This article introduces the foundations that need to be established in order to implement functional SOA processes. Rather than presenting specific tools, we will define a broadly applicable SOA blueprint whose individual modules can be topped up with commercial products or increasingly available open source offerings.

**A Company Blueprint**

Upon examination of Figure 1, the vision of adaptive enterprise computing is illustrated as a meta-blueprint for the overall company with three differentiated levels:

- **Infrastructure Level** - This level is formed by databases, storage systems, application servers, and all other IT resources that are required to run IT systems.
- **Application System Level** - This level houses entire applications, both individual applications and standard software, as well as services relating to SOA, workflow and BPM systems.
- **Process Management Level** - Functional requirements are manifested in the process design and then implemented at the lower levels.

An advanced service-oriented architecture is the most effective option for implementing the functional requirements at the application-system level. The more superior the mapping of existing business services to the functional steps in process models is, the more the business-IT gap shrinks.

Various back-coupling loops represent the actual added value of this meta-blueprint, meaning consistent usage of services and technical processes enables the measuring of KPIs. This in turn facilitates process control and ultimately the optimization of processes. The more progress a company makes in implementing their SOA, the greater and faster are the effects to be achieved for adaptive enterprise computing.

Now that the vision has been defined, we can move on to consider a blueprint at the SOA level to assist in the specific selection of available SOA components.

**Reference Architectures**
A reference architecture represents the ideal image of the achievable IT landscape and requires many steps for its complete implementation, since project-specific external influences tend to introduce deviations from the plan. The task of the chief architect is to ensure that these deviations do not excessively hinder complete implementation by using the appropriate corrective measures. Without continual checks and directional adjustments, the architecture will gradually expand in a multitude of different directions to result in an unmanageable SOA architecture without earning any benefits like cost reduction or agility.

The architect plays a central role not just in defining the reference architecture but also in the overall project cycle, since similar tasks need to be performed after the initial project ends. Introduction of a strategic SOA program requires adherence to the same set of framework conditions across projects, in order to consistently work toward the defined reference architecture.

OASIS has created the SOA Reference Model TC to assist in conceptualization and standardization of the technology, since services possess decisive properties (effect, reachability, visibility, interaction and so forth) that need to be commonly understood. Although still vague, this guide provides a certain level of detail for these concepts to indicate the properties that must support the identification of components in order to derive SOA blueprint requirements. Therefore, services running on runtime platforms that can influence system statuses need to be announced so that they can comply with policies when called by service consumers using defined access options.

Components of a Service Oriented Architecture
Before the SOA blueprint can be explored in detail, let's first examine all of the components that can be found the architect's toolbox. The proper tool must be selected and logically linked to the target architecture for each use case. Applicable components are described in the following sections, although in the form of "product classes" that can be brought to life in the scope of the project rather than at a specific product level:

Runtime Containers
The main aspect of the Java Enterprise environment, the JEE container is part of the basic equipment and the "ecosystem" for a Java-based SOA with base services for transactions, pooling and clustering. Despite the inclusion of lightweight frameworks such as Spring, a JEE container is an elementary component in an enterprise architecture. Selection of the vendor is usually determined by non-functional requirements, project specifications and personal preference. Note that approaches such as the SpringSource Application Platform and OSGi-based systems are playing an increasingly important role.

Message-Oriented Middleware (MOM)
MOM forms the transport infrastructure of an SOA and provides the characteristic of reachability, while focusing on increasing quality-of-service (QoS). In the simplest scenario, the Java message service (JMS) lends itself to message interchange if the service provider and service consumer both reside in a Java environment. Suitable candidates include native products and traditional queue providers that have a JMS interface, such as Apache ActiveMQ, Oracle AQ and IBM WebSphere MQ.

Enterprise Service Bus (ESB)
Strictly speaking, an ESB provides integration patterns and is responsible for essential tasks that range from routing to achieve reachability, message transformation and protocol transformation to enable interaction, to adaptability for system-standard interface coupling and enterprise SOA environment manageability. The ESB is situated between the service provider and service consumer and is used for decoupling and service virtualization.

Common capabilities of an ESB include:

- **Routing** - The advantage of using an ESB is that service consumers do not have to know the exact endpoint of the service provider they want to call. They can simply message the ESB, which then forwards the message based on its content or header information (content-based routing/header-based routing). Using this virtualization layer allows services to be easily replaced with ESB configuration, without having to alert the service consumers or adapt for loose coupling. The virtualization concept can alternatively be accessed via a registry using late binding, as discussed later in this article.

- **Message Transformation** - Oftentimes the service consumers and service providers do not speak the same language, as they are usually based on generally incompatible "information models" or interfaces. The
ESB can deliver a “translation” service to resolve this dilemma. Discussions on working with transformations and the conical data model will follow in subsequent articles.

The ESB can also offer functions such as security and transaction control, while support of its future standard of service component architecture (SCA) is emerging as a trend.

**Business Process and Workflow Engines**

A business process engine is required in an SOA if business processes are being run automatically. This engine manages the processes that are modeled in a standardized language, which is usually the business process execution language (BPEL), but can also be XPDL or jPDL-based BPM in practice. Nowadays often Business Process Model and Notation (BPMN) 2.0 is used for process automation. Existing services within the running process in BPEL are called automatically from the engine and linked together, which is referred to as service orchestration. BPM engines can work either on a standalone basis or embedded in an ESB, for example as a service engine via JBI.

Typical products include ActiveBPEL, Apache ODE, Oracle SOA Suite and Oracle BPM Suite. These offerings differ primarily in high-capacity operation and how they provide extended functions and meet seamless product integration requirements. The transition to workflow engines is exceptionally fluid, especially where human interaction and implementation via XPDL are concerned. BPM was originally only intended for automatically runnable processes. Nowadays, the human interaction feature has been upgraded for asynchronous interaction using the WS-HumanTask and WS-BPEL4People specifications, resulting in a blurred distinction between the two in practical applications.

Executable processes are used not only for the technical mapping of a functional business process model, but also at the service implementation level for EAI integration processes of higher complexity. The interaction and delimitation of the languages and notations will be examined in more detail in subsequent articles.

**Rule Engines**

Functional rules that are typically not hardwired into the program code because they change frequently or require modification at runtime should be shifted to a rule engine. The functional rules can be defined graphically depending on the system or using relatively natural language, before becoming available for retrieval via various interfaces like Java, RMI or Web service. A typical example is the shifting of a functional decision from a BPEL process to a rule in a rule engine. This move avoids complex XPath logic, and increases maintainability and flexibility by making deployment of the modified process no longer a requirement.

Typical products include JBoss Rules (formerly Drools), WebSphere ILOG JRules, Visual Rules Modeler and Oracle Business Rules.

**SOA Governance Programs**

SOA governance programs relate primarily to organizational issues, and are used to regulate the development, monitoring and administration of an SOA. Typical areas of focus include service portfolio management, SLA monitoring, operation and deployment and service domains and domain managers, as well as modeling and development guidelines, roles in the SOA project and transitions between these roles and their tasks.

Furthermore, if an governance program is developed at too late a stage, there is a greater risk of building an uncontrollable SOA landscape that disregards the commercial and technical objectives established in the reference architecture. You could find yourself faced with a landscape that exhibits the same problems as current applications, such as Point2Point connections, cross-correlations and a lack of standardized format or parallel operation to name a few.

Enterprise architecture management tools and SOA repositories are important aids that support the governance process, and usually also provide an initial guide for management.

A couple of common governance products are the Centrasite Software AG and Oracle Enterprise Repository.
Directory Services

The service consumer that recognizes that a suitable service can be found also needs to know how to locate the service, as per the property of visibility. This can be achieved using a directory service, which is regulated by the standard called Universal Description, Discovery and Integration (UDDI). As service repositories arise, they need to address the issue of governance in addition to registry functionality as UDDI. Although using a repository can be essential depending on the configuration level of the proposed SOA, an established modeling tool or even MS Excel may suffice at the beginning.

Business Activity Monitoring (BAM)

Products that use business activity monitoring usually display functional display runtime information in a graphically appealing manner in the typical management cockpits. The visualization of values from running process instances in a BAM dashboard using sensors, as per KPI monitoring, is particularly straightforward. BAM products usually also have corresponding adapters/sensors so that data is also accessible using Java, PL/SQL and other common languages. This enables the monitoring of lower threshold deviations for vehicle reservations within the purchase processes, and appropriate manual and automated countermeasures to be taken.

Well-known BAM products include the Oracle BAM, vFabric Hyperic, Systar and SeeWhy.

Complex Event Processing (CEP)

Complex event processing (CEP) is a relatively new field, and its technologies are still in the early stages of development. The purpose of CEP is to allow for browsing through event streams according to specific patterns that can be uncorrelated in time, space or content. A typical use case is the detection of attempted credit card fraud when the order of events that occur indicates the same card being used within an interval of a few hours in completely different locations. Current areas of application also include financial transactions in high-volume scenarios.

Continuous query language (CQL) provides an SQL-style query language that adds elements for querying and processing data streams to the SQL language constructs. Typical products include the Oracle Event Processing and Esper.

The SOA Blueprint in Action

All of the components that were previously described assume important sub-functions, and can be consolidated into a coherent SOA infrastructure that is called an SOA blueprint (Figure 2). Not every project needs to be armed with a full arsenal of products right away, since this process is evolutionary. In initial SOA project stages, fundamental components such as ESB and/or BPM engines are gradually joined by other components to gradually achieve business objectives, as requirements and technical expertise increase.

Once the appropriate product classes have been selected, the blueprint must be brought to life. Whether a suite solution from a commercial producer, an open source suite or a best-of-breed approach is chosen is of secondary importance from an architectural perspective. Typical product stacks include the JBoss Enterprise SOA platform, Glassfish SOA stack, Geronimo GASwerk SOA stack, SOPERA ASF and Oracle SOA Suite. The bundling of standalone products such as Apache ServiceMix, Apache Camel and Apache ODE into a proprietary suite is also viable. However, the architect must be aware that consistency, manageability and stability are important characteristics whose absence usually results in significantly different solutions.

Another trade-off that must be considered for practical applications is open source middleware offerings, which allow for greater control and more intervention options since the source code can be modified if necessary. However, this alternative first requires the comprehension of highly complex topics, such as the TX behavior of an open source engine. Such changes are either not possible in a commercial suite or usually much more time-intensive, since product patches are used. Note that commercial support can now also be purchased for particular open source stacks.

The consistency and integration of integration products are other areas of concern, since using an integration product that does not cleanly integrate into the landscape is not reasonable. SOA is highly complex and its advantages can quickly slip away if it does not promptly interact with the IDEs due to developer inefficiency. Integrated monitoring consoles are also essential for runtime, since the ability to trace the message flow through the process engine, ESB, rule engine and other components is an important feature of a well-integrated product stack.
With regard to a specific product selection, there are numerous deciding factors that are dictated by the specific requirements of each case and require value-benefit analyses. Adherence to the blueprint is critical and necessary regardless of the actual product selected. Flexibility that aligns with the requirement of design for change is achieved if the selected modules are connected together in a loose coupling. Individual products can come and go, but the architecture must persist.

**Example: RYLC**

Figure 3 illustrates a rental process at a car rental company called RYLC that reveals how the product classes interact with one another at runtime. These interactions are implemented as a BPEL process and provided via an ESB for virtualization.

The steps comprising the company's car rental process are as follows:

*Step 1* - A front-office application calls the ESB service, which starts an instance of the executable process.

*Step 2* - The first service call that is referred to as "readCustomerData" locates all of the customer data in CRM1, using a service that provides the corresponding CRM system functionality in virtualized form via the ESB. An adapter for the CRM system or service can be used directly as an alternative, a method that calls it via the ESB without the implementation of a "detour." The effects of this architectural decision are discussed in subsequent sections.

*Step 3* - Whether or not the customer is to be offered special perks based on their CRM data is a decision that needs to be made. This decision is usually shifted to a rule engine in order to maintain rule changes at runtime flexibly. Technically speaking, the rule is called via a "decision service."

*Step 4* - All of the available vehicles, options and prices that match the query period from the car rental system are determined. As in the second step of this process, the corresponding functionality from the car rental system is extracted as a service and provided via the ESB.

*Step 5* - Possible vehicle combinations are established, from which the customer can make a selection and initiate the binding reservation. Note that the process also needs to pose a question to the user at runtime, which presents an interesting challenge from a technical perspective. This topic is discussed in detail in a proprietary User Interface design pattern article.

*Step 6, 7* - Normal operations resume and services are called via the ESB, which writes the required data back to the CRM systems and arrange for billing. The rental process ends with a query to the customer as to whether his rental enquiry has been successfully completed.
The Accidental Integration Architecture

With our current knowledge of static blueprint and dynamic coupling options, it must not be assumed that only logical architectures are automatically created (Figure 4). Individual components of the SOA blueprint have been used, although without a comprehensive master plan or any breach of the defined reference architecture. A typical "spaghetti" infrastructure has been formed in this scenario that has numerous components communicating with one another directly. A common infrastructure that brings order to the many cross-connections is missing, which prevents the uniform implementation of security and QoS, among others.
Figure 4: A sample accidental integration architecture

A Well-Organized Toolbox
Figure 5 depicts a much improved architecture variant in which an ESB cleanly encapsulates all communications. Reusable adapter services in particular are available via the connectivity layer that can serve the connected systems bi-directionally. Necessary schemes for mapping data structures are provided via a metadata repository. One of the advantages of this architecture is that security and quality-of-service (QoS) perspectives can be handled centrally via the bus, which is a key disadvantage of the scenario presented in Figure 4.

This example plainly illustrates that even the most superior toolbox sometimes cannot lead to the desired solution on its own. The task of the architect is to bring order to the toolbox, and the function and fields of use of each individual tool must be identified. Which tools interact optimally in our blueprint, and which combinations introduce difficulties and/or redundancies? Has the reference architecture been breached? In practice, the outcome of complex projects is heavily dependent on the experience and expertise of the architect in charge.
Common Issues

Let's examine several examples of issues that real-world projects may be faced with when establishing an SOA blueprint.

Question 1

In which fields should be BPEL or BPMN 2.0 used?

Answer 1

The main field in which BPEL/BPMN is used is the implementation of BPM scenarios. Here the process represents the technical implementation of a functional process. There is little logic in the process itself, as all of the complex decisions and activities have been shifted to external services to provide a highly homogeneous perception of the processes to be implemented.

BPEL is also often used as an EAI tool in practice, and many tools include suitably convenient adapters that use wizards to click private services together rapidly. These scenarios focus on the flow of information via various systems and the necessary data transformations, rather than on the functional process.

A third field of use is workflow scenarios. Whether a task is initially processed using an automated service or a manual task is not critical, and products such as WS-BPEL4People and WS-HumanTask blur the boundaries of fully automated service calls. Although supported by some tools, the transport of transaction contexts via BPEL processes should be avoided by default. The framework conditions for stable operation are so complex that impending issues are inevitable by the time the maintenance phase is reached.

Extra care needs to be taken where batch scenarios are concerned. A process that was designed for n-number of calls a day cannot be readily integrated into batch runs. Consider the parallel start of 500,000 process instances as initiated by a batch controller. No process engine will be able to logically process this peak load without requiring precautionary measures.

Question 2
Why do I need an ESB if I already have a process engine tool that includes ESB functionality? The complexity increases and I gain nothing else.

*Answer 2*

The superiority of the support for most process engine tools today has prompted many consumers to implement components directly in the tools, instead of correctly shifting the functionality to the ESB. For example, consider a value from a database table that is required in a process. Instead of defining or changing a service, a database adapter is used directly in BPEL/BPMN because of its simplicity and ease, which is correct in principle. The database adapter can be considered a technical service that has private visibility and so does not have to pass through a service lifecycle.

However, adaptations become risky if the project reaches the maintenance phase. In the worst-case scenario, database connection strings become distributed throughout the entire application to impact flexibility. Making changes during the runtime of the process instance without redploying the modified process template is usually not possible. Conversely, the process is easier to maintain, tidier and much more flexible if it is viewed from a purely functional perspective and the adapter/data-read service is provided via an ESB, whose configuring allows for interventions during runtime.

*Question 3*

The screen flow of my Web application is already implicitly stored in the process flow. Why can't BPEL or BPMN 2.0 be used as a pageflow engine?

*Answer 3*

This question pertains to how we apply the separation of concerns. There are still applications that require internal micro-page flows to which numerous reliable page flow controller frameworks can lend themselves. A use case is usually fully programmed in an application. Conversely, a process focusing on a BPM scenario consists of functional sequences that can each be considered to be a proprietary use case. The process would therefore assume the role of a use case controller, but not have any control over the micro-page flows within the use case. This topic will be examined in more detail in a subsequent article.

**Revisiting RYLC: Selecting the Components**

So, what do the aforementioned considerations mean for our car rental firm RYLC? They have made the decision to go ahead with adopting SOA and the processes to be implemented have been defined, leaving the final question of which blueprint components to use.

The basic condition is the creation of rapid successes that can address the company's problems while providing as many advantages as possible. The conventional approach of introducing a complete SOA landscape from the start is therefore not feasible due to the time constraints. A "quick-start" approach to SOA that can rapidly produce reference projects may be more suitable, in order to enable an efficient transition to creating an SOA landscape.

Two components are selected from the SOA blueprint for RYLC's first rapid-SOA project:

- *BPM Engine* - to implement the new processes, which should no longer run hardwired into the applications
- *ESB* - to address the pending integration scenarios and provide a virtualization layer for services that are orchestrated by the BPM engine

Scheduled for the next phase, the use of a rule engine is classified as necessary in order to gain greater flexibility to accommodate frequently changing rules. No registries, BAMs or CEPs were initially included, as short-term benefits are not expected.
The infrastructure that RYLC will implement for its first SOA project has been established. The upcoming articles explore the services to be implemented in more detail and discuss the corresponding guidelines.

Conclusion: Key Takeaways

- Stick to the concepts, regardless of whether you're starting with an open source solution or a service provider's commercial stack.
- SOA contains a high level of complexity, and any opportunity for reduction must be taken. If possible, use the suite approach and shift the responsibility of integration over to the service provider.
- Future requirements can be fulfilled by replacing favorable and/or open source components with commercial products that may offer greater efficiency.
- Service Loose Coupling is often the most important principle to consistently adhere to when building an SOA.
- Flexibility is achieved if SOA concepts are successful at the tool level, since switching tools is much easier if the concepts are compatible.
- Differentiate a specific tool from its functions. Hot-pluggability is achievable in the context of best of breed.