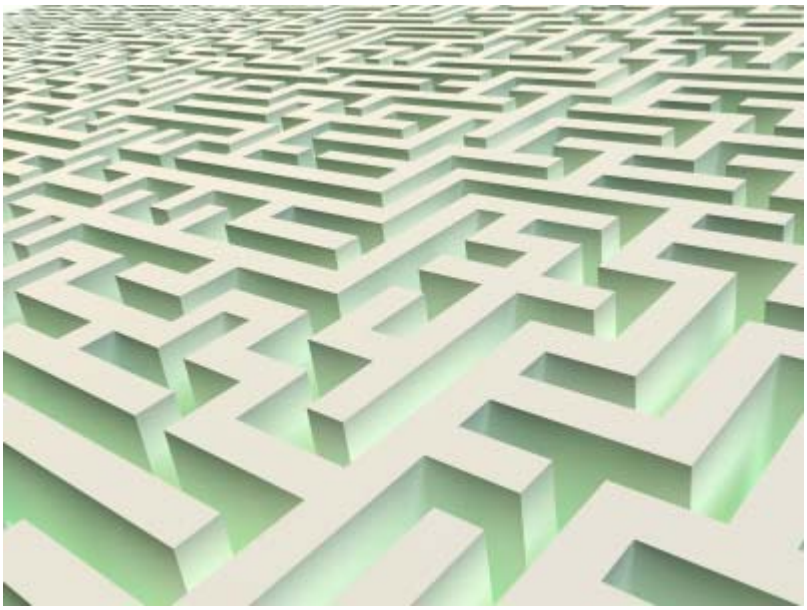

XML Business Templates

A whitepaper delivered at
XML 2000

Joe Gollner, M.Phil.

Revised and Expanded November 2006

Ottawa, Canada

INTRODUCTION

Having become a cliché by virtue of being true, it is far from risky to claim that businesses today are facing a brand new world wherein the walls have literally come tumbling down. It is the centerpiece of the marketing message for every self-respecting new venture in the business to business space to highlight the need to move data between disparate applications. But the realities of integrating business operations are in fact more complex and challenging than is ever mentioned in the brochures or even in the market research. Real business operations must contend with every manner of hurdle from security and performance through to accommodating the different levels of sophistication amongst participants in a business community. The fact is that we do not always get to choose our partners and nor can we foresee what changes lie ahead.

Given the foregoing, the challenge facing business integrators today is to design solutions that meet these challenges on every axis on which they can occur: cost, time, location, capability, media and so forth. XML provides the answer in a number of ways and these ways can be best introduced using the notion of the *business template*.

A business template is a form of overlay that connects the set of application components prepared by a business integrator to address the requirements of a specific business process or operation. At the heart of the template lies XML: the template in fact rests on a foundation of XML schemas and associated documentation and application components. The documentation components include implementation and operation guidelines and demonstration data holdings. The application components represent the *base* level of application support required to operate all aspects of the business operation being fielded. Every part of the template is designed to be explicit, meaning that it is easily understood and modified by people with a variety of backgrounds. And so it is that the application components developed within a template are precisely documented and, in implementation, are designed to provide the minimum (not the maximum) level of functionality needed to allow the process to go forward. For many implementers, this last feat of restraint is an almost unnatural act.

With this package of tools in hand, the business integrator can establish a genuine configuration management environment for controlling the underlying components, including the schemas, and can revisit any one component in the business template in order to work around a particular problem.

THE PAST

As soon as it became even remotely possible, organizations began exchanging information electronically and working out ways to load received data into their operational systems. While in theory this was straight forward to do, the reality was that in many cases people worked late into the night to make this integration happen each and every time there was an exchange. After a little time and several *open* and *frank* telephone conversations, the two parties attempting to exchange data would work out the specifics of the exchange at the most minute levels of detail. And things, from then on, would work out well - as long as nothing *changed*.

EDI

With the advent of EDI standards, a good deal of labour was saved and, for standard transactions, organizations needed only to adopt the necessary protocol and transaction sets in order to assert their expectations for incoming data. There was work still to be done, in integrating the data from the transaction sets with the target systems, but at least the transaction set specification became a common point of reference for all involved. While EDI suffered from many of the limitations of its technology era, it remained a substantial improvement over what came before. And just as people had taken to the phone in order to synchronize their exchanges, now people would work out their requirements in a standardization forum.

As is well known, the chief value of the legacy EDI standards lies in the huge amount of business knowledge that was distilled into the various transaction set specifications. It is also well known that this value was reached at a high price when measured in terms of time and effort spent in negotiation. Everyone should experience this type of negotiation just once in their professional lives.

Around these EDI standards grew an entire portfolio of application components that covered the full range of possible events including transmission, acknowledgement, validation, storage and delivery, and processing. The accrual of these application components is largely what put EDI out of reach for most businesses. At the time, there was no alternative to relying on expensive Value-Added Networks (VANs) and commercial EDI translators. And then there was the work of collecting and loading the data, or of unpacking the transactions and making the received data available for processing. And if someone needed to deviate from what the standardization had determined, then the

entire weight of the application environment fell onto their shoulders. So things worked reasonably well - as long as nothing *changed*.

The mental image that comes to mind is that of Salmon swimming up stream. The ideas were there, but not the infrastructure.

SGML AND XML

With the arrival of structured markup languages, it became possible to address a wider range of interchangeable information types than could ever have been addressed under the auspices of the EDI standards committees. It also became possible to deviate from any given standard even when there were numerous benefits associated with maintaining standardization. Not least of these benefits would be the ability to broadly share application components and to be able to integrate systems using a globally common point of reference.

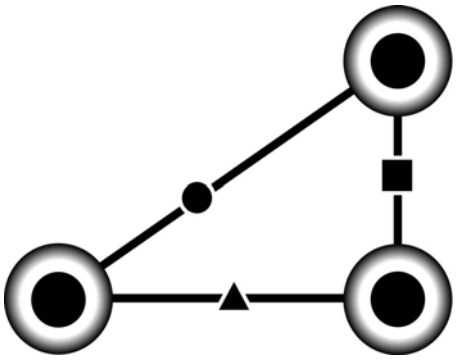

Figure 1 – Custom Data Interchange

Following the original pattern of data interchange, the initial efforts at exchanging data using structured markup were highly custom solutions intended to move data between two parties. As the content being addressed was generally different than that already support by the EDI standards and given that this content was often voluminous by comparison to simple business transactions, the mode of physical exchange was usually magnetic tape.

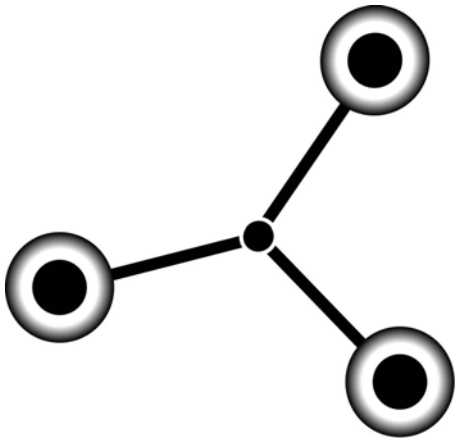

Figure 2 - Simple Data Interchange Standards

The level of effort associated with negotiating and then maintaining custom interchange protocols soon led to the adoption of what we will call "Simple Data Interchange". Under this scenario, Document Type Definitions (DTDs) were developed in much the same way transaction sets were defined for EDI in that a community of participants would negotiate a standard mechanism for exchanging their data. In principle, this would allow the trading partners to exchange data. In reality, the types of information typically being addressed by these DTDs were complex in that the number of possible permutations of data arrangement was usually infinite and the task of building applications to usefully process those contents was often very challenging.

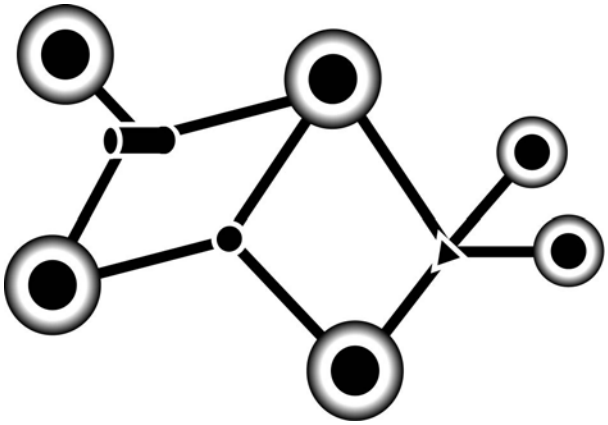

Figure 3 - Protocol Proliferation

As was regularly bemoaned in the technology press (and continues to be), the first generation of structured markup interchange specifications that were developed using SGML exhibited little if any

intentional commonality (and frequently widely varying levels of quality). Various communities of users, frequently hailing from within the same organization, began creating interchange specifications oriented to the needs of their particular vertical industries. When the time came to fund the development work required to integrate these interchange specifications into the internal operations of each of the participating organizations, the impact of even small differences came to be more apparent. In some cases, it meant that authors were required to learn different syntaxes and to modulate between different uses of the same term as they moved between different applications. In other cases, it meant that there would be redundant effort required to develop similar supporting application components in different environments to achieve essentially the same thing. Letting a hundred flowers bloom, it turned out, was not the best development strategy in this particular case.

Not surprisingly, organizational management soon became irked by an apparent lack of coordination. Their ire however should perhaps have been more appropriately directed at the lack of operational integration that gave rise to the stove pipes in the first place and not specifically toward the DTD development efforts per se. Nevertheless, a lack of coordination at all levels was manifestly obvious.

During the years in which XML has been front page technology news, some media pundits have declared that XML is fragmenting into many versions. Some even proclaim that XML's failure is inevitable in the absence of a clear, universally embraced markup standard. These views show just how ill-informed many in the press can be while still being free with their opinions. That there can be different markup languages created using XML *is* the point of XML. That people will always need the ability to deviate is a central tenet of any good systems design. That there will never be a set of universal standards applied to the level of business discourse is absolutely certain. That there are tremendous benefits to be realized through the coordination of markup languages both within communities and within organizations is also granted, but these benefits stand as an objective to which working practice can gravitate, as if toward a strange attractor amid the chaos of daily transactions.

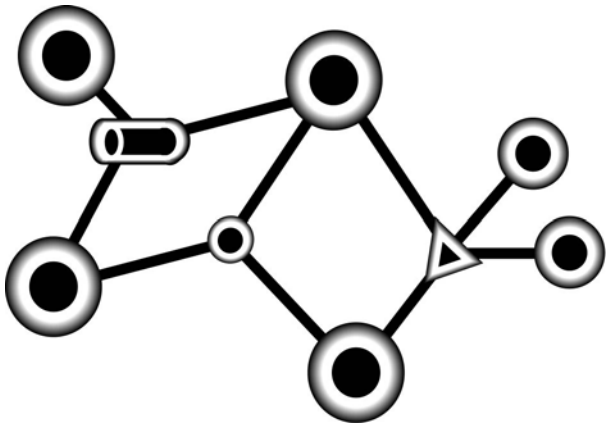

Figure 4 - The Evolution of Data Interchange

Again following the patterns set out by the EDI standards, as soon as they were put into use, each and every data interchange DTD began to evolve into a more complete protocol. This was in response to the desire amongst trading communities to facilitate the entire transaction through automation and to develop shared application components. By collaborating on application components, the overall cost of implementation could be reduced and processing outputs and behaviour could be made more predictable.

With the increasing adoption of structured markup for exchanging information, the accumulated burden for each organization began to be felt more and more acutely as organizations became involved in more and more trading partnerships. Add to this the fact that at the highest levels, the commonalities between interchange standards, that had evolved separately, became more striking as the scope of various standards grew to encompass larger and increasingly overlapping domains.

Finally, and again following the pattern seen with the more traditional EDI standards, many of the markup based protocols evolved in a direction that progressively raised the threshold for participation. The first generation of SGML interchange protocols, and even the first generation of XML interchange protocols, suffered from an increasing technical specialization that often made them less accessible to many potential users and inadequate documentation frequently compounded the problem. The automation components developed for many of these protocols battled against the challenges of addressing an infinite variety of contingencies and in so doing became problematic, incomplete and expensive to implement. Where changes were introduced, and they always were, the real disadvantages of maintaining overly complex application components came to the forefront. Once again, change proved to be *fatal*.

LESSONS LEARNED

Surveying the history of structured data interchange points out several key themes:

- interchange protocols are developed by *communities of interest* that are associated with specific types of transactions, usually occurring within specific vertical market sectors,
- interchange protocols evolve in such a manner that introduces increasing numbers of automation components over time,
- interchange protocols typically evolve in isolation from each other and from the organizational requirements of the participating trading partners with this giving rise to substantial levels of redundancy between interchange protocols,
- interchange protocols, and their supporting application components, gravitate toward increasing technical complexity while attempting to address sometimes huge numbers of contingencies,
- interchange protocols do not accommodate changes gracefully especially when, as is generally the case, the pressure to implement changes arises well after the initial implementation.

BUSINESS TEMPLATES

The rather awkward growth of interchange standards in the past can be directly linked to the initial, and almost exclusive, focus of those efforts on the *data* to be exchanged. This tight focus then leads to data interchange structures that come under constant pressure to adapt as more and more business requirements are encountered during implementation and operational use. Furthermore, the view of interchange protocols as data handling solutions leads to an overwhelming emphasis being placed in the technical aspects of the interchange problem. The solution to technical challenges is naturally to introduce automated components that will facilitate the events. But interchange transactions are in reality business events and ignoring this fact condemns interchange protocols to be the continuous recipient of unforeseen change requests, to which, from a data handling perspective, give rise to ever more inventive technical solutions and ever more challenging maintenance problems. In truth, many of the challenges

are much better addressed at the business level where a simple meeting of minds can avert weeks, if not months, of technical head shaking.

The reality is that the data interchange event is merely one physical transaction within a larger business process that involves numerous applications and infrastructure pieces, numerous rules and expectations, and numerous participants – human, organizational and technological. Once seen in this light, the manner in which interchange protocols are seen changes in subtle ways.

To be a little provocative, it is for the above reasons that the vast majority of schemas, which we will use to include DTDs, developed to date in either SGML or XML, are poorly done. Indeed, most of them are *perniciously* inadequate for the roles they are supposed to play.

A MORE COMPLETE PICTURE

It becomes clear, in surveying the history of data interchange, that from the outset, the focus should be placed on the business objectives being set by the participants and that these business objectives can then be decomposed into its constituent pieces - only one of which will be the technical implementation of the data interchange protocol.

It should also be clear that the evolutionary rate of change for transaction scenarios should be accelerated so that modifications can be made as early in the specification process as possible and certainly before the most significant development activities are undertaken. With the way it has been done in the past, evolutionary pressures are encountered after the investments have been made in many of application and infrastructure components. As is now abundantly clear, this is the expensive and risky way to approach the problem.

But if building the data interchange schema is not enough, what is to be done instead. The answer: data interchange schemas should be developed as part of larger business templates that seek to provide a more complete and balanced description of the data being exchanged, the events being supported and the requirements being addressed and that attempt to establish an operating capability for use in exercising the scenarios as soon in the specification process as is possible.

THE ANATOMY OF BUSINESS TEMPLATES

In truth, business templates are not as profoundly revolutionary as this historical build up might suggest. Rather, a business template is a more complete and more balanced approach to specifying interchange protocols designed to ensure that the emphasis is placed on the process being facilitated. This shift of emphasis then requires that the determination of the data interchange specification, and the associated application components, is done as an organic part of implementation activities as opposed to a predecessor. As the name suggests, a business template is a form of conceptual overlay that outlines what needs to happen. It is also a physical artifact that is intended to be accessible to the business managers who make up the sponsors of the initiative. It is a physical artefact in that it is a formal description of the process being implemented *and* the implementation of an initial operating capability. This initial operating capability is a collection of highly flexible, and therefore very simple, automated components that can evolve *in near real-time* in conjunction with the definition of data interchange specifications.

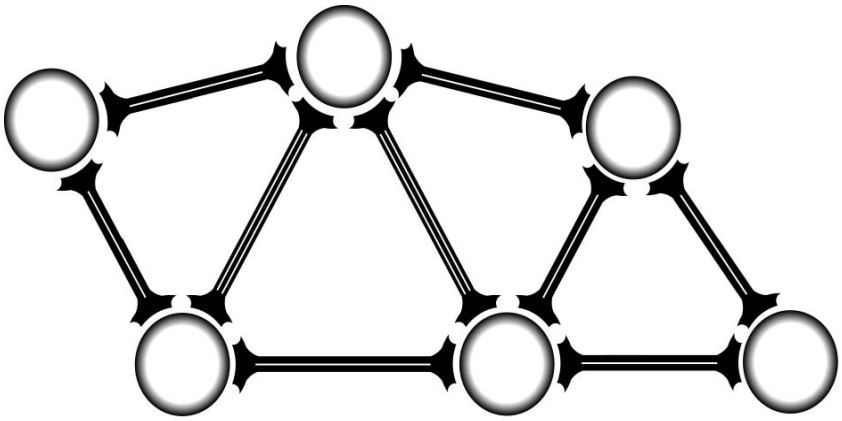

Figure 5 - Business Templates

Redressing one of the more striking methodological weaknesses seen in many SGML and XML implementations, the business template strategy replaces the focus on data elements with a focus on business objects that can be treated as manageable configuration items. As became the practice of the best implementers of structured markup technology, and there are many who have already travelled down this path, a more formal and more rigorous analytical approach is demanded when building business templates. The designers, in establishing a business object, define the data structure for the object as well as its behaviour. Furthermore, under the business template strategy, these business objects are situated in a larger process scenario where, on the analytical side, usage examples (Use

Cases) can be developed, and on the implementation side, initial operating capabilities can be developed to refine the behaviour of each object.

The idea behind with business templates is to make iterative development more prominent in the design of interchange protocols but it also goes further than this. The real intent behind business templates is to move briskly past "prototyping" and to move to implementation as soon as possible. So long as the investment in application components can be kept low and the tools that are deployed allow for rapid adjustment, then the idea is that the *real* business requirements will come to the surface when people are attempting to do their jobs with the new template in place. In an environment of continual and accelerating change, a development posture that attempts to divert energies away from production activities in order to "prototype" functionality always has a air of unreality to it - for the operational staff being involved and the developers. It is always a little like shadow boxing. Adopting a general strategy and technology toolkit that allows new business objects to be injected into production seems the only practical path ahead.

Under the rubric of the business template, the interchange protocol becomes a more robust collection of components. Just as the migration to schemas from the original DTD syntax provides more avenues to elaborating on the content being described, so the evolution to business templates requires that each "element" be associated with:

- documentation components suitable for presentation as part of business descriptions,
- requirements descriptions that set out the original business and technical requirements to be supported by the business object and the business rules that will be applied,
- processing and style specifications that have been modularized to correlate with the managed level of the schema,
- relationship descriptions including not only the identification of child and parent elements but also the business object classes from which it will inherit properties,
- application components that have been modularized to correlate with the managed level of the schema,

- sample data sets that are both functionally and technically complete and that provide content for use in testing and in demonstrating the functionality for management, users and maintainers, and

- metadata for use in managing the business object, making it available for reuse, and determining its applicability in different scenarios.

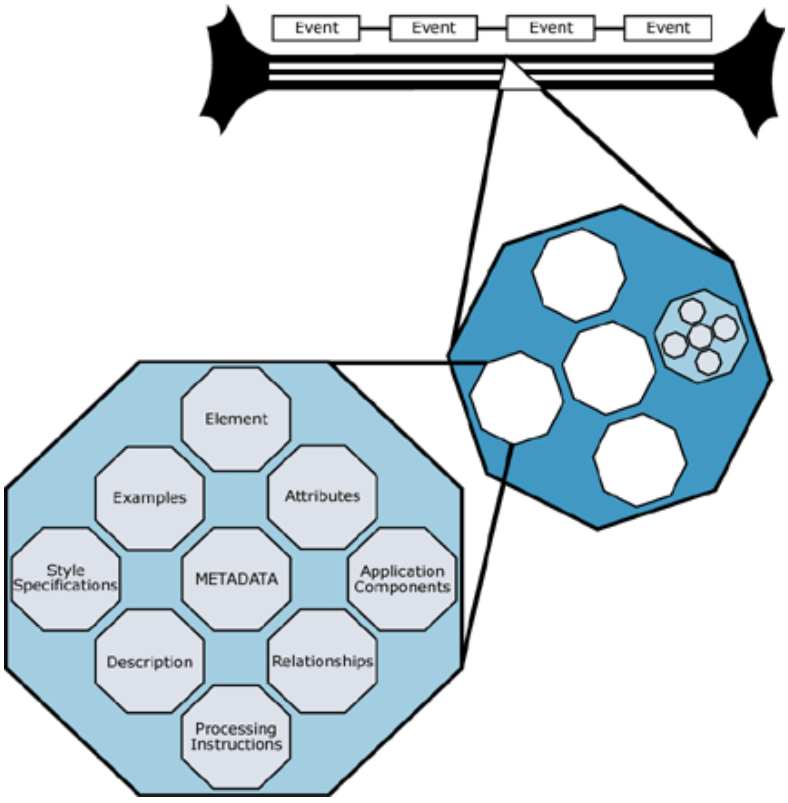

Figure 6 – Inside Business Templates

What is important about business templates lies in the emphasis it places on specifying and managing complete business objects, as opposed to simple data element definitions, and on moving to implementation as quickly as possible so that the real business requirements can be brought to bear on the development process.

In practice, business templates couple XML schemas with extended documentation and demonstration data sets and with application components that are as mainstream as they are simple to modify. The start up functionality of a business template is generally implemented using XML instances for all data sources to be processed, in lieu of introducing database tools into the initial

stages of the process, and leveraging scripting languages such as XSLT and JavaScript to implement initial HTML interfaces and web server executed events. Scalability is not the initial objective of the business template; functional completeness and tangibility, from a business perspective, is the goal.

With engaging business people who will execute real transactions, the team really starts to work - implementing changes and extensions in near real time. As the interchange protocol starts to take shape, the different participants in the interchange community can determine which shared applications should be developed if any and, more importantly, how they will each implement their own part of the process.

As the development effort on a business template comes to a close, the automated components that have been iterated throughout the process have arrived at a level of functional completeness so as to be feasible components of, or alternatives to, any additional levels of automation that may be introduced by the community or by an individual organization. The introduction of a capable database engine to facilitate transaction storage, indexing and retrieval might be a completely logical implementation step. It should not however be mandatory.

Within the realm of business templates, all technologies are viewed as tools that may load subsets of the data domain in order to implement enhanced capabilities or to accelerate a given event. The data, in XML, remains the *master copy* at all times and the availability of the base operating capability, provided by the initial automated components, reduces the status of any commercial product dramatically. And so it is that just as SGML grew in part to escape the tyranny of publishing products, so XML and business templates will evolve to put all technologies in their place - that of being an optional tool that is used to achieve specific benefits and then cast aside whenever it becomes an obstacle. With business templates, the original radicalism of SGML is taken to its logical conclusion.

As organizations move into a brave new world of e-business, in addition to facing the challenges of truly inter-operating with other organizations, they also come upon a tremendous opportunity to reassess how technology has been historically leveraged. Most, if not all, organizations have been burned at least once by falling for the "comprehensive automation" myth used to sell monolithic enterprise applications. Hopefully, XML and the business template strategy will spell the end of this form of "*chump-ware*" and will help organizations to better match their technology investments with the benefits they can realize.

THE ORIGINS OF BUSINESS TEMPLATES

The ideas that have been sketched out as **XML Business Templates** found their initial expression in a presentation given at the Continuous Acquisition and Lifecycle Support (CALS) conference of 1996 in a paper entitled "Coordinating SGML Project to Maximize Corporate Benefits". This paper outlined the lessons learned from a large collection of implementation projects where the benefits of the strategy being expounded were proven effective or the negative consequences of its absence demonstrated in graphic detail.

ONE IMPLEMENTATION EXPERIENCE

The delivery of this paper at XML 2000 was paralleled by a particularly instructive implementation project where the merits of the XML business template strategy were again underscored. In this particular case, the client organization was undertaking a large-scale integration effort that would combine the administrative and operational systems for a multi-national engineering consortium. This case was instructive because it illustrated some of the problems to be overcome when introducing concepts developed within one technology domain into domains dominated by what might be called more traditional application technology.

The integration of such a wide range of application components, encompassing financial accounting, customer relationship management and project control, was posing this customer a variety of challenges. When engineering content management was added to the portfolio of requirements, and with it the need to exchange content with a variety of partners and regulatory stakeholders, the complexity of the planned integration began to exceed the grasp of even the largest of the mainstream system integrators.

The customer was somewhat daring in that it introduced into its system integration consortium a firm that specialized in XML-enabled solutions. This team in turn introduced the notion of XML business templates as a way to elucidate requirements and facilitate their trial and validation. XML pathways would be specified for all exchanges and all content resources would similarly be associated with an XML expression that could be adopted in order to circumvent application incompatibilities.

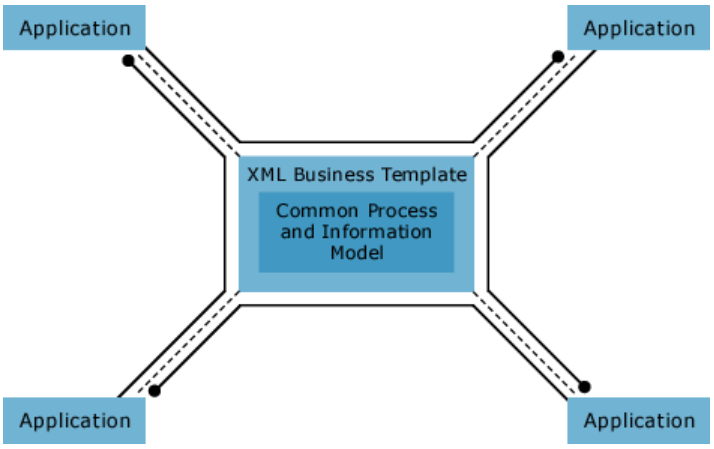

Figure 7 - Minimalist Integration Infrastructure

For the larger partners within the system integration team, this idea seemed unusual and even unprofessional, accustomed as they were to adopting (indeed selling) large-scale enterprise application licenses with price tags running into the tens of millions of dollars. The usual objections were raised about security, message handling integrity and performance. These players remained somewhat displeased when it was shown that these very topics were part of the alternative strategy although they were introduced as implementation enhancements to be addressed once the business process requirements were hammered out in practice using the XML business templates. Their opposition stemmed from two sources. First they were uncomfortable with the introduction of a new technology (XML) into the equation when they had minimal experience working with it. Second, they disliked the notion that the customer would direct their early investments towards elaborating their requirements and not sinking money into the software products they wanted to sell and with which their staff were comfortable working.

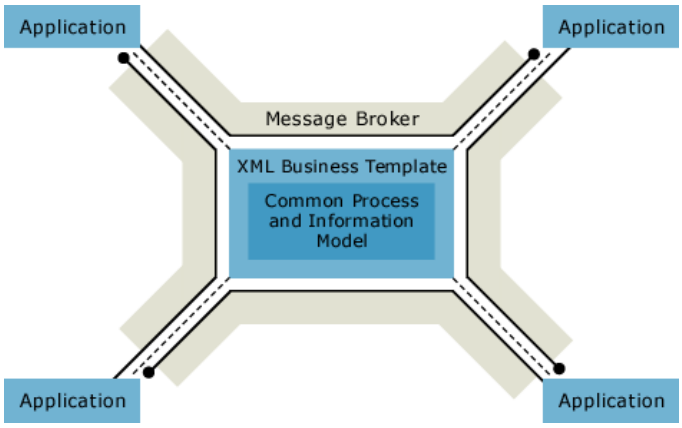

Figure 8 - Robust Integration Infrastructure

Figure 8 illustrates what became part of the evolving integration architecture for this customer. The message broker that one of the system integrators was determined to force into the project was shown to have a potential role, so long as it could be proven to work with open specifications for all messaging transactions. It turned out that the touted broker did not offer connectors to all of the applications that needed to be integrated so, ironically, the champions of the broker investment actually invoked the existence of the XML business template as a way to handle as-yet unsupported connections. This, it was hoped, would allow the early acquisition of the message broker to remain on the books.

This case is instructive as it showcases the point that the purpose of XML business templates is enabling organizations to maintain the flexibility and freedom of choice that should come with the openness and simplicity of XML. The objective is to ensure that organizations establish and maintain firm control over their technology infrastructure and retain the right to discard components that cease to serve their purpose or to adopt new ones that offer genuine benefits. This approach allows these organizations to postpone heavy financial outlays on application software until they have fully validated their business, and indeed technology, requirements. Only in this way will technology infrastructures start to exhibit *agility*. Of course, genuine agility is the last thing that most technology vendors would ever want to see enabled.

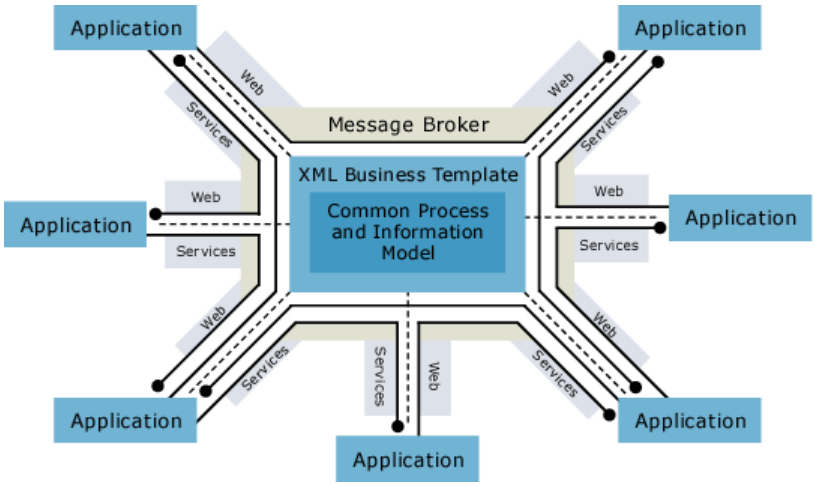

Figure 9 - XML Business Templates and SOA

Surveying the technology landscape in the more recent years (i.e., 2006), the attractiveness of the XML business template approach seems to be steadily improving with the widespread endorsement of a Services Oriented Architecture (SOA) wherein XML-enabled messaging stands as the universal interface between application components. Here, XML business templates continue to demonstrate their value in the exploration and refinement of knowledge assets being brokered and the business rules that govern their handling. Interestingly, most of the rhetoric heard about SOA does not focus on the specific benefits being promoted by XML business templates. It should come as no surprise that these benefits remain something that most, if not all, technology vendors would prefer their customers never experience.

In the implementation example being referenced, the tug-of-war between the XML business template strategy and the agendas of the large system integrators continued, on many fronts, for almost two years. In some battles, the XML business template strategy found powerful adherents among the business stakeholders and here it was used to very good effect. In other areas, however, the technology prejudices of the large integrators were echoed within the client, typically by technology managers fearing novelty above all else, and in these areas massive technology investments were then followed by the recriminations that sometimes attend major disasters.

