# Utility Computing Architecture Supporting Service Oriented Architectures

**Badri Sriraman**
**Lead IT Architect**
**(Unisys)**

**Rakesh Radhakrishnan**
**Enterprise IT Architect**
**(Sun Microsystems)**

**Ramaswamy Rangarajan**
**Principle Network Systems Designer**
**(Sprint)**

**April, 2005**



# Table of Contents

## Introduction and Overview

So far in this series of papers on Service Oriented Architectures the authors have highlighted the significance of Model Driven Architectures, Component Based Architecture and Event Driven Architectures to Service Oriented Architectures, in conjunction with Identity enabled Networks. This paper attempts to highlight the significance and value proposition of leveraging the fundamental principle's and concepts behind "Utility Compute Architectures" (UCA) for "Service Oriented Architectures" (SOA), for XML based Services. This paper highlights the fact that UCA is an excellent fit for Services built on SOA – and UCA, CBA, EDA, MDA, SOA –all approaches compliment and support User Centric Agile Solutions.

In ***Utility Compute Architectures (UCA),*** computing is accomplished via a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate. The utility model seeks to maximize the efficient use of resources and/or minimize associated costs. Here the word utility is used to make an analogy to other services, such as electrical power, that seek to meet fluctuating customer needs, and charge for the resources based on usage rather than on a flat-rate basis. This approach, sometimes known as pay-per-use or metered services is becoming increasingly common in enterprise computing and is sometimes used for the consumer market as well, for Internet service, Web site access, file sharing, and other applications. It should be noted that another version of utility computing is carried out within an enterprise. In a shared pool utility model, an enterprise centralizes its computing resources to serve a larger number of users without unnecessary redundancy, where the IT division charges the rest of the Enterprise division for its services. From a technology perspective UCA leverages advances in Grid Computing (dynamic grids), service provisioning technologies, consolidation (networks/compute/storage) and auto-discovery mechanisms (such as JINI).

In ***Service Oriented Architectures (SOA)***, an Enterprise's Architecture is developed in a Service Driven approach, where each service is autonomous and can be considered to be a Service Building Block (SBB). Using an analogy between the concept of service and a business process, in SOA, loosely coupled SBB are orchestrated into business processes that support customer goals and/or organization's business goals. More than just a component of reusable code, a SBB becomes part of a running program that can be invoked by a client without having to incorporate the code itself. A SBB by definition is reusable and replaceable, that is, one SBB's service is reused again and again by other services for the functionality it provides and SBB's service (provider implementation) can be replaced by another (another providers implementation). In SOA – the SBB's themselves can be categorized into basic/foundation services, management services, security services, business services, portal services, etc. It should also be noted that a SBB is offering a specific functionality for an Enterprise and transcends projects, i.e., a Digital Rights Management Service (DRM) as a SBB is only implemented once in an enterprise architecture and can be reused across projects that deal with delivering content, services, multi-media, etc. In SOA the communication flow is closed to new unforeseen inputs once the communication flow has started, i.e., they are typically well defined and the boundaries well set.

***Component Based Architecture,*** the basis for distributed component architecture embraces mechanisms and techniques for developing coarse yet reusable business/technical implementation units that is environment/container aware, which decomposes a service of SBB, into multiple plug-able and distributable parts associated with presentation logic, business logic, resource access logic, integration logic, network event logic, security logic and more. A component based approach evolves from object oriented design principles (encapsulation, polymorphism, inheritance, etc.), and leverages a foundation set of infrastructure technologies (based on J2EE/J2ME/JAIN or .NET/OSA/Parlay) for seamless integration. Component Based Architecture (CBA) fundamentally modularizes large scale (monolithic) systems into multiple coarse-grained, durable and reusable technical units that could be implemented by multiple vendors yet integrated into a larger enterprise system. CBA typically engenders designing the internals of these coarse-grained, business-aligned component boundaries, through finer-grained object-orientation. Lately the capabilities to instrument components with service contracts around security and SLA's, has made component based design even more powerful.

***Event Driven Architecture,*** embraces mechanisms for coordinating the callers and providers of service, producers and consumers of data, sensors and responders of software events with variable level of communication coupling, with variable spectrum of message correlation and with variable options to deliver quality of service. EDA engenders a network that listens to thousands of incoming events from different sources, wherein complex event processing results in intended system response. EDA supports dynamic, parallel, asynchronous flows of messages and hence reacts to external inputs that can be unpredictable in nature. For example EDA approach can enable a caller invoke a provider SBB using events without knowing who provides it or what address the provider uses, what choice amongst multiple providers exists, while load balancing across them and selecting amongst these providers with varying levels of qualities of service based on the caller's requirements. Meanwhile the same software event that represents the request or the events generated by the service in response can be of interest to other SBB in the network, opening a channel for value-add to the customer and business. An EDA can coordinate synchronously or asynchronously between software endpoints, and possibly provide both synchronous and asynchronous access between the same participants. In EDA, simultaneous streams of execution can run independently of each other to fulfill a customer request or system responsibility, typically an event bus serves as a platform to manage integration and/or choreograph a larger process.

In ***Model Driven Architecture***, modeling an enterprises system (data, systems, and model of your data model) in conjunction with meta-data, which keeps a record of an enterprise's architecture in-terms of data, information, application, services, technology and implementation (platform specifics) is the basis. There are three basic tools used in an MDA as defined by OMG (Object Management Group) – (see http://www.omg.org/mda ): ***Meta-Object Facility (MOF)-*** The MOF defines a standard repository for meta-models and, therefore, models (since a meta-model is just a special case of a model)**. XML Meta-Data Interchange (XMI) -** XMI defines an XML-based interchange format for UML meta-models and models; by standardizing XML document formats and DTD (document type definitions). In so doing, it also defines a mapping from UML to XML. **Common Warehouse Meta-model (CWM) -** The CWM standardizes a complete, comprehensive meta-model that enables data mining across database boundaries at an enterprise and goes well beyond. Like a UML profile but in data space instead of application space, it forms the MDA mapping to database schemas.

This paper is the last in a series (on SOA), that includes;

- Model Driven Architecture enabling Service Oriented Architectures
- Event Driven Architecture augmenting Service Oriented Architectures
- Identity Centric Architectures (IDEN) extending Service Oriented Architectures
- Component Based Architecture supplementing Service Oriented Architectures
- Utility Compute Architectures supporting Service Oriented Architectures

All five papers describe the power in SOA when other conceptual architecture patterns associated with MDA, EDA, CBA and UCA are blended with SOA. With an identity enabled end-to-end environment, it adds more meaning to the notion of Service on Demand and True Service Mobility – anywhere, anytime, any network, and any device. The first paper highlighted how MDA enables SOA, following that the second paper highlighted how EDA, in conjunction with MDA, augments SOA. Along the same lines the notion of end-to-end Identity enabled Environment, was discussed, to highlight the Core Service Building Block that enhances Service Oriented Architectures. The previous (fourth) paper addressed the value-proposition and supplementing capabilities brought to SOA by CBA, in conjunction with EDA and MDA. This paper will highlight the alignment of UCA in support of SOA and how CBA, MDA and EDA with IDEN pave the way to do so.

## Value Proposition of Utility Computing (UCA) for Services (SOA):

Utility Computing Architectures go beyond traditional GRID Computing Architectures, wherein, web services concepts and technologies are also leveraged for GRID Services (based on OGSA- Open Grid Services Architecture, EGA- Enterprise Grid Alliance, etc.). In the past, GRID computing solutions where meant for Services that where compute intensive and grid-able (i.e., workloads such as batch jobs that where distributable to multiple compute nodes) such as;

- Derivative Analysis (Monte Carlo)
- Protein Modeling (Computational Research)
- Reservoir Simulation
- Electronic Design Automation (Simulation and Analysis)
- Movie Production (and 3D Visualization) Rendering)

However, lately (with the advent of XML and Java) Services in a SOA built based on CBA are distributable in nature making common business services (CRM, SCM, ERP and more) as Grid-able (and granular) Services. In a paper titled “Building Next Generation Grid-Enabled Business applications with Today’s Technologies”, the authors, Victoria Livschitz (Sun), Stacey Farias (Sun) and Seth Polansky (Visa), explain how not only 1) Job-oriented applications (e.g., billing reconciliation jobs) and 2) Batch applications (e.g., reporting on Data warehouse) are suitable for Grid computing environments, but also, 3) Transactional applications (e.g., Supply chain) and 4) Session-driven applications (e.g., shopping cart at Amazon) are also a good fit for Grid computing environments making all types of business applications Grid-able. According to the authors these 4 types are further clarified in the paper as;

- Job-oriented applications – These are traditionally the most common type of grid applications, found far more often in the realm of scientific and advanced engineering computing than business computing. The term “job” stands for a unit of computation that can be carried out independently of other computations. This term in general is used to describe parallel processing environments.
- Batch applications – Batch applications constitute a very important part of business applications. Request for computation are typically accumulated in a “batch” for future processing, where transactions are atomic, and long in duration (typically minutes), key operational metric is the length in the transaction window.
- Transactional applications – Transaction applications typically have requests for computation originate from external clients triggered from event outside of applications control, each transaction is processed individually and immediately (typically as threads), transactions are atomic and short in duration, load may fluctuate heavily between peak and average conditions, key operational metric is throughput –transactions per second.
- Session-driven applications – These are interactive business applications that are often designed around the notion of a “user session” which the relationship between a user and applications, across many transactions. From a grid computing standpoint even though session management and transaction processing are handled by separate application components, the design issues are largely uniform.

Also today’s Grid’s leverage advances in technologies that consolidate and virtualizes resources (compute/network and storage) and automates the provisioning of services to virtualized resources. These solutions simplify IT operations by allowing you to manage your data center as if it were a single system. Using virtualization and automation technologies, these Grids’s takes operational efficiency to new heights. Both hardware and software provisioning enables administrators to effectively manage growing numbers of network services across complex IT infrastructures and also optimizes resource utilization through dynamic workload distribution and fine-grained partitioning of individual servers. By bringing crucial abstraction and automation into the data center, Grid’s can help lower TCO and improve productivity, service delivery, and corporate “agility” that offers a critical competitive edge. These Grid solutions are built on a common set of architectural principles, setting the stage for the delivery of a single Grid interface for managing an entire enterprise and beyond.

There are many types of classifications of Grid Computing;

From a usage perspective they are classified into;

**Compute Grids** enable you to turn a set of independent systems into a single compute resource that can be employed by large numbers of users to run jobs of many types.
**Data Grids** consist of distributed storage devices along with the necessary software to provide transparent, remote, and secure access to data wherever and whenever it is needed.
**Access Grids** provide a secure, anytime, anywhere connection to compute and data grids -- yielding a single Web-based point of delivery for services, content, and complex applications (a.k.a. Identity Grids).

From a deployment perspective they are classified into;

**Enterprise Grid** that are deployed within an Enterprise/Organization for Inter departmental sharing (e.g., Home Depot and Coke as enterprises deploy GRID computing for enterprise applications and services).
**Partner Grid** that are deployed within multiple Enterprises across multiple departments as well (e.g., Fannie Mare deploying a Grid computing environment in conjunction with other partner financial institutions)
**Service Grid** that are deployed using the Utility model supported by xSPs (2003+), wherein the xSP, could also be hosting web services based on SOA (typically these are telecom companies that are hosting services such as sales force automation, payroll services, etc.).

There are many aspects to the evolution of Utility Computing especially when considering how today's Grid Computing Solutions leverages concepts behind SOA, MDA, EDA and CBA. By leveraging these concepts the alignment of SOA to UCA is improved extensively.

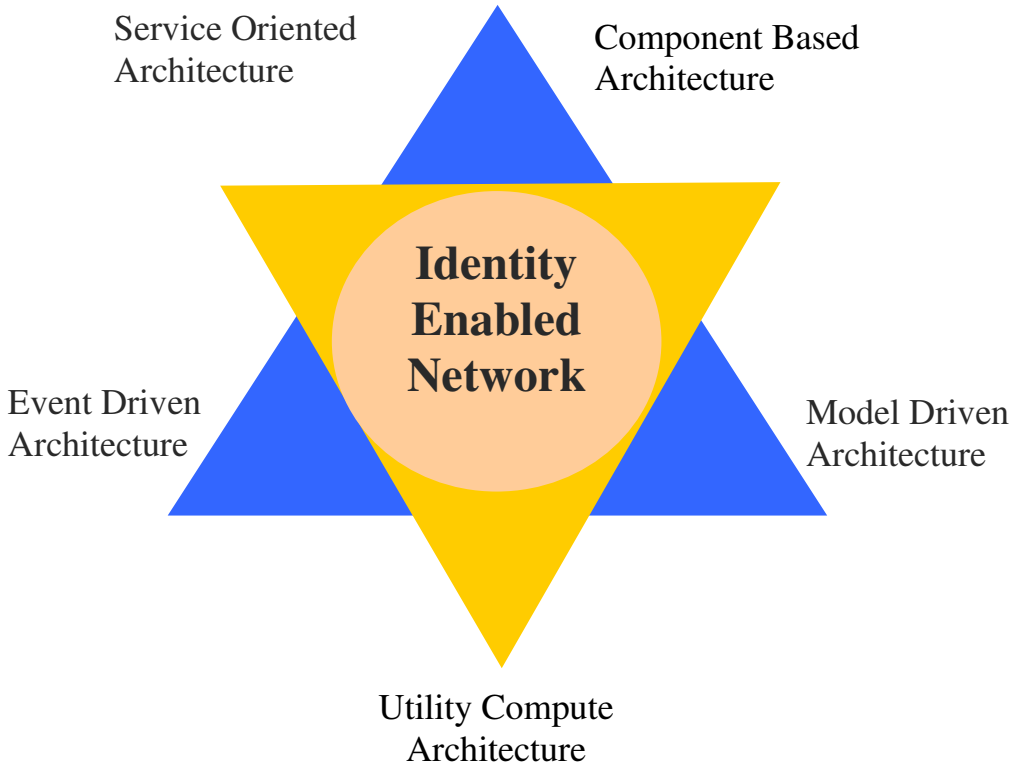


**Figure 1: SOA combined with EDA, CBA and MDA on UCA**

There are several perspectives from different architectural stakeholder viewpoints where utility based computing add value to services, and they are;

- ***Agility of Technology Infrastructure for highly fluctuating Service Requirements (UCA for SOA)***
- ***Auto-discovery, auto-recovery and auto-provisioning for Grid Elasticity (CBA for UCA)***
- ***Event driven Container Alignment for responsive and proactive grid computing (EDA for UCA)***
- ***Modeling resource abstraction for intelligent (heterogeneous) resource provisioning (MDA for UCA)***
- ***Services built as threaded and instrumented Components lends itself to Utility Grids (SOA for UCA)***

## Agility of Technology Infrastructure for highly fluctuating Service Requirements (UCA for SOA)

This perspective acknowledges that XML based "web" Services within an SOA typically do not have a defined set of systemic quality requirements, such as maximum number of concurrent users, transactions per second, etc. The Service's systemic quality or non-functional requirements vary extensively and are highly fluctuating. Traditionally when a HRMS system or a Payroll system is deployed by an Enterprise it is within defined boundaries. For example, a total of 2000 HR employees will access these systems with a maximum concurrent usage of 20% within the Corporate WAN, between 8:00am and 8:00pm PST everyday (except the 3rd Sunday of every month). However when the same payroll system is exposed as a "web service" accessible over the net by any user capable of discovering the service and any other "web service" that requires the specific functionality, then it possess challenges in terms of infrastructure resources. In the previous case with defined pre-set requirements, the infrastructure (compute/storage and network resources) for the system is derived based on the requirements and dedicated for such a service. However, with highly fluctuating requirements the technology infrastructure for "web services" is mandated to be agile, with the capability to dynamically, in an automated manner, provision and de-provision resources based on need. This makes Utility Compute solutions that leverage the 3 key features/advancements in GRID computing a perfect fit for web services.

- All 3 (compute, data and access) GRID can be deployed for a service
- Services themselves are distributable components that are heavily model-driven and instrumented
- Event oriented alignment of Service requirements to Resource allocation

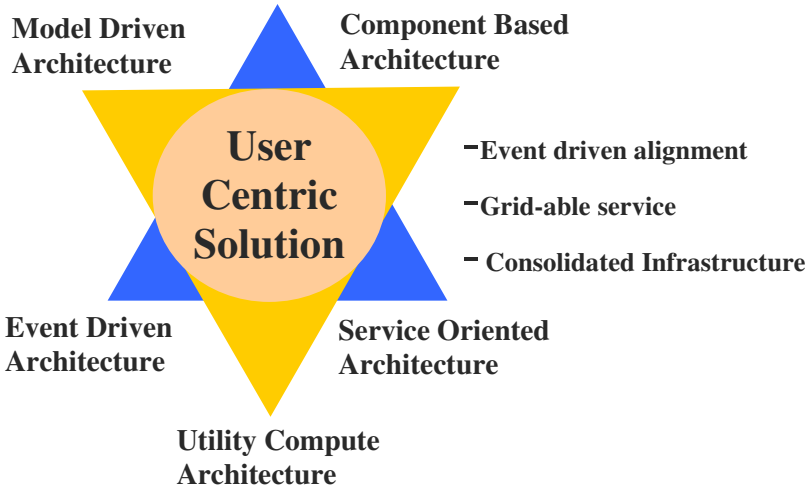


Figure 2: **Alignment of SOA to UCA**

From a CIO/CTO perspective this is a strategically significant synergy that not only reduces the costs and TCO associated with applications and services that an Enterprise needs, it also addresses the TCO associated with Technology Infrastructure required to support these services. Additionally while SOA addresses agility in the Business Services layer, UCA addresses agility in the Infrastructure services layer. The combination of SOA supported by UCA offering true end to end "Corporate Agility".

### Auto-discovery, auto-recovery and auto-provisioning for Grid Elasticity (CBA for UCA)

This perspective presents, advances made in GRID computing that makes it more conducive to run web services, in the form of auto-discovery, auto-recovery and auto-provisioning. Taking into account the code mobility aspect of components and its impact on service elasticity, in the background, one sees a need for intelligent alignment of resources to such services to ensure elasticity at the Infrastructure level. For e.g., if within a GRID environment there is a sudden spike in a security sensitive service that requires high-throughput at node level (meaning the end point where the data gets decrypted is also the node that runs the logic associated with the service data – data is unencrypted only in memory) encryption and decryption, this might trigger a event to discover the availability and allocate/provision more crypto accelerators hardware components to such a service for that duration in time. This requires additional infrastructure services within a GRID that can detect such a need and discover the availability of such resources and provision them automatically. Similarly dynamic changes to the storage environments might call for allocating more solid-date cache for specific services during specific scenarios, etc. This implies that infrastructure level resources (such as solid state cache, crypto accelerators, NIC's, cpu resources, and more) are dynamically added and removed from a GRID environment, discovered by services and used by them. Component based technologies such as JINI Services add these capabilities to GRID computing by making traditional GRID computing more a dynamic GRID, that is also self healing- another characteristic of elasticity (meaning hardware component level failures do not impact service availability).

Resiliency as a characteristic is tightly coupled with auto-discovery and auto-provisioning, in the form of auto-recovery. With the introduction of a state repository (that maintains the state associated with all connections, sessions, transactions, event and messages within a GRID), recovery of the service is made possible even when hardware component failures occur. Recoverability is addressed within GRID's via recoverable file systems, recoverable storage, persistence maintenance, cache synchronization, and more.

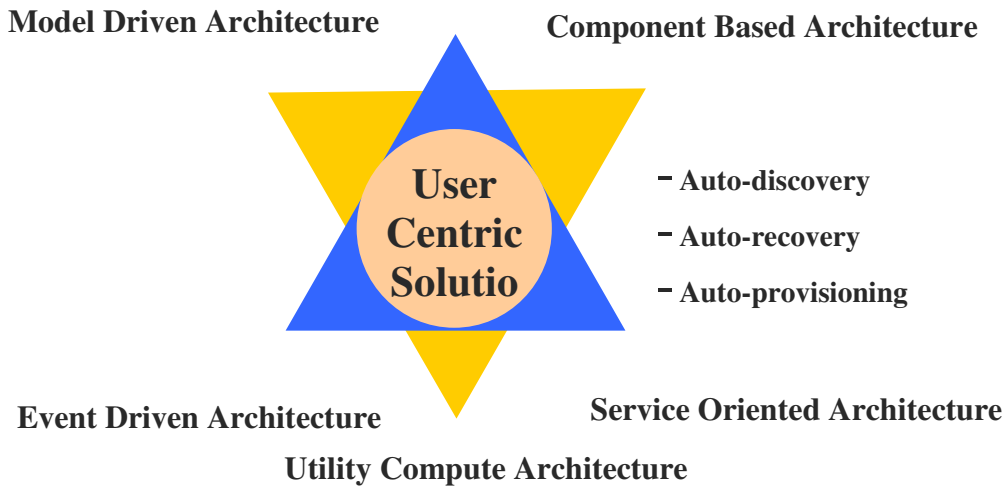


**Figure 5: CBA with container managed components supplementing SOA**

### Event driven Container Alignment for responsive and proactive grid computing (EDA for UCA)

This perspective takes into account the advances in Grid computing that automates alignment of application service containers-ASC (such as J2EE, web containers, JAIN SLEE containers, etc)., to system service containers-SSC (such as Solaris Container or Linux as a system container), that makes UCA an excellent fit for SOA when SOA leverage these container capabilities. Proactive and responsive Grid's will have embedded container alignment engine's (CAE) that align ASC's to SSC's on a constant basis. While ASC's abstracts and aggregates underlying application services (JAAS, JCE, session/state repository, persistence mechanisms, etc.) for business and communication applications, SSC's abstract and aggregates underlying system management services (resource management, telemetry, etc.), for the ASC's. This CAE leverages events, rules, triggers, conditions, policies (SLA/OLA), and more. This approach makes all Services in a SOA built based on component based designs, grid-able and align-able. (see paper on CAE for more on ASC/SSC and CAE).

Modularized Services as Components distributed in a GRID that react to event (Event Containers), ensures;

- Run-time Service requirements are taken into account
- Real-time response is possible
- Proactive measures can be taken based on policies
- Predictive behavior models are addressed

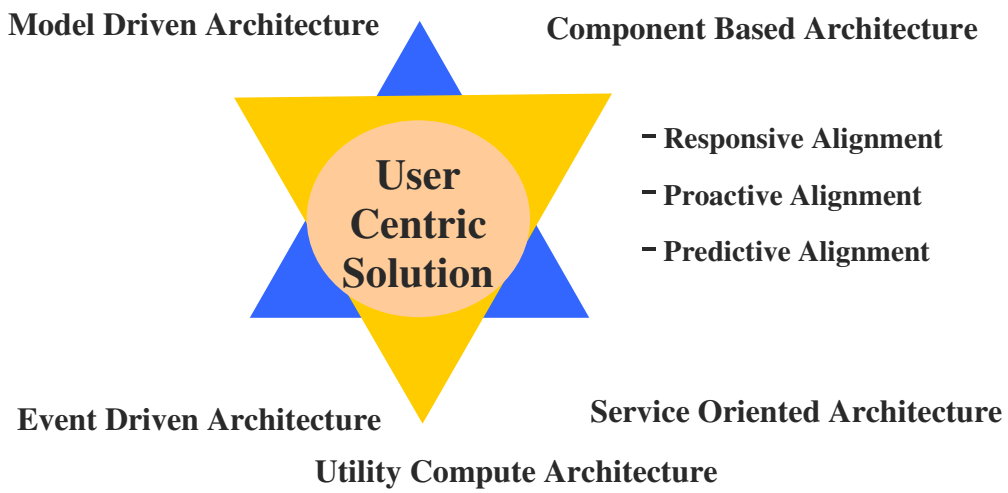


**Figure 6: CBA with distributable components supplementing SOA**

### Modeling resource abstraction for intelligent resource provisioning (MDA-CIM/MOF for UCA)

This perspective takes into account the model driven approach to ensure heterogeneity in a Grid environment, addresses co-habitation and co-existence of different vendors hardware components with a single GRID. For example, Storage Networks (viewed as Data Grids), could be implemented with differing implementation of SAN switching, SAN caching, SAN storage building blocks or subsystems (ones that specialize in redundancies, high-performance, etc.), SAN management software, SAN enabled dynamic File Systems, SAN metric capturing systems, etc. However, with the design principles behind MDA, one can abstract majority of these vendor implementations and align appropriate storage capabilities to the appropriate Service requirements. Similarly in conjunction with PIM and System Service Container based abstraction compute resources can also be leveraged from different vendor's implementations, and platform capabilities can be matches with Service systemic quality requirements. Therefore taking the PIM approach in conjunction with MOF (meta-object factory) and SSC agility can be introduced in a dynamic Grid environment supporting a SOA, in terms of mixing and matching implementations based on Service needs. Services that require higher levels of system qualities (based on SLA) can be mapped to more expensive compute and storage resources from a Grid, and at the same time Services that are not stringent in terms of systemic quality requirements can be mapped to resources that have lower levels of quality characteristics. However if an SLA around a service up-leveled, moving that service to appropriate matching resource components need not involve major migration efforts, when the platform characteristics are abstracted from service requirements via modeling and containers.

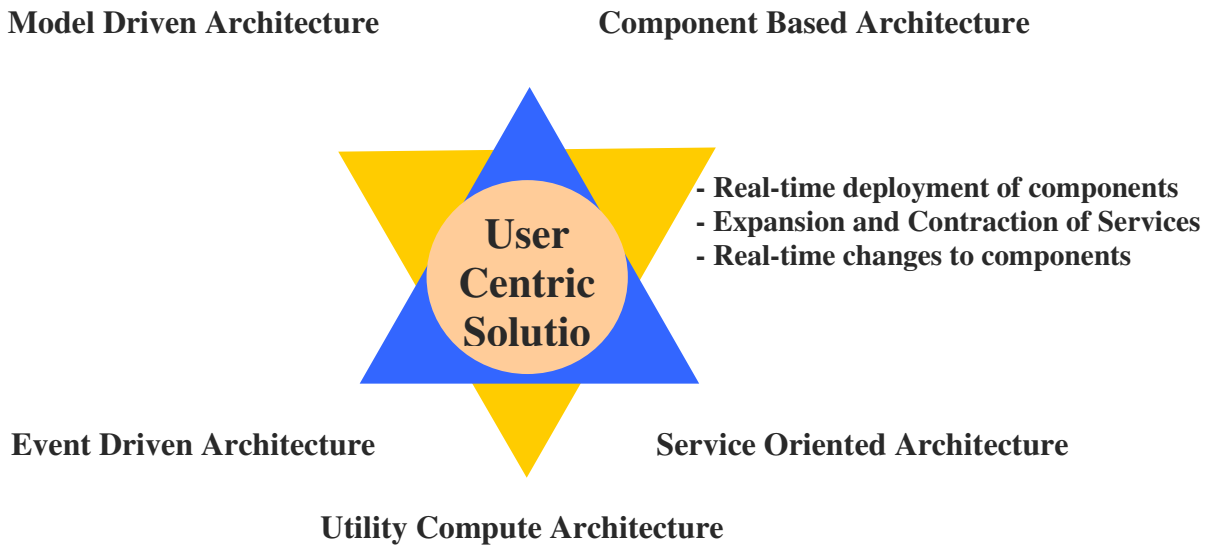


**Figure 7: Code Mobility for Service Elasticity with Affinity**

### Services built on instrumented and thread-able Components lends itself to Utility Grids (CBA+SOA for UCA)

This perspective takes into account the instrumentation and thread based development (container based thread pooling and container based instrumentation) that occurs when services are built based on component based designs and its implications for Service Grids. Instrumentation of components gives grid computing environments a layer of intelligence to ensure automated alignment. A layer, that takes into account application behaviors, around security, performance, state recovery, and more, under varying conditions. This layer of intelligence, through component instrumentation makes services and applications take into account metrics around SLA's and OLA's and work in conjunctions with Grid services –such as auto-discovery services and auto-recovery services. Similarly, component based and task intensive designs running on containers leverage thread pooling, making services highly threaded in nature. This directly enhances the value of high-throughput grid environments that leverage advances in throughput computing and throughput networking. SOA and its user centric nature makes services highly task intensive, i.e., fine grained services with a streamlined interface (xml based) combine multiple SBB (service building blocks) constantly to render task intensive applications, these services in conjunction with container alignment also leverage other thread based services –for alignment with grid services, management services, events, security services, and more. This makes container based thread scheduling and thread pooling a common requirement across all services. Thread rich environments created using CBA and SOA are a great fit to UCA that heavily leverage throughput (both compute and network) via advances in multi-threaded/multi-core chips (single core running multiple threads and multiple cores running multiple threads). Implying not only, align-able services, but also align-able compute and network components.

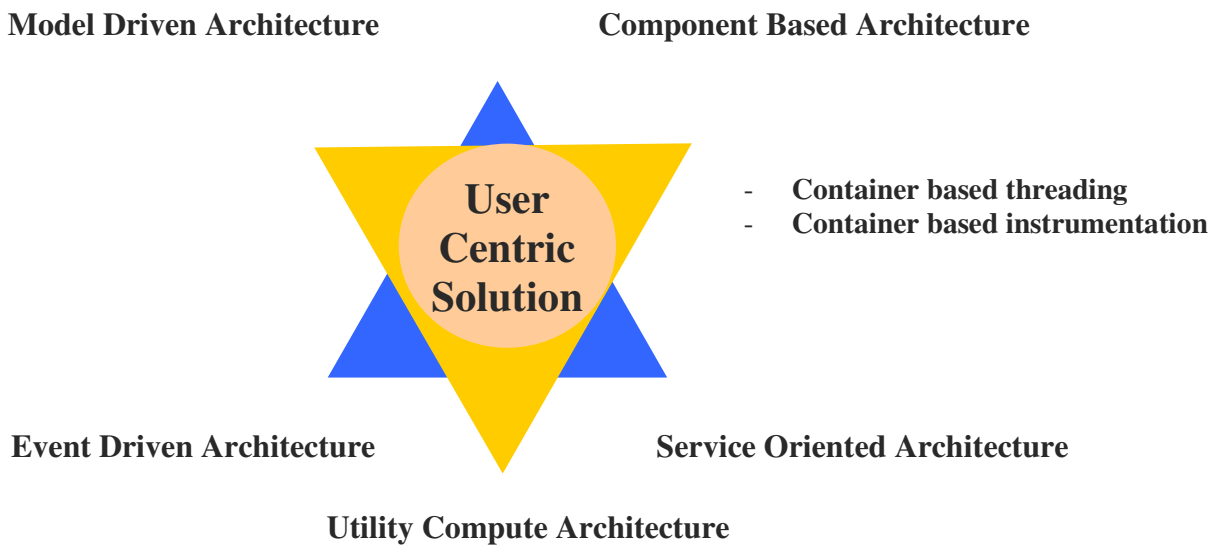


**Figure 8: Highly threaded and instrumented components**

## Conclusion

SOA on UCA embraces all the five areas of synergies discussed between CBA, EDA and MDA as discussed in this paper. It needs to be understood that this combination of conceptual architectural patterns (a.k.a., a conundrum of conceptual compounding) leverage synergies from one-another creating a virtuous cycle of systemic values (as opposed to vicious cycle). Obviously this virtuous cycle of systemic values leads to more Agile Software and Hardware Architecture that can accommodate change more gracefully, than when compared to other approaches.

Cohesive and serviceable components that engender durable services hosted in component containers (providing configurable QoS) at the same time distributing components for dynamic service interactions (between many/types of Containers) plus Instrument-ing components for higher Service Quality along with Impact-free localized changes (that enable continuous re-factoring) all lead to the possibility of addressing changes more gracefully, whether in the functional or non-functional requirements. Leveraging a DEN/IDEN (directory/identity enabled network) and delivering via an Event Execution environment then the Enterprise (Technology) Architecture is set to achieve Totally Agile Solutions that uses end-to-end integration with straight through processing. All these approaches leads to the strategic benefit associated with CBA supplementing SOA, MDA enabling SOA and EDA augmenting SOA, especially in a web services world, where Services are delivered on demand. Along these same lines the Services built as components running on Containers, with an alignment engine (i.e., dynamic Grid computing) that ensure policy based alignment of resources to services, based on SLA, ensures that SOA is a good fit to a more evolved Grid computing environments. In fact service networks are service grids (that run Identity enabled Services), while the core network is an Identity and Event Grid (that run Core Identity Services and tightly tied identity based services), with the access networks acting as Access Grids running edge services that get distributed across the edges of different access networks. This will ensure “seamless mobility” – agility in a different dimension that ensures that we can go beyond “user mobility” and “device mobility” and offer true “service mobility”.