White Paper
August 2007
www.arjuna.com

Reliable IT for the Agile Enterprise

arjuna

middleware for reliability

Contents

Introduction

An Agile Enterprise is one that is able to thrive in today's complex, changing business environment, rapidly adapting in response to new market opportunities and customer requirements.It is recognised that an Agile Enterprise must be built on an agile IT infrastructure that can itself react quickly to changing demands; any delay in doing this can result in a loss of competitive advantage. However, the infrastructure must be robust as well as agile, so that it can be relied upon in the face of changing requirements and demands.

To add to CIOs' problems, these challenges have to be addressed in a climate in which there is great pressure to reduce the costs of IT by increasing utilisation. This is encouraging the deployment of IT infrastructures which attempt to meet these twin goals through the increased sharing and reuse of the key IT resources: software, hardware and data.

This paper gives an overview of the trends driving agile IT, and shows why supporting enterprise applications within a highly agile system raises difficult problems. In particular, increased, dynamic data sharing can have severe consequences for the reliability of the agile enterprise. Failure to address these problems will significantly restrict the ability of the new emerging infrastructures, such as Grid computing and Extreme Transaction Processing, to fully address the goals of agility and cost-reduction. Investment in solutions which do not support data sharing in enterprise applications will prevent enterprises from obtaining full value from their most valuable asset - their data.

Supporting the Agile Enterprise

Enterprises need to be agile in order to gain and sustain competitive advantage. A major trend for CIOs is the drive to IT infrastructures that can support the Agile Enterprise. We have identified three key types of agility that have implications for the underlying IT infrastructure:

- **Agile Business Infrastructure** – allows the enterprise to respond to changes in customer demand for their existing services and products.

- **Agile Business Information** - allows the Enterprise to extract maximum benefit from its data in order to rapidly reach informed decisions.Typical enterprises have a wealth of valuable information locked in silos that are distributed throughout the organisation. It is recognised that current business trends are escalating the need for enterprises to break down traditional barriers and share previously siloed data[1]. This is not just required within the enterprise - organisations now have to work with a variety of distributed partners due to global sourcing, virtual enterprises and outsourcing. Further, the growth in mergers and acquisitions, worth nearly a trillion dollars in the US alone in 2005, creates a need to combine previously separate data sources and applications. To meet these requirements, enterprises must have the ability to locate, access, integrate and process their data.

[1] Master Data Management: Extracting value from your most important intangible asset, Seth Halpern, SAP

- **Agile Business Processes and Agile Development Methodologies** – allow the enterprise to efficiently create new products and services in order to take advantage of new market opportunities.

An Agile Enterprise must be able to do all of these faster than, or at least as fast as, their competitors. This cannot be achieved without an IT infrastructure that supports agility across three key classes of IT resource: software, data and hardware:

Software agility refers to the deployment of software across the IT infrastructure, as and when it is required to meet key business metrics such as performance, cost and reliability. This is enabled by support for dynamic deployment, e.g. by Dynamic Resource Brokers[2].

Data agility refers to the efficient and dynamic sharing of data as required by the applications that operate on it. This is enabled by support for data exposed as services, dynamic data deployment, replication, caching and synchronisation.

Hardware agility refers to the way that the underlying hardware is used to support dynamic software and data deployment. This is enabled by using commodity hardware, virtualisation and standards-based management.

Therefore, agile IT combines and reuses software, data and hardware resources, as required to meet changing business demands. It does this by building on an underlying IT infrastructure that is capable of dynamically deploying and managing software and data.

Reusable IT

In recent years there has also been a noticeable trend within the enterprise to reduce costs associated with IT systems by improving utilisation levels. CIOs are under pressure to balance cost reduction against the continual drive to improve performance and have directed their efforts into making more efficient use of existing IT resources. In particular, the focus has been on reusing these three key resource classes:

Software reuse reduces the cost of developing and maintaining new software. This has been a driver for the widespread adoption of the Service Oriented Architecture (SOA) approach to building systems. SOA technologies, such as Web Services, allow applications to be constructed from reusable components which can be shared between applications. Industry standards have led to high levels of interoperability, so simplifying the process of combining services into applications. This has encouraged the recent growth in the use of workflow languages such as BPEL, which support the rapid integration of services through the provision of simple, visual design tools. Software reuse is therefore decreasing the time spent on developing and maintaining new software and is

[2] Extreme Transaction Processing: Technologies to Watch, Pezzini, Govekar, Natis and Scott, Gartner Core Research Note G00146107

helping to decrease the Total Cost of Ownership.

Data reuse encourages the exploitation of valuable information resources throughout the enterprise, while reducing the cost of maintaining and synchronising duplicated data. One effect of the move to SOA is to break down application silos, in which application logic and data are tightly coupled, into a set of services that can be combined in bespoke ways to build new applications (***Figure 1***). Exposing data as services in this way makes it easier to construct applications that share data, obviating the need for each application to manage its own copy.

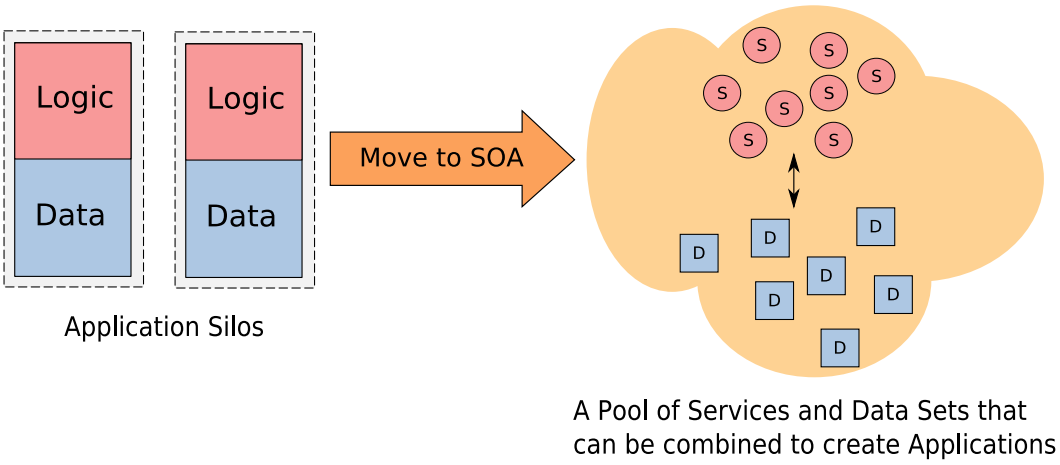

Figure 1. Adopting SOA breaks application silos into a set of services

Hardware reuse increases utilisation and reduces the number of separate boxes that must be purchased, maintained and managed. It is achieved by allowing a hardware resource to support a set of applications, rather than dedicating each resource to only one application.

Agility Increases Complexity

The trade-off for providing agility and greater resource utilisation is the increased complexity of the IT infrastructure. As previously discussed, one effect of the move to SOA is to break down application silos into a set of services that can be combined in bespoke ways to build applications. This makes it easier to construct applications that share both services and data. For example, ***Figure 2*** shows how extra value can be extracted from enterprise data by integrating two, previously separate, data resources.

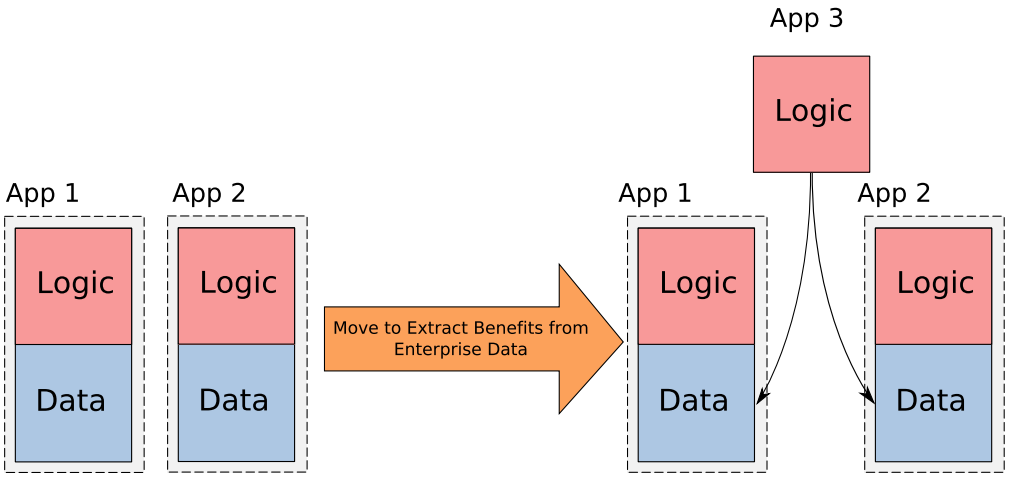

Figure 2. Integrating Data from Two Applications

However, as a result, data sharing is more common and its effects are harder to predict. It is much more difficult for a user to clearly understand the behaviour of a dynamically evolving IT infrastructure. This can result in severe problems, particularly if the data resources were originally designed to be accessed within a siloed environment, or if the subtleties of shared access were overlooked in the design of the infrastructure. For example, if there is a hardware failure then it may no longer be clear which applications are affected, nor how they can be recovered. In a siloed environment that is manually managed, applications affected by hardware failure can be clearly identified as they are bound to particular hardware resources. This is not the case with an agile infrastructure.

The move towards greater agility can therefore produce problems in the enterprise's IT infrastructure. Whilst it increases data sharing, it can also lead to a loss of knowledge and therefore control over the degree and nature of the sharing that is taking place. This exposes the enterprise to a number of risks which are addressed in the reminder of this whitepaper. As we will see in the next section, architectural trends designed to support agile computing are also exacerbating the situation.

Architectural Trends Increasing Data Sharing

Some enterprise applications have sufficiently low performance requirements that they can run on a single server. However, when performance requirements increase to the point where one server is unable to cope, more scalable solutions are needed. Typical examples are Compute Clusters and Three-Tier Systems which involve multiple machines collaborating to provide a single service.

In a typical Compute Cluster, a service is deployed onto a set of machines, and a load balancer distributes the stream of client requests across the machines in order to provide high performance.

This approach works well for applications that do not share data. However, for many key applications (e.g. e-commerce applications) sharing data is

a necessity. For these, Three (or more) Tier Architectures are common. In Three-Tier architectures, applications are divided into the *Presentation Tier*, *Application Tier* (also called the *Logic Tier* or *Business Logic Tier*) and *Data Tier*. The Presentation Tier runs on Clients (e.g. in browsers) generating requests that are sent to the Application Tier. For scalability, this tier usually runs on a cluster, with requests being load-balanced across the available machines. The data that application tier machines need in order to process the client requests is held in the Data Tier – often in database servers. Separating the application and data tiers allows them to be scaled independently. If there is an increase in usage of a service, more machines can be added to the Application Tier to handle the increased rate of client requests. This may in turn place extra load on the Data Tier, in which case it may have to be scaled by upgrading the database servers or fragmenting ("sharding") the data across a set of database servers. Another approach is to try to minimise the load on the Data Tier by main-memory caching in the Application Tier. Some recent products allow these caches to be managed as a single, shared resource across all the nodes in the application layer. These distributed caches provide a degree of fault-tolerance (by replicating data across the nodes) as well as improved performance (by reducing the frequency of accesses to the databases in the Data Tier)[3] .

Although clusters and Three-Tier systems provide scalability, they may not be cost effective. The reason is that it is typical for each application to run on its own set of hardware resources, which must be capable of handling the maximum load. However, the standard load is likely to be much lower than the maximum and so for most of the time, resources are highly underutilized. This approach is therefore safe but inefficient.

In an attempt to address this, Grid solutions are being introduced. These group a set of machines, from one or more clusters, into a shared pool and allow one or more applications to use resources from the pool as required to meet the changing demands placed on them. For example, an e-commerce application that was subject to an increase in usage could expand across more machines, contracting again if and when demand lessened. If different applications' performance requirements rise and fall at different times then having them share a pool of resources is more efficient than providing each application with its own dedicated set of machines that is sufficiently large to meet peak demand. Further, Grids offer a way to smoothly integrate new applications that add value to the company. This avoids the need for a large initial capital outlay, which often has to be based on crude estimates of demand.

Initially, Grid solutions were developed within organisations ("internal Grids"), largely by combining existing machines. However, more recently, hosting companies have offered external Grids that remove the need for organisations to manage their own hardware resources - instead, they access them over the Internet. The Amazon EC2 service is a pioneering example of this.

The logical next step will be to federate internal and external Grids and allow work to move seamlessly between them. This will enable, for example, applications to run entirely on an internal Grid within an organization during times of normal

[3] Extreme Transaction Processing: Technologies to Watch, Pezzini, Govekar, Natis and Scott, Gartner Core Research Note G00146107

load, but to expand to incorporate resources from an external Grid during times of maximum load or system failure.

Therefore, while the rise of SOA has resulted in more data sharing between applications, the move to n-Tier and Grid architectures has produced a second dimension of data sharing – sharing within applications whose components may now be distributed across a cluster, or even beyond (in the case of federated Grids).

Data Sharing and its Risks

In this section we delve further into the nature of data sharing in an agile IT system, before looking at the risks this can pose to the enterprise.

In a siloed enterprise, each piece of data is encapsulated in its own data management system, and the application itself can control access to the data. In contrast, data in an agile enterprise can be shared, replicated and distributed on-demand around the IT infrastructure and there is no single application or data management system to control access to it. Indeed, it is often very difficult even to identify which services and applications are using a data resource. This is known as the *Hidden Sharing* problem, and is one key aspect of what has become widely known as *SOA Chaos* – the loss of control and understanding that can result when the SOA vision is implemented in an enterprise.

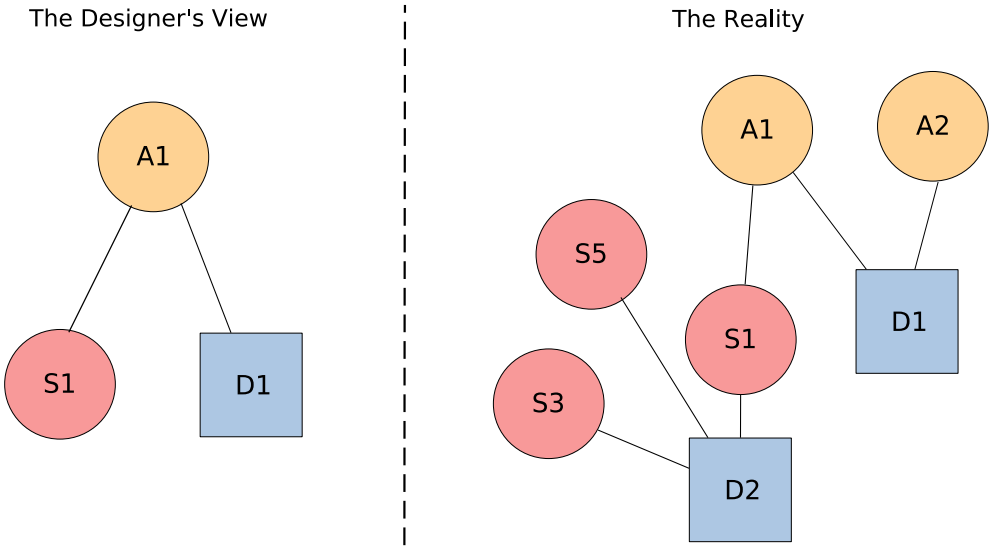

Figure 3. Hidden Data Sharing: (left) the designer's view; (right) the reality

For example, a designer builds a new application A1 that uses service S1 and data set D1 (***Figure 3***/left). As far as the designer is concerned, the application sends requests to S1 and D1 and receives responses from them. Because of the encapsulation offered by services she has no idea of what other services S1 calls, nor what data sets they access. Therefore, she does not know that S1 accesses a data set D2 that is already used by other services S3 and S5. Further, data set D1 was already being used by one other application A2, but

neither she, nor the designer of A2 know that it is now shared (***Figure 3***/ right).

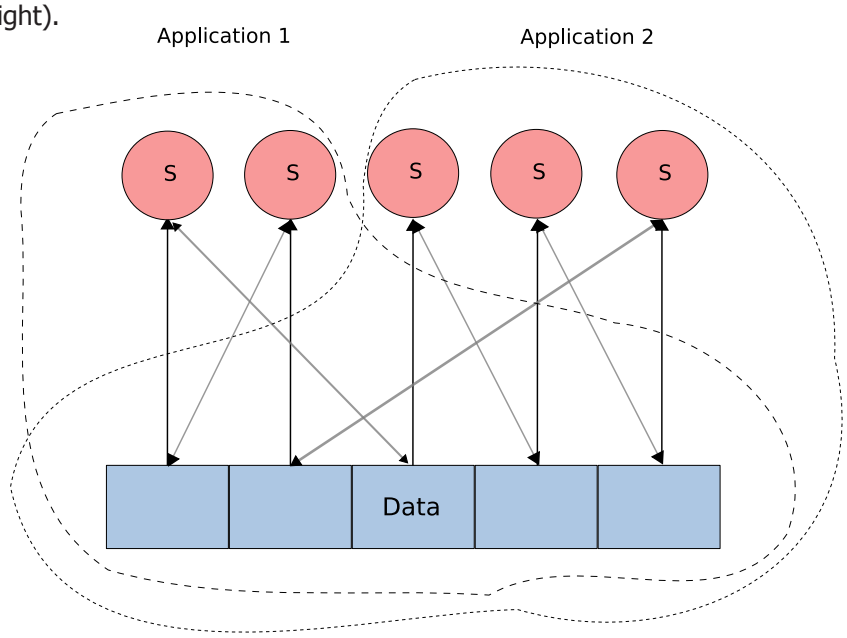

Figure 4. Two Applications Sharing Data

As a further example, ***Figure 4*** shows two applications sharing data. A danger here is that *Application 1* may have been designed and built to operate on its own siloed data. Deployed on a Grid, its designers would have identified no need for techniques to control sharing. Later however, *Application 2* is written and deployed to operate on the same data. If *Application 1* is not modified so that sharing between these two applications is controlled by mechanisms such as transactions then severe problems will occur. In some cases, an application will require access to data in one or more databases, in which case more advanced techniques such as distributed transaction coordination may be required.

In the next section we examine the nature and effects of the problems created by sharing in more detail.

Coherency and Consistency Problems

When two or more computations share data, there are two concerns regarding the correctness of that data set: coherency and consistency.

A data set is strictly coherent if each access is strictly ordered in real-time i.e. each computation sees every access as it occurred in real-time. Less-strict coherency can be acceptable for certain computations e.g. a serialised ordering in which each computation sees the same view even though the accesses might not be in the same order as they occurred in real-time.

Figure 5 illustrates two client services S1 and S2 accessing the same data set. If the accesses are always seen by both S1 and S2 in the same order as that

shown, then the data set is strictly coherent. An example of a requirement for strictly coherent data might be an application which is monitoring, in real-time, fluctuating prices. Here, the exact price movements and direction are important to the application's success.

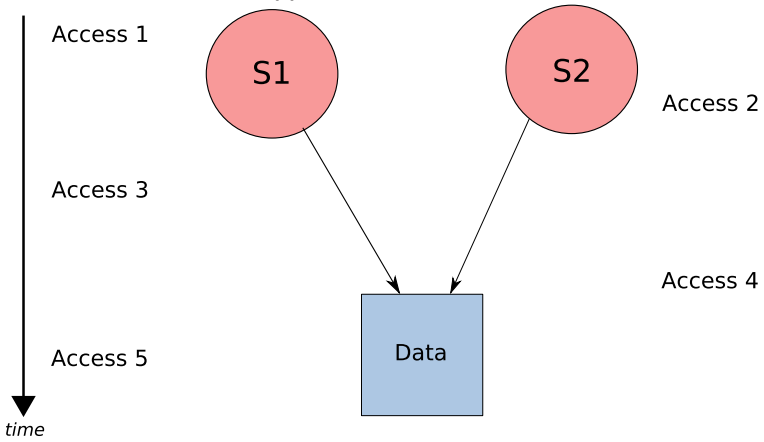

Figure 5. Coherency Example

Coherency is complicated by the requirement for internal consistency of the data set. Support for consistency is required because some views onto a data set, although coherent, may be 'invalid' (with respect to some pre-defined set of invariants). These potentially invalid views are created by a partial update of the data as a computation moves it, in a series of steps, from one consistent state to another. An inability to protect other computations from the existence of these inconsistent states may cause errors. Therefore, the existence of inconsistent states needs to be communicated so that the system can make informed decisions. For example, the system might provide locking to isolate data sets which are in inconsistent states, or it might allow other computations to proceed so long as they have knowledge of the inconsistency and can take care of potential problems.

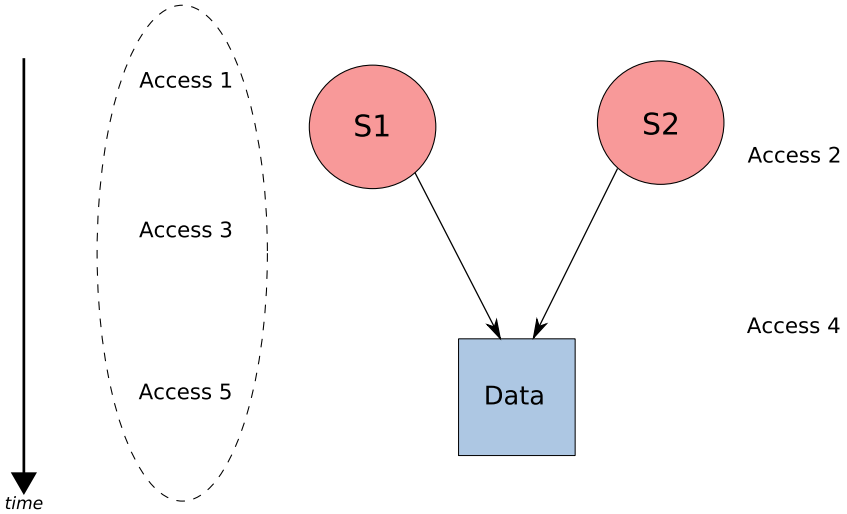

Figure 6. Consistency Example

Figure 6 illustrates two client services S1 and S2 accessing the same data

set. S1 needs to make a number of changes to the data set to move it from one consistent state to another. The changes required by S1 are delimited in the example by dotted lines that denote a 'scope' within which partial changes may temporarily leave the data set in an inconsistent state. The system needs to know about the inconsistent state so that it can take appropriate action. For example S2's accesses 2 and 4 which interleave with S1's accesses in real-time could be denied, delayed, deliberately ignored or compensated for. A simple example of a data set with consistency requirements might be a set of address lines associated with a customer. The visibility of partial changes to the address when the customer moves premises might lead to parcels being misdirected. A more complex example would be the accounts of a large corporation whose books must always move, through a series of intermediate steps, from one state in which they balance to another in which they also balance. A failure to prevent other applications accessing the data in this inconsistent state could have dire consequences, for example if the accounts were to be audited at this point.

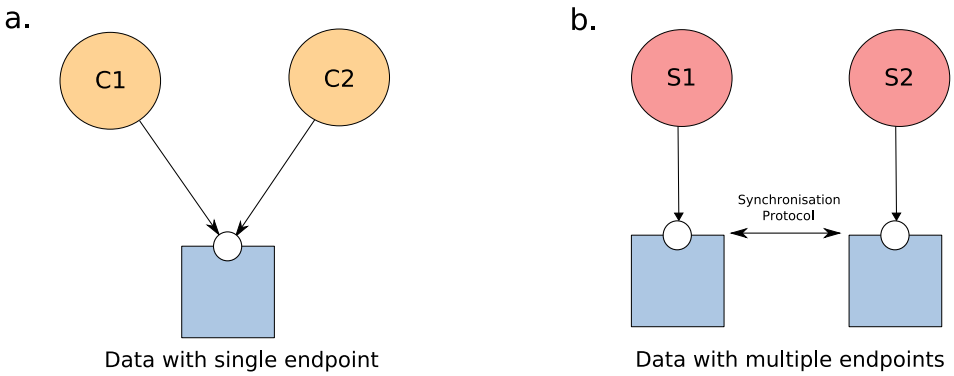

Figure 7. Synchronisation Example

Figure 7(a) illustrates the fact that when the data set is accessible via a single endpoint then coherency and consistency are both relatively easy to support as the endpoint can synchronise access. However this may result in performance problems in the agile enterprise if the endpoint becomes a bottleneck, for example because an increasing number of services are accessing the data. Further, if the computations are running remotely from the data in a distributed, agile infrastructure then latency may create problems.

To solve these performance problems, in ***Figure 7***(b) the data set has been distributed through replication, caching or partition, or some mixture of both. This makes the data accessible via multiple endpoints. However, addressing the performance problems introduces new issues. Synchronisation must be introduced to ensure coherent or consistent views, and this requires a protocol which communicates between the distributed parties.

Reliability Problems

Another manifestation of the sharing problem arises when there is a failure. Reliability is a key for most enterprise applications. An agile infrastructure has

the potential to achieve this through redundancy: if one node fails then the remaining nodes can carry on the work. However, the degree of data sharing affects how complex it is to achieve this. If there is no sharing then a failed computation can simply be re-run. Many existing Grid solutions offer this level of fault tolerance. However, if data is shared, then it is imperative that the effects of failure are contained so that inconsistent, partially updated state is not allowed to permeate through applications. Having infrastructure that knows when data is in an inconsistent state can be of great benefit when a failure occurs. If a computation has not completed its work at the point when a failure occurs, it may leave data in an inconsistent state. However, a system with the appropriate support could either 'rollback' the state to a previously consistent state (backward recovery), or 'compensate' to create a new consistent state (forward recovery). Without this, there is the real danger of corrupting the data on which the enterprise relies.

Even determining which applications and data sets have been affected by a failure can be a problem in an agile infrastructure. Dynamically deploying software, data replicas and caches in order to meet varying demands means that if there is a hardware failure then it may not be clear which applications are affected nor how they can be recovered. Recovery can be even more complicated if applications are widely distributed across resources that are managed under different regimes. The danger is that there are may be no consistent, infrastructure-wide mechanisms for detecting failure, reporting failure, deciding what action to take and performing recovery in a co-ordinated manner.

Conclusions

Data sharing is at the heart of many of today's key enterprise applications. Important business drivers are pushing an increase in data sharing through Service Oriented Architectures and the need to extract more value from the distributed data spread around the enterprise. In this whitepaper we have shown how data sharing can lead to problems that need to be addressed in order to preserve the integrity and reliability of the data on which enterprises are now so heavily dependent.

CIOs are under pressure both to re-architect their infrastructures to be more agile through the use of SOA, and at the same time to consolidate their IT infrastructure to maximize the utilisation of existing resources. This has led many enterprises to explore the use of Grid and related technologies such as virtualisation. These technologies support the dynamic deployment of resources across the IT infrastructure and so offer the opportunity for sharing to avoid underutilisation and to meet the challenges of dynamically changing loads. As a result, they can remove the inefficiencies created by existing siloed solutions in which each application is tightly bound to a different subset of the enterprise infrastructure. However, there is a downside as agile IT reduces the knowledge and control that enterprises have over when and where data sharing occurs. This can lead to severe problems, including corrupted data being propagated throughout the enterprise.

Without solutions to these problems of data sharing, Grids and other forms of

agile IT will not be able to support key enterprise applications. This will prevent enterprises from fully exploiting the benefits of consolidating IT resources. Instead, new silos will be created, with key enterprise applications separated from those that are able to run safely on Grids. The result will be continued IT over-provisioning and an inability to react in an agile but robust manner to changes in demand for enterprise applications (**Figure 8**). It will also prevent the enterprise from responding rapidly to new business demands that require data to be shared across those applications.

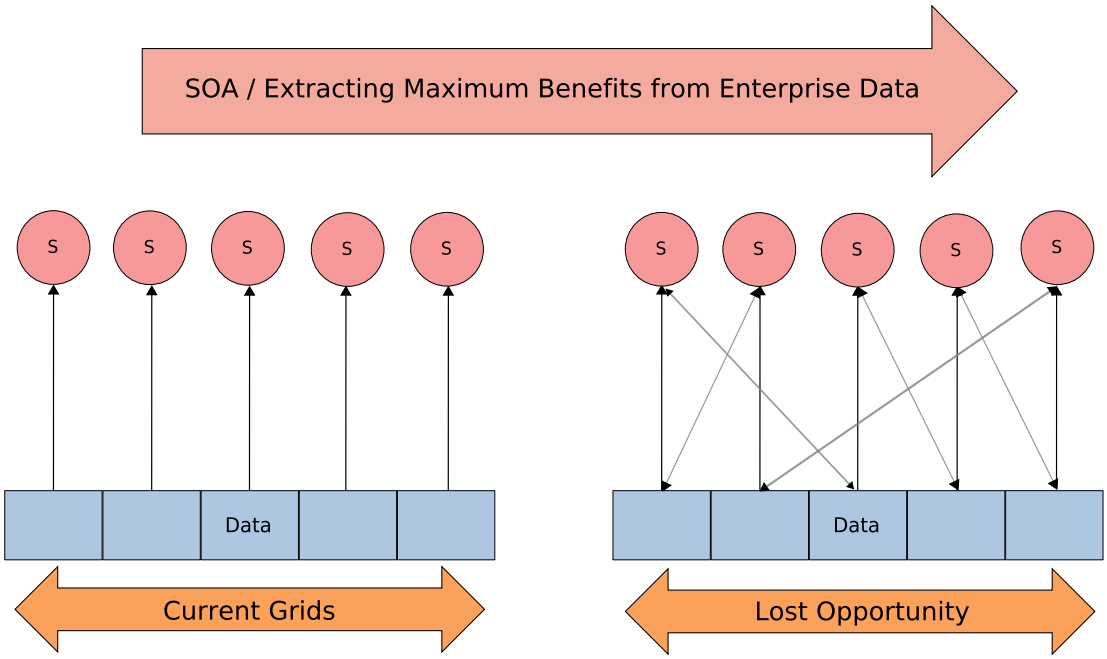

Figure 8. The inability to support enterprise applications that share data represents a lost opportunity

Therefore, a key question for Grid providers is how to evolve their infrastructures to embrace transactional enterprise applications. New solutions are required to achieve this. Based on our experiences in managing more static forms of data sharing in enterprise applications over the past two decades, we believe that these new infrastructures must be able to:

- Identify data sharing and protect applications from the consequences: **fault-prevention**
- Monitor data sharing: **fault-detection**
- Record data changes to aid recovery: **fault-recovery**

However, we have not yet seen these features in commercial Grid infrastructures. Many Compute Grid infrastructures do offer fault-recovery through restarting the applications, but this is not sufficient for enterprise applications that share data. Many Data Grid solutions focus on data coherency without providing scoping mechanisms for data consistency, but this is not sufficient for enterprise applications and could result in inconsistent enterprise data being exposed to other applications and so propagated through the organisation, with potentially

disastrous consequences.

It is therefore essential that CIOs, IT managers and buyers are aware of the potential problems that may affect their enterprise applications as they move towards a more agile IT infrastructure. Enterprises should consider these issues when evaluating the proposed solutions offered by Grid technology vendors. We believe that no current offering addresses all of the problems we have highlighted in this paper. Consequently, we are currently prototyping a solution that embodies our ideas on how to support reliable data sharing in an agile infrastructure.

About Arjuna

Based in Newcastle, England, Arjuna Technologies Ltd is a Centre of Excellence in reliable distributed systems. Arjuna has been active in the research, development, production and support of reliable systems for over twenty years. In 1997, it released the world's first CORBA Object Transaction Service followed shortly after by the world's first commercially-available Java Transaction Service. In 2002, as HP Arjuna, it released the world's first Web Services Transactions product. Today, Arjuna is working on applying its ideas on reliability and fault-tolerance to the Grid.

Arjuna Technologies Ltd: Floor A, Milburn House, Dean Street, Newcastle upon Tyne, NE1 1LE
United Kingdom