be described in a language which is both rich enough to capture distinctions and relationships, and which is equipped with a broad enough, standardised

vocabulary - an ontology - to specify the type of thing resources are, and the conceptual class hierarchies which resources belong to.

I recall attending a meeting about the Semantic Web with Professor James Hendler, a leading researcher, back in 2002 at Virginia's Center for Innovative Technology, off the Dulles toll road. "Who exactly is going to tag the millions of web pages with semantic web mark-up?" I asked. Professor Hendler was not sure, and I thought at the time this was a bit of a show-stopper. If people weren't going to do it, then natural language machine systems were going to have to get a whole lot smarter in understanding content (see chapter 12). And if they became that smart, perhaps we didn't need to do the semantic web tagging in the first place?

Five years later and the Semantic Web is still work-in-progress, but the idea of tagging has caught on. Not the finely analysed conceptual hierarchies of the professional ontology developers, but folk-tagging by millions of ordinary users applied to user-generated content on an increasingly large number of sites. There is a widespread belief that there are powerful network effects which will drive tag-convergence to a standard 'folk ontology' from below: we shall see.

User-generated content sites which encourage tagging need to provide automation to assemble ontologies bottom-up from the most popular tags being proffered, and to present possible tags to their customers as they seek to classify. This is the fast way to get convergence. With widespread and consensual tagging in place, the synergy with search engines is readily apparent. The promise is: 80% of semantic web functionality for 20% of the effort.

Another not-wholly-predicted success story has been wikis, such as the Wikipedia. Again, the software provides an easy to use framework for users to add and review content - their own and that of others. Arguably-utopian beliefs that convergence to a high standard of content would occur seem to have been largely born out in practice, lubricated by a light touch of moderation (as in - helped along by Moderators).

Some of the more enterprising vendors are putting all these technologies together as enterprise wikis with tagging and search. For enterprises whose main asset is the intellectual property of their staff, this is probably a new kind of desktop software worth having, once past release 1.0. I would like to believe that next-generation network carriers should also be included in this elite category.

Web 2.0 should not be *conceptualized* in terms of its new applications (wikis, etc), or its user-generated content, or its software-as-a-service potential. In today's Internet, connectivity is important, of course, but the *paradigm* has been defined by specific applications: web servers, email, etc. In web 2.0, the paradigm is *connectivity itself*. Around the notion of 'platform connectedness' we see an explosion of new ways to converse: asymmetrically and peer-to-peer, people and application systems. As we discussed in chapter 2, the NGN is foremost a middleware platform, a nursery for new patterns of communication, which can then be packaged as services.

Summary

In this chapter, we have looked at the train-wreck which is today's legacy IT infrastructure. The result is strategic immobility, as it is usually cheaper to build an additional IT layer alongside existing systems for a new product. The overall ratcheting-up of costs due to this added complexity is not usually identified in the business case. We then looked at modern ways to build carrier IT systems, using the 'Enterprise Service Bus' concept and a 'Service-Oriented Architecture' exploiting web services.

The practical problems of IT transformation were then examined in case studies of projects 'Ultimate' and 'Diamond'. We briefly touched on issues of data management, and a 'common data model'. We then looked at strategies for transformation which have a greater prospect of success, and explored an Internet self-service model for carrier IT. We finally assessed Grid Computing and Web 2.0, both flagged as new technologies and concepts which need to inform the next-generation network project..

Getting back to basics, the critical problem, as always, is that of legacy. Legacy systems cost more to adapt to future methods of operation and new systems architectures than they are worth, and strangle attempts at process efficiencies. The beginnings of wisdom lie in the recognition that most legacy processes and systems are unrecoverable, and must be left to wither. The NGN is an opportunity to start anew, and the challenge for top management is to grasp this point, and force through the next-generation organisation against all resistance. The consequences of half-measures will be bleak indeed.

References

- [1]. Machiavelli, N., *The Prince*, Oxford University Press, 2005.
- [2]. Information Age, The master key, *Information Age*, p. R3, March 2006. (IBM sponsored supplement).
- [3]. Taylor, I. J., *From P2P to Web Services and Grids*, Springer, 2005.