

Contacts:

Clive Longbottom
Quocirca Ltd
Tel +44 118 948 3360
Clive.longbottom@quocirca.com

Dennis Szubert
Quocirca Ltd
Tel +44 845 397 3660
Dennis.szubert@quocirca.com

Functional Re-use and SOA

The Service Interface as an Enabler for Software Re-use Best Practice

Executive Summary

Service-Oriented Architecture (SOA) provides a foundation to achieve software reuse. A key part of the SOA value proposition is the benefits realised from software reuse.

Main Findings

- **Re-invention is a major cost to businesses**
Many organisations continue to buy or re-create similar, if not identical, functionality across different applications. The management of such separated function can result in sub-optimal business process support, and additional costs to the organisation.
- **The drive for rationalisation means that similar functions must be combined**
As organisations strive for infrastructure rationalisation, functional rationalisation should be a large part of this. The "culling" of similar functionality across multiple applications cuts down on overall complexity and provides greater business flexibility.
- **Discrete functional components create greater flexibility**
By providing a service infrastructure, functionality can be provided as discrete components, callable by those processes requiring such function.
- **Discrete functional components are more scalable and resilient**
Through the provision of discrete functional components via an SOA, scalability can be provided through simple utility-style provisioning, and resilience can be managed through the use of stateless services.
- **Software re-use speeds up response to market threats**
The capacity to make use of discrete functions that have already been proven, and are simple to include within new composite applications means that solutions are faster to create, and organisations can be more flexible in how they create strategies to meet market dynamics.
- **Granularity of function is the main issue that has to be addressed**
When looking at functional components, a suitable level of granularity must be maintained. Major software vendors are attempting to position their offerings as being SOA-compliant, but the levels of granularity means that this argument is often a moot point. For true software re-usability, the level of function must be optimised to serve as many different processes as possible.
- **Regression testing can be optimised through software re-use**
A carefully optimised SOA environment will ensure that an upgrade to any specific function will automatically provide that optimised functionality to all processes calling it, catering to all dependencies

Conclusion

SOA is a driving force for future functional use within organisations. However, these functions must be viewed as being shared services for all processes. Functional re-use will be the main means of ensuring that organisations can respond rapidly and effectively to market dynamics, and that improvements to specific functions will have the optimum impact across the whole organisation.

An independent report by Quocirca Ltd.

www.quocirca.com

Commissioned by IBM

quocirca

1 The Need for Functional Re-use

Businesses in fast moving markets need the capability to respond rapidly to changing market conditions, and flexible IT systems are crucial in enabling the business to respond quickly and cost effectively. This is a major business motivator for reuse, where existing functionality can be leveraged by multiple processes, and new processes can be facilitated by the building of composite applications using an amalgamation of existing functions. Monolithic applications do not help in this regard as they provide little capability to respond to rapid changes - each new process requires new coding, as well as hard connections both within and across applications. However, with a set of services that provide common business functions useful across all the different applications for developers and users to take advantage of, new applications can be created or current applications extended rapidly.

The traditional monolithic application development approach also means that identical (or at least similar) functionality gets replicated numerous times across applications in the enterprise. This has an impact on management, overall cost of licensing, and even computing capacity requirements, and as more emphasis is placed on the utilisation of technology within the organisation, we are fast approaching the position where the real estate, power and cooling requirements of even a basic infrastructure becomes a significant part of the IT budget. Not only is such reinvention costly in terms of time and effort, but perhaps more significantly, also is prone to introducing errors that could impact the organisation's capacity to transact business. Single instances of functionality are inherently more manageable, as reusable services are not only "build once" but also "maintain once".

Historically, there has been little in-built capability for other applications or processes to make use of the functionality embedded within existing applications – highly expensive, often proprietary approaches for systems integration have been required, and any changes to one part of an infrastructure has generally led to the failure of the overall process as incompatibilities have been found. However, with the advent of web services and service oriented architectures (SOAs), we are now in a better position to look at how we can move to wards a more reusable approach to function, and so create a more responsive, flexible, manageable system that utilises more of the hardware assets we have at our disposal.

2 Background to Software Re-use

Software re-use is not a new concept. Although it did not enter mainstream software application development until much later, object-oriented programming has its origins in the Simula programming language of the 1960s, and the idea of software components was first espoused at a NATO conference on software engineering in 1968. Many vendors have tried to provide their own means of managing functional components in the past, generally through coding environments. Digital Equipment Corporation (DEC) had the Distributed Directory Service (DDS), where libraries of code could be stored, and vendors such as Borland, Rational and so forth provided the means of storing uncompiled code items that could be brought in to new code as required. However, these systems were highly vendor specific, with little capacity for interoperability, and still generally required a code compilation prior to any usage. Similarly, Microsoft's DCOM (Distributed Component Object Model), introduced in 1993 to enable interprocess communication in any programming language that supported it, was a proprietary platform that worked only on Windows.

The first real attempt at a fully independent capability for code reuse was with the Common Object Request Broker Architecture (CORBA) in the 1990s, which attempted to create a standardised environment for the exchange of information between discrete items of functionality. A standard defined by the Object Management Group (OMG), CORBA enables software components written in multiple computer languages and running on multiple computers to interoperate. Unfortunately, CORBA adoption was piecemeal, and the standards could not easily be retro-fitted into existing environments. While it promised to deliver much in the way code is written and software is constructed, CORBA failed to deliver on some of its promised features, sometimes even the most fundamental ones. Some of its failure can be tracked down to the way in which CORBA was created as a standard, and the lack of other widely adopted standards that it could build off, but some reflect more

fundamental problems in the design, and others yet stem from the very principles on which CORBA was built. These problems have led to a significant decline in CORBA usage in recent years. On the other hand, some CORBA implementations became very successful in certain verticals such as telecoms, aerospace, finance and automotive, and many of CORBA's underlying principals and technical approaches made their way into subsequent technologies, which can now be brought together within an SOA environment.

Although SOA as a concept is essentially technology independent, today's understanding of SOA is built from work carried out through the late 1990s and early 2000s into web services. These services, built as discrete pieces of code, revolved around the use of specific standards, such as the Simple Object Access Protocol (SOAP), the Universal Description, Discovery and Integration (UDDI) repository and others to create flexible environments where code could be reused. However, problems with UDDI scalability and with organisations wanting to ensure rapid deployment of solutions meant that Web Service based solutions were often just replacing hard-linked applications with hard linked services.

Today, SOA has broad software and systems integrator support, and is built around the use of de facto standards, such as the use of web services as an underpinning, of J2EE and .NET as coding environments, and of scalable repositories for service identification and control. However, SOA is not just around new code – there is a strong need within organisations to ensure that investments in existing IT assets are maximised, and there is tooling available in the market to make functionality held within existing enterprise applications available as callable functions within the SOA environment itself.

3 SOA and Functional Components

A SOA environment is the ideal place for code re-use. SOA implementations are seen by the majority of vendors as the way forward, and standardisation of approach is widely adopted. Therefore, a piece of function created for one process can be called by any other process that needs that function.

An SOA environment also creates the basis for greater resilience and scalability. Discrete functions can be relatively easily provisioned on the infrastructure to provide greater scalability as required, and multiple instances of the same function can provide the resilience required to maintain function through point technical failures. Indeed, the use of discrete functional components enables these to be served as software as a service (SaaS), where the function can be hosted externally and called as required. Other benefits of this approach include the lack of need to own and manage the infrastructure to support the function as the hosting company takes on that task, the extra levels of resilience and security that can be provided by external hosting companies, and access to alternative business models that can be based on subscription, transaction load, tiered usage or whatever. Also, SOA provides the means for integration across the firewall, an area that has been a major problem in the past for those companies looking at the original Application Service Provider (ASP) model in the 1990s.

If we look at the way an SOA should work, we start with an initial component that initiates a process or task. This component may not have a great deal of capability in itself – it may only provide a graphical interface to the user, or it may be a process stub aimed at initiating a machine-to-machine transaction. This initiator knows that it needs an action to be carried out, and makes a request via a repository for such a function to be made available. The repository identifies what functions are available, matches the best function to the needs of the initiator and makes the connection between the initiator and the servicing components. This servicing component may not be able to complete the function itself, and so that would then become the initiating component, and would again request the repository to identify and make the connection to the best function for completion. This process continues until the needs of the overall business process are completed.

As can be seen, SOA requires a different approach to that which has been utilised historically. The application centric approach, where organisations bought a specific, monolithic solution to meet its needs in a specific area often led to a lack of flexibility, and to the issue that

Quocirca has researched where many companies are spending up to 80% of their IT budgets on systems maintenance and support, rather than on investing in differentiating systems and processes that support the business goals.

SOA can provide a support environment where the business is placed firmly back in control. However, there are areas that need to be borne in mind – areas such as component management, in user friendliness, in functional granularity and so on. These areas will be covered later in this document.

4 Existing Applications and Functional Re-use

A complete replacement of existing infrastructures to adopt SOA is generally not viable, and existing applications must be looked at as to the role they will play in the new environment. Large monolithic enterprise applications will need to be examined as to what processes they are supporting and what functions are being replicated across the applications. For example, we may find that we have multiple sources of customer information, and yet we really need to ensure that any process requiring customer data makes a call to a single data source where we know the information will always be up to date. There may also be multiple billing engines, making it difficult to show a customer or a supplier what the state of their account is at any one moment.

What is needed is to ensure that existing applications can be abstracted to make functionality visible and available so other processes can access it in a simple manner. By “wrapping” existing applications so that built-in functionality can be abstracted to appear as discrete functions, additional flexibility is provided – we get to choose which redundant functionality should be dropped, and which functionality is best of class. We can then use the chosen functionality to ensure better effectiveness of function – all customer data changes can be “owned” by one function, any business reporting can be carried out through one reporting engine, and so on. We can also gain better efficiencies – process flows can be dealt with through more common routes, and optimisation of these routes can be more easily dealt with.

To do this, a greater level of functional abstraction than is generally available in existing applications is needed. This can be done through tools that enable the mapping existing process flows across and within existing applications, and we can also create SOA entry points into existing applications through the use of connectors and adaptors, running across an enterprise service bus to provide the enterprise capabilities needed.

This ability to encapsulate and expose existing application functionality makes the SOA approach an attractive option for modernising tried and trusted legacy systems, as opposed to the more radical alternatives of rip and replace, or lift and shift (i.e. porting onto another platform).

5 Granularity of Function

When we look at breaking down a business process, we end up with a set of business tasks, each one of which will need one or more technical functions or technical “services” to facilitate it. For example, we could have a business process which is “turn a prospect into a customer”. A task within this process could be “close a sale with a customer”, or we could have one which is “get customer’s credit card details”. The first example would require a set of technical functions to complete – from knowing who the customer is, what they are attempting to buy, whether we have it in stock, what the price is, and so on. The second example is far more targeted, and could just be a case of getting a long string of digits, a card end date, and a security number. Indeed, we could split this down further to a business task that is just “get customer’s credit card number”.

The problem is to ensure that we hit the right level of granularity. Too high a level of granularity, and a process finds itself bouncing backwards and forwards as it calls different services all the time, and the resulting response times becomes incredibly slow. If we go for too low a level of granularity, such that the granularity is more at the process level than at the task level, we move more towards the hard-wired existing application approach – we get high

response levels for the standard processes, but exceptions are difficult to manage and any change to the standard process takes a lot of coding time and retro-testing time.

By mapping out business processes and seeing what commonality of functions there are within specific functions, we can begin to choose the right levels of granularity for our services. For example, with a credit card transaction, we have a certain approach irrespective of card type – whether it be via a debit or accredit card. We need to know the card number, we need to know the end date (and possibly the start date), the name on the card, and then there may be other information that we want for validation purposes, such as issue number and cardholder address details. This then means that we have a simple, defined set of rules that can be created as a defined service, and then any process that has a task that requires credit card details can call this one service. If we need to change the way that we deal with credit card details (for example, if we need to support chip and PIN in some way), we then only have to change the single function – and all processes reliant on getting credit card details will now have this functionality available to them.

We must also look at the various types of function – there will be some that can be common across an organisation's complete value chain (e.g. customer details), whereas some may only be common across our own enterprise (e.g. a whole customer record), or a single department (e.g. regulatory requirements in pharmaceutical laboratory work). By understanding these different types, we can begin to prioritise our approach to functional re-use.

Quocirca advises that a three-fold approach is taken to this. Firstly, identify those functions that are common throughout the value chain. Focusing on these functions will have the maximum value to the business – commonality of function through the value chain will lead to the capability for massive automation savings and will result in fewer transcription errors, as data will no longer need to be manually transcribed from one system to another. Secondly, there is a need to look at whether we are in a highly regulated environment – if so, then concentrating on the common functions within the regulatory framework will provide the greatest paybacks, whereas for those in a less regulated environment, concentrating on organisational common functions will provide the greatest payback. Thirdly, the capability to capture lower-level granularity where it is seen to be reused on a regular basis will help. Here, we are looking at what groupings of reusable functions/services are being used, and whether it is possible to create a “super component”, where such a grouping can be pulled together and optimised as a single entity, rather than as several separate entities.

By gaining the right balance between function and granularity, redundancy of function will be more easily identified within the existing technology environment. Existing applications can be tuned to support the functions required, and redundant functionality can be turned off as applicable. This will lead to better overall infrastructural efficiencies, and will free up resources to support the important functions identified.

6 The Role of “Best Function”

Historically, the argument has been between single-vendor solutions and best-of-breed approaches to the provision of a solution. Quocirca proposes that we now look at solutions from a “best function” approach – what functions do we already have in place to support our business process needs, what functions can we abstract from existing applications, and what functions will we need to bring in from elsewhere (either through the purchasing of discrete functional components from a software vendor, or through the creation of bespoke functional components).

Here, we see the main strengths of reuse. Reuse does not necessarily come down to the creation of new functionality in a way that makes reuse easy: it also involves utilising technologies that can make existing functionality buried in existing applications available to other systems.

Often, organisations will find that 60-70% of the functionality required is already available within their existing systems. For example, as previously discussed, the presence of a billing

engine within an existing enterprise application is often seen as being particular to that application. Through understanding the business logic underlying the application, combined with the use of suitable connectors, this functionality can be made visible to other processes requiring a billing engine.

Going further, we may find that we have more than one billing engine in the organisation, as many existing applications will have their own dedicated billing engine. By evaluating the effectiveness of each of these engines, we can then choose that which provides the best function, and make that our default billing engine going forwards.

In some cases, we may then find that any single solution so uncovered does not meet our requirements going forwards. Here, we need to look at whether we can make a specific solution meet our requirements through the use of additional callable functionality that we can create as web services, or even using functionality from an alternate existing solution.

Only where such an approach is seen to be of such complexity that response times will be strongly hit, or where the complexity of the end solution is such that maintenance and support will be difficult should we look to the wholesale replacement of the function with newly-written code.

7 Utilising Functional Components in an Organisation

In today's dynamic markets, organisations must be far more responsive to market pressures. The current dependencies on going back to technologists every time changes are required at a technical level means that windows of opportunity are often missed. The use of large, monolithic applications has meant that many organisations have had to change their business processes to fit in with the approach of the application. This is no longer a valid option, and the way that an infrastructure is architected now has to reflect that an organisation must be in control of its own processes, and that the organisation will need to change these processes on a regular basis to be able to successfully compete in the market.

Functional components move an organisation towards this goal. We need to look at ways in which the business itself can utilise functional components to build up composite solutions to immediate business process issues – and to be able to do this to meet exceptions as well as standard processes. Therefore, we need to be able to make functional components visible as business assets, and provide the basis for business people to be able to string these functions together in a simple manner in order to facilitate their process needs.

Firstly, any functional components available for reuse need to be identifiable. To maintain such an asset base of functional components, we will need a registry of some type. The registry must be capable of holding information on those functions that have been made visible from existing applications, those functions that have been created specifically for our needs, and those functions that have been bought in off the shelf. This registry should be able to take calls from components requiring a service, and match these calls with functions available from its knowledge of what is available. The registry does not store the components itself – it just maintains links of what the components are, what the components are capable of doing, and where they are. As we move increasingly towards a more utility view of an infrastructure, this knowledge of where a function is at any time becomes far more important, as the function may be being provisioned and de-provisioned across a virtualised environment.

We also need a way of enabling users to rapidly create a composite solution. However, we also need to ensure that we adhere to the needs of governance, audit and regulation, and that wherever possible, existing composite solutions are utilised that have already been proven and checked for compliance. For the majority of organisations, giving free rein to every user to build composite solutions does not make sense. Large parts of the employee base will be task oriented, and will not need to have such flexibility provided to them. It is far better to start with those decision makers who can have the biggest positive impact on the

organisation's business through being provided with such flexibility. The sort of tooling that they will require will be of a visual, workflow orientated nature – being able to replicate existing process flows rapidly and then being able to carry out “what if?” scenarios to see what changes would provide the optimal overall solution. The tools should then be able to create the underlying “glue” that will bring the discrete functions together to facilitate the business process, through the use of an enterprise service bus, specific connectors and an understanding of the requisite business logic context.

Finally, we need to ensure that everything is audited. Today's applications have the benefit of being able to replicate transactions – the in-built rigidity of the approach means that changes are difficult to make, and a process that was carried out once will likely be carried out the same way for a long period of time. With composite solutions, however, we need to ensure that a process that may only have been used once is fully tracked so that we can demonstrate exactly what was done at that point when required.

7.1 Reuse in retail and commerce

As an example of reuse, we can look to the retail sector. Over the past few years, retail has had to move from a pure bricks and mortar play, to a multi-channel, multi-offer environment. Those who were first movers faced significant problems – their existing solutions were often home-grown or proprietary, skills in creating new web-centric shop fronts were scarce, new applications were prescriptive in their approach. However, the underlying processes for the retailers remained essentially the same.

For example, the main aim for a retailer is to sell an item to someone. An offer has to be made, either via a shop, or via the web (or by any other means for example a mail shot). The prospect who is interested in that offer has to be captured and the sale completed. Within this, we need to make sure that we have the item in stock, or advise at what point it will be available, make alternative offers, try and up-sell and so on – tasks that are the same no matter what channel we come in through. The explosion of multiple possible channels means that we have the opportunity to gain more customers – and also to lose them very easily to a more agile competitor.

Therefore, we can see that generally, a retailer will have the majority of the function it requires already in place to facilitate these tasks. The problem has been that the new eCommerce application tended to replicate all of this function, in the interests of being a “complete” solution. Now, we have multiple customer records, so that a customer who has ordered via the web is not seen as an existing customer when they enter the shop. Credit checks may need to be replicated – and an existing shop customer can find themselves being denied an account due to insufficient history through the web site. Integration to back-office systems have to be done twice – even though the end result should always be the same.

By utilising the existing technical functions to support the new channels, today's retailers are finding that the benefits are huge – changes to offers are reflected across all the channels, offers made to customers are seen everywhere. Processes can be simplified and optimised, without the various channels getting out of sync, and leading to poor customer satisfaction.

The problem is only going to get worse – no sooner have we come to grips with a standard eCommerce channel, than we have to start looking at possible new channels, such as the possibility of alternative web stores (e.g. in Second Life), in different means of customer contact, such as instant messaging, IP telephony, kiosks and so on.

To ensure that we do maintain synchronicity between all the channels, it is important to ensure any new channels can be easily plugged in to the existing environment, and for this, we need to look for a platform that will co-exist seamlessly with the chosen service bus architecture we have in place, and that can optimise the reuse of the functions we already have in place.

8 What to look for when considering code/function reuse in an SOA environment

When looking to maximise organisational effectiveness and efficiencies through code and functional re-use within an SOA environment, there are many aspects that should be borne in mind:

- Standards – A discrete piece of functionality that has been created without looking to the support of open standards will not provide any benefit to the organisation. Functional components must be created to be open for use by any calling process.
- Interoperability – Similar to standards, functional components must be able to understand each other and to interoperate. For example, a called function may well need to call an additional function to deliver a solution back to the owning process.
- Manageability – A means of discovering and managing functional components is required. This should show all process dependencies, so that any risk associated in the changing of a single function can be easily seen
- Business usability – Functional re-use can only be seen to provide the greatest value to a business where the greatest flexibility can be demonstrated. A means of using process modelling tools and matching existing functional components with process needs will drive added value.

9 Customer Example

Panasonic, the manufacturer of consumer, business and industrial products had moved into multi-channel distribution, and had implemented multiple disconnected systems to meet what was seen as new functional requirements for its multiple audiences (e.g. business, consumer and channel). With hundreds of thousands of items across its brand portfolio, attempting to manage this across multiple systems was not viable in the long term.

9.1 Customer's business issue

Panasonic needed to put in place an efficient, cost-effective eCommerce solution that would meet its immediate and ongoing needs within its distribution chain. Panasonic's 20,000 dealers were already supported on an integrated SAP/IBM WebSphere Application Server system, whereas its own employees and the employees of its largest, favoured business customers had access to eCommerce shops via systems created with Microsoft Commerce Server and IIS. Integration between these consumer eCommerce sites and the back end SAP systems had proved problematic, and multiple instances of equivalent functionality was leading to high costs of maintenance.

9.2 Customer's approach to solving business issue

Panasonic decided to move its entire eCommerce strategy over to IBM's WebSphere Commerce and the WebSphere Application Server platform, providing a more manageable environment that met its immediate requirements while providing the development and connectivity flexibility for the future. Also, the development environment within WebSphere, based around the J2EE standard, provides for easy reuse of developed functional code. Further, the common approach to the technology allows for reuse of existing and new backend functionality – for example, all new B2B sites being set up now utilise the same catalogue data from a single DB2 database.

9.3 Customer's perceived benefits from implemented solution

Eric Keil, Director of eBusiness for Panasonic, states that the reusability of code under WebSphere has been a major factor in time to capability and in cost savings for Panasonic. He says "Because of the reusable Java code supported by WebSphere Application Server, we've reduced the development time for new Web sites from three months to one week, saving 92 percent of the time and costs".

Further, by moving to an architecture that utilises open standards, Panasonic has now moved to a document and message exchange system across its value chains, where XML is utilised to provide required information to, and to receive orders from their dealers.

10 Conclusion

SOA provides a real direction forward where organisations can look to regain control of their own destiny. However, there is no place for wholesale rip and replace or lift and shift within today's businesses, and future flexibility has to be catered for. The reuse of existing technology investments has to be a major consideration, and defining and uncovering the right levels of functionality within existing applications is a main design point.

Also, the need to ensure that new functionality is created as reusable services will help in ensuring that the future needs of the company can be suitably covered – the aggregation of existing services into a composite solution will be far more rapid and effective than trying to hard-link monolithic applications or write brand new code each time there is a major change in the markets.

Although there are business issues around managing the plethora of services that may result from going for a full SOA environment, ensuring that a solid basic architectural design is chosen before beginning implementation will ensure that such issues are minimised. Similarly, possible issues around end user capabilities for creating composite applications on the fly need to be catered for, with the choice of suitable tooling and for audit capabilities to show how transactions and interactions occurred within such an environment.

SOA is the way forward – those organisations getting SOA right will be the winners in tomorrow's dynamic markets. Ensuring that existing functionality can be reused, and that new functionality is written with reuse in mind, will help organisations be in the leading group.

About Quocirca

Quocirca is a company that carries out world-wide perceptual research and analysis covering the business impact of information technology and communications (ITC). Its analyst team is made up of real-world practitioners with first hand experience of ITC delivery who continuously research and track the industry in the following key areas:

- Business Process Evolution and Enablement
- Enterprise Applications and Integration
- Communications, Collaboration and Mobility
- Infrastructure and IT Systems Management
- Utility Computing and Delivery of IT as a Service
- IT Delivery Channels and Practices
- IT Investment Activity, Behaviour and Planning
- Public sector technology adoption and issues

Through researching perceptions, Quocirca uncovers the real hurdles to technology adoption – the personal and political aspects of a company's environment and the pressures of the need for demonstrable business value in any implementation. This capability to uncover and report back on the end-user perceptions in the market enables Quocirca to advise on the realities of technology adoption, not the promises.

Quocirca research is always pragmatic, business orientated and conducted in the context of the bigger picture. ITC has the ability to transform businesses and the processes that drive them, but often fails to do so. Quocirca's mission is to help organisations improve their success rate in process enablement through the adoption of the correct technologies at the correct time.

Quocirca has a pro-active primary research programme, regularly polling users, purchasers and resellers of ITC products and services on the issues of the day. Over time, Quocirca has built a picture of long term investment trends, providing invaluable information for the whole of the ITC community.

Quocirca works with global and local providers of ITC products and services to help them deliver on the promise that ITC holds for business. Quocirca's clients include Oracle, Microsoft, IBM, Dell, T-Mobile, Vodafone, EMC, Symantec and Cisco, along with other large and medium sized vendors, service providers and more specialist firms.

Sponsorship of specific studies by such organisations allows much of Quocirca's research to be placed into the public domain.

Quocirca's independent culture and the real-world experience of Quocirca's analysts, however, ensure that our research and analysis is always objective, accurate, actionable and challenging.

Quocirca reports are freely available to everyone and may be requested via www.quocirca.com.

Contact:

Quocirca Ltd
Mountbatten House
Fairacres
Windsor
Berkshire
SL4 4LE
United Kingdom
Tel +44 1753 754 838