TOWARDS A SERVICE-ORIENTED ARCHITECTURE: A FRAMEWORK FOR THE DESIGN OF FINANCIAL TRADING APPLICATIONS IN THE SOUTH AFRICAN INVESTMENT BANKING ENVIRONMENT

W. Vos[1] and **M.C. Matthee**[2*]

[1]Department of Informatics
University of Pretoria, South Africa
[1] wvos@investec.co.za, [2] machdel.matthee@up.ac.za

ABSTRACT

Service-Oriented Architecture (SOA) enables organisations to let their business drive their IT strategy, and creates a technology strategy that is aligned with that of the business. SOA is an architectural style that enables the integration of disparate systems, independent of the implementation technology or physical location, through encapsulating and integrating business processes as a collection of coarse grained, loosely-coupled services. This article aims to examine the ability of SOA to satisfy the requirements and concerns posed by financial trading systems, and to present a SOA framework for building an automated trading application.

OPSOMMING

Diens-georiënteerde argitektuur (Service-Oriented Architecture (SOA)) stel organisasies in staat om die IT-strategie vanuit die besigheid te bestuur, en skep ‘n tegnologiestrategie wat belyn is met dié van die besigheid. SOA is ‘n argitektoniese styl wat die integrasie van uiteenlopende stelsels moontlik maak, onafhanklik van die implementeringstegnologie of fisiese ligging, deur besigheidsprosesse te enkapsuleer en te integreer as ‘n versameling van losweg-gekoppelde dienste. Die doel van hierdie artikel is om SOA te ondersoek as ‘n moontlikheid om te voldoen aan die vereistes en knelpunte wat finansiële handelstelsels stel, en om ‘n SOA-raamwerk vir die bou van ‘n outomatiese handelsapplikasie te bied.

[1]The author was enrolled for the M.IT degree at the Department of Informatics, University of Pretoria.
*Corresponding author.

1. INTRODUCTION

To create a sustainable competitive advantage, organisations continuously need to re-evaluate their position and quickly adapt to changes in the business environment (Moitra & Ganesh [1]; Eisenhardt & Martin [2]; Harreld et al. [3]). This is especially true for the financial industry. In fact, agility in the stock market is an absolute imperative. Organisations in this industry need to adjust their trading strategies very quickly to exploit new trading opportunities in a rapidly changing market. This kind of productivity is critical to create new algorithms for pricing and trading, and to free up more capital through timely and accurate risk management. Also, the demands imposed by regulatory and/or legal compliance in the financial sector are becoming even stricter. So IT has to produce more compliant and more flexible solutions, while also delivering business value.

The challenge is to integrate business and IT processes so that IT supports the business. The emphasis is on the need for flexible IT systems that enable business agility (Mitchell [4]; Moitra & Ganesh [1]; IBM [5]). However, organisations are often burdened with heterogeneous legacy systems. The financial indstury is plagued by these and by various other architectural challenges that will be discussed shortly.

To meet these architectural challenges, Service-Oriented Architecture (SOA) is proposed as an underlying architecture that aims to enhance the interoperability and data exchange capabilities of applications (Xiaorong [6]). It allows for easy integration of heterogeneous components to satisfy business requirements.

2. ARCHITECTURAL CHALLENGES IN THE FINANCIAL ENVIRONMENT

In a typical financial institution, large numbers of complex services – such as risk management, trading, and portfolio management – are handled by a range of individual applications (Pan & Viña [7]). To optimise business processes and achieve a higher level of business process integration, there is a need to integrate the existing functionality and facilitate interoperability between them.

With this goes the need for efficiency (Birman [8]; Birman et al. [9]). Many of the commercial vendors provide software to automate business process integration through web service integration – e.g. IBM Websphere and Microsoft Biztalk. Current technologies, such as web services, focus on providing interoperability using internet-based communication standards such as SOAP. However, financial systems are usually based on propriety protocols (Birman [8]; Birman et al. [9]). This means that, in order to facilitate interoperability between these systems, messages have to be translated from their native format to SOAP and vice versa. This results in inefficient solutions that work against the requirements of (for example) trading applications, which require high levels of throughput and low levels of latency (Birman [8]; Birman et al. [9]).

In addition, in order to enhance the organisation's agility, there is a need for flexible architectures (Mitchell [4]; Moitra & Ganesh [1]; IBM [5]). The legacy architectures encountered in these organisations make it difficult to change, and complicate integration with other enterprise systems. As a result, these organisations often perceive IT as a constraint on business rather than an enabler (IBM [5]).

In the face of limited budgets, there is enormous pressure to preserve and leverage existing IT assets. Little value is gained in rebuilding legacy systems in a new technology when it is meant to meet the 80/20 rule (Jones [10]). In addition, when legacy systems are simply ripped out and replaced, the knowledge and practices embedded in these systems are lost (Bruner [11]; Smith & O'Neal [12]). The architecture must be flexible enough to support the gradual replacement of legacy systems, if need be.

This article explores SOA as a possible solution to address both the business concerns of

agility and the architectural challenges that hinder organisational agility in the context of an investment banking environment. This is done by proposing a conceptual SOA framework for building an automated financial trading platform that is capable of addressing both the business and architectural challenges explained above. The layout of the rest of the paper is as follows: Section 3 investigates the underlying concepts and principles of SOA, and indicates the link with Enterprise Architecture. The research method is discussed in Section 4, after which the business context of the investment banking environment is explored. The proposed SOA architectural framework is discussed in Section 6. Section 7 includes an evaluative discussion of the framework, and section 8 offers some final remarks.

3. SERVICE-ORIENTED ARCHITECTURE

SOA is a distributed architecture style for building systems based on loosely coupled, coarse grained independent components called services (Brown et al. [13]; Mehta et al. [14]). SOA introduces a paradigm shift that views services as the building blocks of applications (Mehta et al. [14]).

Services are seen as interfaces that provide a set of functions that can be invoked irrespective of their location, without any knowledge of the implementation details or the technology in which they have been developed (Perrey & Lycett [15]). A service encapsulates complete business functions that are intended to be reused and engaged in new transactions across enterprises. In the same way that an object in Object Oriented development represents a real-world 'thing', a service represents a real-world 'business function'.

SOA promises interoperability regardless of the platform, service provider, or location of the service. SOA provides an approach for how to describe, orchestrate, and publish services to support their discovery and use (IBM [16]). Service-Oriented Architectures consist of three main structural elements (or roles, according to IBM [16]): a service provider, who publishes the availability of services; a service registry/repository - an entity that serves as broker, registering and categorising the published services and providing search capabilities; and a service consumer – a software agent that interacts with the service by requesting execution of the service (Huhns & Singh [17]; IBM [16], Papazoglou [18]).

Many financial institutions have deployed a SOA in order to create streamlined, simplified and easy-to-manage technology infrastuctures (Sanchez [19]). Wachovia Bank transformed its architecture into a Service-Oriented Architecture that can be leveraged by all business units (Margulius [20]). Zimmermann et al. [21] used web services to breach the technology differences between existing client components, to provide a unified interface and integrate inter-organisational banking systems. These examples of SOA deployment all introduce concepts and ideas, such as a unified point of access to ensure consistent access to the systems, or technology neutral interface services to allow for the system to integrate with other organisational systems. These concepts and principles provide a foundation on which one can build. However, these solutions are examples of how SOA can be achieved by means of web services. Web services are the most common implementation, since for most organisations it is the simplest approach to implementing a loosely-coupled architecture (Microsoft [22]). However, web services are not the only way in which to implement SOA (Brown et al. [13]). Many issues remain to be resolved in building high performance financial transaction systems using the existing technologies (Birman [8]; Birman et al. [9]). Current technologies such as web services focus on providing interoperability using internet-based communication standards like SOAP, while financial systems are usually based on propriety protocols. The problems associated with this – such as the inefficiency resulting from the translation of messages to SOAP and vice versa – have already been discussed in section 2. The SOA framework proposed here was designed for the building of financial trading applications with the above issues in mind.

From a business perspective, SOA must provide value to the organisation beyond just

solving technical integration issues. In order to create value, the IT efforts need to be aligned with the business strategy. Ross [23] explains that Enterprise Architecture is used to place the technology standards in the context of the business requirements. Jones [10] describes it as the process of understanding the business, IT, and vendor strategies, and defining the practices for the organisation to align these strategies in order to achieve their business goals and objectives. Let us look, therefore, at the relationship between SOA and Enterprise Architecture.

3.1 The relationship between SOA and Enterprise Architecture

Afshar [24] notes that SOA is a framework within Enterprise Architecture, and aims to achieve the same business goals: minimising the total cost of ownership, and creating flexible business solutions that improve business agility, reduce the time to market, and provide support for global expansion. SOA substantially impacts all the key aspects of Enterprise Architecture. The business services proposed by a SOA form the foundation of the business architecture and process architecture (Afshar [24]). Marks & Bell [25] argue that SOA forms a conceptual business architecture where business functionality is exposed as shared, reusable services. The business processes, services, and events are converted to appropriate application services that create and support the product/service architecture. The services themselves form the application architecture, while the information architecture is satisfied by ensuring data standardisation and availability through the service interfaces.

Jones [10] notes that there are four steps to defining a service-based architecture. The steps are about putting the technology perspective into the context of the business so that the solutions are aligned with the business and support the business vision at all stages. The first is to determine the *what*. This involves identifying and understanding the nature and scope of the business services. The next step entails the identification of the entities that will interact with the services, i.e. the *who*. The third step involves the understanding of *why* the service is required by the entities that interact with it. Combined, these aspects provide the direction to the fourth step: *how* the services should be implemented in order to align the IT efforts with the business strategy. SOA does not replace the different aspects of Enterprise Architecture, but instead provides a framework that builds on the same aspects (as indicated above) to provide direction for how it should be implemented. The architecture models created to address the different levels of concerns (Zachman[26]) should deal with the *how* and provide structure for strategy, implementation, and support (Jones [10]).

The purpose of SOA is not to focus on the detail of the applications, but to ensure that the business context is understood (Josuttis [27]). In order to understand where one needs to focus as an IT organisation, it is critical to understand the context of the services and why they interact. If this is understood, the different processes can be represented as services, and combined to form the business process.

4. RESEARCH METHOD

The proposed SOA framework is a conceptual architecture – a structural model that abstracts from the implementation and execution of the architecture, and thus provides a useful starting point on which to base more detailed architectural work (Reekie & McAdam [28, p 39]).

To create this framework the following activities were performed. A literature survey was conducted to ensure a good understanding of SOA. The researchers also conducted interviews and observed how people currently work in three different South African financial institutions that provide a fair representation of the twelve investment banks in South Africa. The interviews were held with individuals (including traders, brokers, and portfolio managers from each organisation) in order to develop a full understanding of the trading processes, the participants involved in them, and the business context in which the

model will be expected to work. The identified processes, requirements, and concerns served as the basis for the functional elements, and provided the forces to shape the architecture. Although the framework has not yet been implemented, nor evaluated against similar architectures because of the sensitivity around intellectual property and propriety knowledge of the banks, it was evaluated through interviews with a practitioner.

5. UNDERSTANDING THE BUSINESS CONTEXT

In order to understand the business context, the trading processes of three different financial institutions were investigated. The researchers conducted interviews and observed how people currently work in these organisations. The list of questions (based on the four steps suggested by Jones [10]) used to conduct the interviews are available from the authors. The data were analysed by analytical induction (developing causal explanations) and identifying critical incidents (Myers [29]), which translates in this context into functional units of work. The 'critical incidents' involved in the trading process were identified as:

- receiving an order (message containing the buy/sell instruction) from a variety of sources
- validating the order, and necessary checks such as available cash balance
- executing the order
- routing the orders to the appropriate destinations
- matching the executions with the orders
- reporting the executions.

Algorithmic trading systems provide a platform to analyse real-time market data quickly, to uncover opportunities, and to respond immediately by placing and managing orders in the market (Bates [30]). Each algorithm it executes represents a different strategy (e.g. Volume Weighted Average Price, Time Weighted Average Price, Pairs, etc.) which defines the logic for detecting the trading opportunities. The traders decide which strategy to use for a given order based on the client's instructions. They provide the input parameters for a given algorithm, such as price limits and total number of shares to buy, which is then used to determine the timing, the price, and the quantity of an order to trade. The orders can also be received electronically via FIX (Financial Information eXchange) messages.

It was apparent that organisations in this industry need to adjust their trading strategies very quickly to exploit new trading opportunities in the rapidly-changing market. This means that once an opportunity has been identified, these organisations need to be able to move forward immediately and adapt existing algorithms – or create new algorithms – with the minimum effort. This emphasises the need for flexible architectures that enable organisational agility.

It became clear that traders/portfolio managers need to trade quickly to respond to market conditions. In the same way, the automated system is expected to be able to process the data quickly and respond within milliseconds. In order to automate the trading process fully, the orders need to be sent to the automated trading system automatically. It should therefore be able to interoperate with the various applications used to receive the orders, and send back the execution reports as the traders do. This highlights the need for performance and interoperability with other systems.

6. THE PROPOSED ARCHITECTURE

In the section below, the authors give an outline of the proposed conceptual architecture. The architecture has not yet been implemented, and is purely conceptual. A short evaluation of the architecture is given in section 7.

6.1 Architecture overview

To maximise both efficiency and interoperability, the researchers differentiate between two sets of services: *interface* services and *internal* services. The internal services each perform a business function. 'Internal' means that these services interact only with other internal services or interface services. To maximise performance, the internal services communicate using a predefined native message format.

The interface services provide a façade to the internal services. The interface services are wrapped by different endpoints that deal with interoperability (the input/output adapters in Figure 1). These services are responsible to interact with external client applications, and translate the messages into the native format used by the internal services.

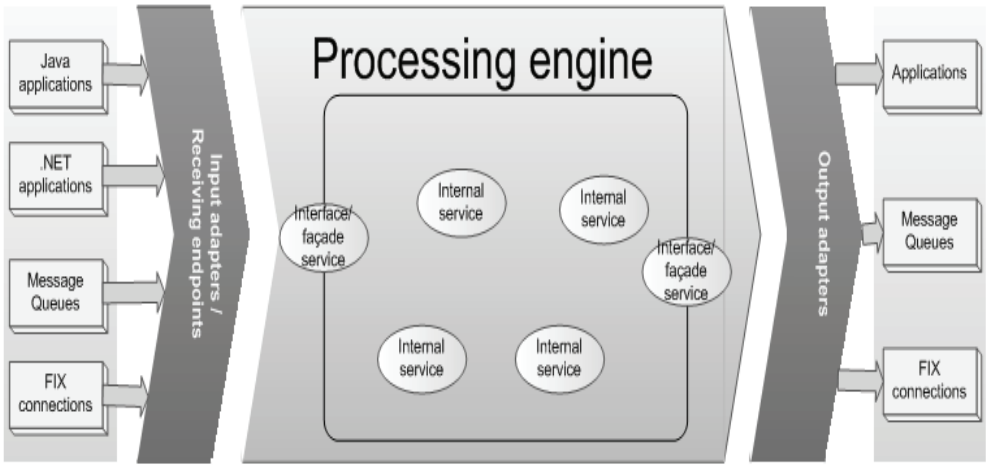

Figure 1: High level architecture

Different bindings can be used between different services. The binding between the interface services sending the order into the trading system can be configured to use a Microsoft Message Queuing (MSMQ) binding, to ensure reliability and thus a guaranteed delivery of an order; while the binding between the trading engine and the services responsible for placing an order into the market can be TCP binding, to ensure speed.

Apart from the functional units of work listed in section 5, a number of additional functional and non-functional requirements need to be addressed, as gathered from the interviews and observations:

- The system should be able to process large volumes of incoming messages and respond within milliseconds. In addition, the system should be able to maintain dedicated, low latency FIX connections to multiple brokers.
- To support an increasing number of trades and algorithms/analytics, the system must be scalable. This implies that one must be able to add new and configure existing endpoints dynamically.
- The system must provide sufficient security measures such as authentication and authorisation to ensure confidentiality, reliability, and integrity, to avoid or minimise expensive misdeals.
- Last, the system must provide the ability to recover from a failure, provide a full audit trail of events processed, and monitor capability.

The next section presents a more detailed discussion of the architecture that addresses these requirements, along with the architectural challenges discussed in section 2.

6.2 Detailed Architecture

Figure 2 below presents a detailed overview of the proposed system architecture:

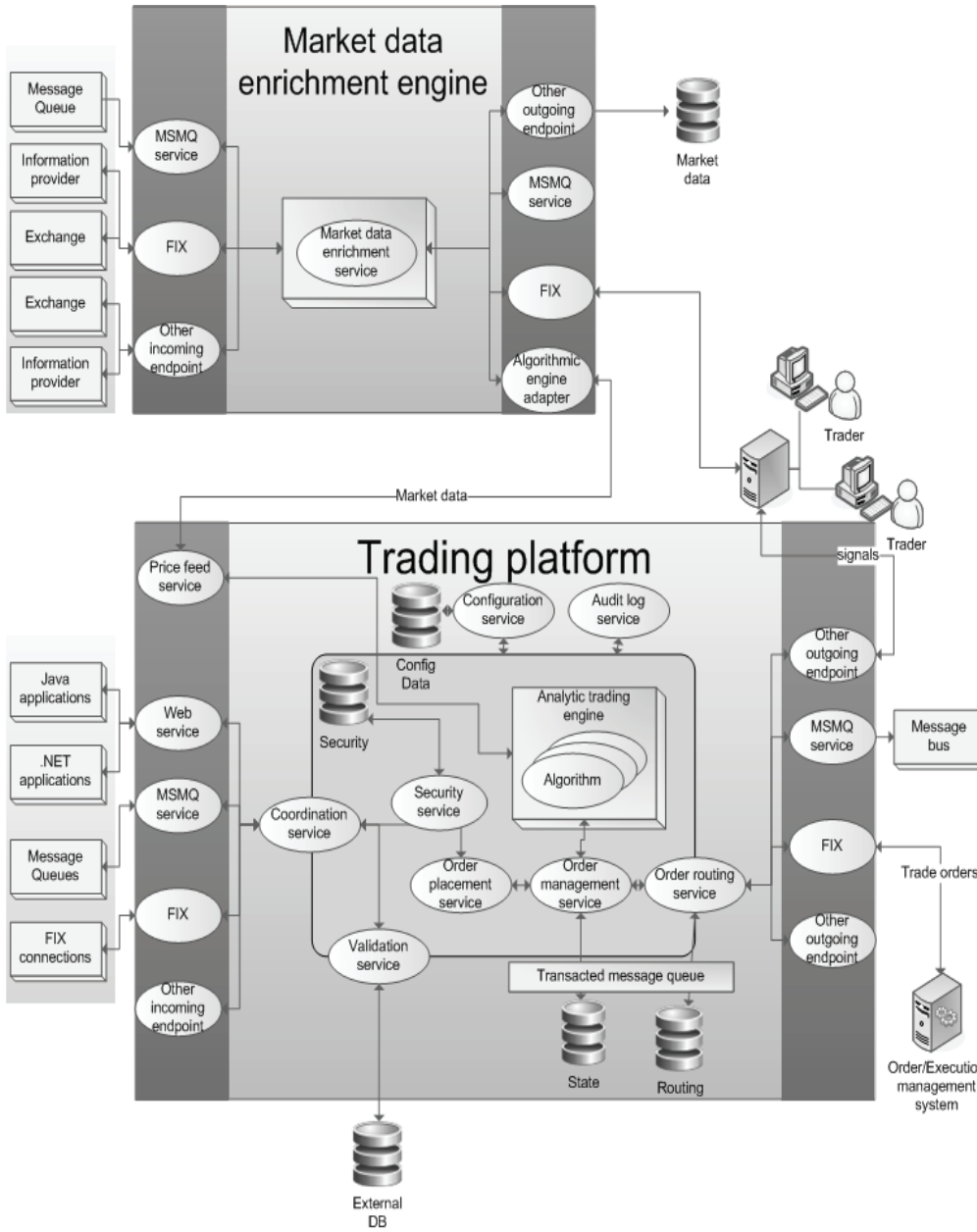

Figure 2: Detailed system architecture

The market data enrichment engine subscribes to real-time data sources, enriches the data with value-added metrics such as VWAP (Volume-Weighted Average Price) and risk models commonly used by algorithms, and publishes the enriched data to services/applications that subscribe to it, in real-time. To persist, the processed data asynchronously and without affecting performance, a service responsible to handle only the persistence also subscribes to the output.

To define a service-based architecture, each of the tasks and concerns is separated and encapsulated in separate services. Orders are received from any external client application that interacts with the system by passing messages (containing order details and the execution algorithm) through the input adapters.

The message is sent to the analytic trading engine at the heart of the system after validating it through the validation service. The trading engine then executes it. The algorithm makes use of the enriched market data to determine how much of the total order to trade at a given point in time. The algorithm conducts a 'what-if?' analysis by analysing the market and trade data against the client mandates and restrictions.

The trading engine generates messages containing the resulting instruction (order) and passes it to the order management service, which in turn updates the status of the order and sends it to the routing service. The routing service then routes the order to the appropriate destinations. The order status is updated according to the trade data (execution reports) flowing back into the system. The orders are also electronically routed to the trading desk, and allow portfolio managers to monitor the status of their orders throughout the day and direct allocation of executed trades across their account base.

Each of the components will now be discussed in more detail in order to explain how the functional and non-functional requirements and concerns specified in section 6.1 are addressed.

6.2.1 Integration with heterogeneous applications: Adapters

The adapters are located between the coordination service and the external applications that connect to the system. The adapters make it possible to integrate with different systems using different transport protocols. Although the adapters are labelled 'input' and 'output', they are bidirectional. The input adapters/receiving endpoints represent the interface services, which translate incoming messages into the native format used by the internal services and back to the format expected by the external systems.

The output adapters translate the system responses and send them to the appropriate destinations. The output adapters can also be used to route messages to risk management systems so that compliance offers can maintain client and firm-wide guidelines, and evaluate where portfolios and holdings stand versus restrictions.

The services achieve maximum interoperability through the use of technology-neutral standards. Just like Zimmermann's [21] web service adapter layer, the adapters allow for the integration with inter-organisational banking systems. However, the FIX protocol serves as the de facto standard for securities trading around the world (Technical Analyst [31]). It would be inefficient first to convert every FIX message into a SOAP message, and then into the native format of the system and vice versa to facilitate interoperability. For this reason the architecture allows for extending the system, and supports flexibility by adding adapters to ease the integration into existing architectures, both contemporary and legacy. Figure 2 illustrates four types of adapters (interface services): Web service endpoints, message queue endpoints, FIX endpoints, and other endpoints. Together these services expose an interoperable layer of interfaces that allow for integration with heterogeneous applications.

6.2.2 Security and reliability: Adapters

Each protocol supported by the mentioned endpoints defines its own standards and offers functionality for security (such as authentication and encryption), distributed transaction coordination, and reliable communication.

Each adapter handles protocol specific requirements, and is responsible for translating a

message into the internal format and delivering it to the coordination service. For example, the FIX adapter is the only adapter that may contain any FIX communication, so there is a complete separation between the adapter's responsibilities and the internal services. This follows a modularised approach, and abstracts the protocol specific requirements from the rest of the system components. It promotes loose coupling, and also allows for extending the system to integrate with other sources if required.

Thus, in addition to providing an interoperability layer, the function of the adapters is to maintain a reliable and secure connection between the external environment and the trading platform by implementing the protocol-specific standards.

6.2.3 Consistent access: Coordination service

The coordination service represents a façade that provides a simplified interface to the system. It provides consistent access to the system, and enforces operational concerns such as security, validation, and auditing.

Although each protocol defines its own security standards, the adapters are not responsible for controlling the authentication to the system itself. Instead of having each adapter implementing authentication, the security credentials and user context parameters are passed to the coordination service, which then handles the authentication and authorisation requests. In addition to the session authentication, the coordination service enforces additional security measures, activity logging, and validation by passing the messages through the security service layer and validation service prior to delegating the market order request to the order management service.

This promotes a loosely-coupled architecture and adds flexibility, in that one can easily add new adapters without having to care about implementing authentication and validation. The service reduces the dependency of the adapters on the internal services of the system by providing a single point of entry that abstracts all internal communication from the 'outside world'.

6.2.4 Authentication and authorisation: Security service layer

The security service is used by the coordination service to perform authentication and authorisation checks. Much like Zimmermann's [21] architecture, the service requires details such as the application and session identifiers, which are then validated against a database of authorised logins. The login details include username, password, and the IP address of the message's origin. This is used for two purposes. First, in the case of adapters that establish a dedicated connection with external applications, such as with FIX adapters, the result of the verification determines whether to accept or reject the connection request; and second, as mentioned above, the coordination service invokes the security service prior to sending the order to the order management service. In this case, since the message can be received from an adapter that does not maintain a dedicated session, the source of the message (IP address of the sender) is used to validate whether the service consumer is authorised to send an order.

6.2.5 Validation: Validation service

The validation service is a composite service consisting of a collection of validation objects. It is used to ensure that the order is valid prior to sending it to be processed. The validation can range from verifying an account number to checking the available cash balance.

6.2.6 Order management: Order management service

The order management service accepts new orders from any client application that interfaces with the system using the input adapters. It creates messages to send new orders, cancel pending orders, or replace existing orders based on the instructions from the

trade engine. These messages are then routed via the order routing service to the output adapters. The service maintains a memory map of all orders and associated executions to maximise performance.

The service also persists the order and execution state to database for failover purposes. A dedicated persistence service ensures that the persistence happens asynchronously and does not affect performance. It also frees the people developing the analytic from the responsibility of managing orders themselves, and allows them focus solely on implementing a new trading strategy. This adds flexibility to the system and supports business agility.

6.2.7 Executing algorithms: Analytic trading engine

The analytic trading engine is the central nervous system of the trading platform. It is responsible for hosting and executing the algorithms that decide how to slice a large order into smaller trades and when to send the trades to the market.

It receives the message (order) containing the algorithm to use through the order management service. The trading engine creates and loads a new instance of the algorithm and runs it each time it receives an update from the price feed service, or a notification of execution from the order management service, until the order is completely filled. When the conditions specified in an algorithm are met – for example, the lowest offer is below the VWAP – the trading engine sends a message (containing the order details) to the order management service, requesting that an order be sent into the market.

Each of the trading algorithms is developed and compiled into a separate module such as a .NET library (in the Microsoft world this would be a DLL). The modules are handled like add-ins that can be dynamically loaded at runtime, and run as a separate thread in the trade engine service. Each algorithm must implement a common interface that defines the mechanisms to receive input parameters and send output data – the messages sent as a response. The analytic trading engine executes and interacts with the algorithms by invoking the operations defined in the interface.

Based on the algorithm name specified in the message, the analytical trading engine finds the physical address information, invokes the operation, and runs the algorithm (see figure 3). The algorithm represents the actual trading strategy, which is identified by its name (such as VWAP, TWAP, Switch, etc).

```
ReceiveMessage(Message message)
{
1. mapping = find mapping by message.AlgorithmName
2. assemblyInstance = load assembly from mapping.Path and mapping.AssemblyName
3. algorithm = create instance of mapping.ClassName and initialise with
   message.Parameters
4. add algorithm to pool of running algorithms
5. run algorithm
}
```

Figure 3: Load and execute algorithm from message

The modular design allows for agility and flexibility in enhancing an existing algorithm or adding new algorithms.

6.2.8 Real-time prices: Price feed service

As illustrated in Figure 2, the price feed service simply subscribes to the output of the market data enrichment engine, and feeds the prices to the trading engine for processing.

6.2.9 Message routing: Order routing service

The order routing service is responsible for sending the orders received from the trading engine through the order management service to the relevant output adapters, to place the order into the market, send it to an order management/execution system, or notify the risk management or trader applications. Since the system may trade in both local and foreign equities and possibly different instrument types, the routing service is responsible for sending the messages to the appropriate trade venue through the relevant adapter. The routing service is also responsible for sending the executions report received through the outgoing adapters back to the order management service.

For each message received (see Figure 4), the routing service finds all the rules where the *message type* and the *message source* match the type field and the target field in the message respectively. There can be multiple rules for a message type coming from the same source, allowing one to multicast the message to different destinations. It evaluates the predicate, and only continues if the function returns a positive result. For each matching rule, a copy of the message is created. The target field of the copy is updated to the destination specified by the rule, and the source address is updated to the original message's target. The message is then routed to the destination.

```
RouteMessage(Message message)
{
1. rules = find rules where rule.MessageType = message.Type and rule.MessageSource = message.Target
2. if predicate = true
a. for each matching rule:
i. messageCopy = create copy of message
ii. messageCopy.Source = message.Target
iii. messageCopy.Target = rule.MessageDestination
iv. Send messageCopy to destination
}
```

Figure 4: Message routing

6.2.10 Endpoint management: Configuration service

The system can be divided into three distinct parts: the input adapters and coordination service, the trading platform itself, and the routing service with the output adapters. To support greater flexibility in possible deployment strategies and to allow for scalability, a best-practice design is to ensure that the different parts of the system remain 'black-boxes', completely autonomous from one another and any other services that may use them. The key is to abandon any assumptions about where or how the services are being hosted. They may be co-located on the same machine, or distributed across a network.

To ensure that each service remains autonomous, each service is responsible for maintaining its own configuration. All configuration information is stored in a database rather than a configuration file. Each service has its own repository. Since the services can be distributed across a network, and possibly separated by a firewall, only the service itself is allowed to query or update its repository. So the only information stored in the service's configuration file is a connection string pointing to the configuration database. From a manageability perspective, it should be possible to view, manage, and configure the overall system configuration in an integrated way. This is where the configuration service comes into play.

When the services start up, they load the configuration settings from their individual configuration repositories. New clustered nodes of the same service can easily be set up by simply copying the service to another server, without modifying the configuration file. This implies that we can dynamically expose new endpoints of the same service (see clustering

below). Using a single repository per service, we thus centralise the configuration management of each service. The repository is used to store any type of configuration setting. Since this information is not hard-coded into the service itself or stored in a XML-based configuration file, configuration changes can be made to the overall system without manually editing files and restarting the services. The configuration service defines a contract that is implemented by each component/service of the system. The configuration service is used to query and push updates to the various services. To support the dynamic configuration, any update to the individual services results in a notification being sent to the configuration service. The configuration service then notifies each of the individual services to update its binding information, keeping all the servers in a synchronised state.

6.2.11 Monitoring: Configuration service

The configuration service does failure detection against the services in the system, and receives notifications when new service instances are brought online. This not only allows it to notify other services in the system, but also allows an administrator to query the configuration service, view the various services participating in a cluster, and see which services are online or offline.

6.2.12 Load balancing / clustering: Order placement service

The architecture's support for dynamic configuration gives rise to another concept: virtualising service endpoints across clustered nodes. The centralised service repository enables one to create new clustered nodes for load balancing purposes by simply starting up a copy of the service on different machines, while pointing the connection string to the same repository.

The order management service and trading engine can be coupled together as a service set and clustered across a number of machines. Once the services have been started, the configuration service will notify the other services of the new instances. The order placement service will register the new instances and immediately perform load balancing against them. The order placement service is therefore a proxy service for the order management service. It is responsible for routing the orders received from the coordination service to an instance of the order management service, using a load balancing algorithm.

6.2.13 Auditing: Audit log service

The audit log service logs a complete history of all the transactions and events that occur on the system. To ensure guaranteed delivery, the system services do not interact directly with the audit log service. Instead, the services interact with a proxy that posts all messages to a transactional message queue. The audit log service binds to the transactional message queue from which it receives the messages. As a result, the messages are inherently asynchronous and disconnected. If the audit service is offline, the messages simply remain in the queue. The messages will be delivered and processed the next time the service starts up. This approach enables the system to offer guaranteed traceability of the history of all the messages, and to be compliant with the financial industry's strict auditing regulations.

7. DISCUSSION

To evaluate the feasibility of the proposed framework and its possible contribution towards organisational agility, the application of the proposed framework was demonstrated to a systems architect from one of the three organisations, by conceptually mapping it to their existing architecture. (Only a high level overview of this mapping is available from the authors in order to protect the organisation's core business architecture.) The researchers subsequently interviewed this system architect. Here are some of his remarks on the feasibility of the framework:

> *" … the use of standards aimed at interoperability - web services and the like - along with the focus on ensuring that adaptors for various domain-specific protocols, makes the architecture well-suited to the existing environment it has to fit in to".*

On its contribution to business agility, he said the following:

> *"The architecture promotes the creation of loosely-coupled components. This in itself should result in lowering the barrier of entry to introducing new components or re-using existing components in new and novel ways. Because of the abstraction and modular design, we will be able to quickly introduce new trading strategies, or enhance existing algorithms without affecting the system itself."* (Interview questions and results available from authors)

Although the above remarks reflect the opinion of only one system architect, they do give an indication of the general viability of the framework. However, this is a model representing a *conceptual* architecture framework that is abstracted from implementation. The focus is not on implementation, but on understanding the problem domain and how it influences and shapes the design of the services. It is recognised that further research is needed to determine the viability of this framework. One such study could be the creation of the detailed architectures, and then evaluating which technologies are most suitable for implementing the proposed model.

8. CONCLUSION

There is a growing body of literature on the potential value of a service-oriented approach. However, most research uses web services to explain SOA and how it can be used to build flexible business processes. The contribution of this research is that the focus of the proposed SOA framework is on the *technical implementation* issues, and it therefore drives the adoption of SOA from a *technology perspective*. By understanding the business context and the flow of information between the processes, the different activities were used to identify the functional elements and how they relate to each other, and not how they will be implemented. This will allow one to select the technology that will best realise the proposed architecture, and thus placing the technology in a business context so that it supports the business vision.

The framework supports interoperability and integration with other trading, risk management, or other financial applications; and it supports flexibility in terms of adding and configuring existing functionality, as well as the dynamic addition of new and modification of existing trading algorithms. It also addresses concerns such as reliability, scalability, security, and high performance.

9. REFERENCES

[1] **Moitra, D. & Ganesh, J.** 2005. Web services and flexible business processes: Towards the adaptive enterprise, *Information and Management*, 42(7), pp 921-933.

[2] **Eisenhardt, K.M. & Martin, J.A.** 2000. Dynamic capabilities: What are they?, *Strategic Management Journal*, 21(11), pp 1105-1121.

[3] **Harreld, J.B., O'Reilly, C.A. & Tushman, M.L.** 2007. Dynamic capabilities at IBM: Driving strategy into action, *California Management Review*, 49(4), pp 21-43.

[4] **Mitchell, R.L**. 2006. Morphing the mainframe, *Computerworld*, 30(5), pp 29-31.

[5] **IBM**. 2006. *Expanding the Innovation Horizon Global CEO study*. [online] Available from: http://www.ibm.com/bcs/ceostudy. Accessed 29 April 2008.

[6] **Xiaorong, X**. 2007. *Service-oriented architecture for integration of bioinformatic data and applications*. PhD thesis. University of Notre Dame, Indiana.

[7] **Pan, A. & Viña, Á**. 2004. An alternative architecture for financial data integration, *Communications of the ACM*, 47(5), pp 37-40.

[8] **Birman, K.P.** 2004. Like it or not, web services are distributed objects,

Communications of ACM, 47(12), pp 60–62.
[9] **Birman, K., van Renesse, R., & Vogels, W.** 2004. Adding high availability and autonomic behavior to web services, in: *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, May 23-28, 2004. IEEE Computer Society: Washington, pp 17-26.
[10] **Jones, S.** 2006. *Enterprise SOA adoption strategies*. C4Media: USA.
[11] **Bruner, R.F.** 1999. An analysis of value destruction and recovery in the alliance and proposed merger of Volvo and Renault, *Journal of Financial Economics, 51*(1), pp 125-166.
[12] **Smith, K. & O'Neal, E.** 2003. Bank-to-bank deals seldom add value, *ABA Banking Journal,* 2003 (December), pp 7-10.
[13] **Brown, A.W., Delbaere, M., Eeles, P., Johnston, S., & Weaver, R.** 2005. 'Realizing service-oriented solutions with the IBM rational software development platform, *IBM Systems Journal*, 44(4), pp 727-752.
[14] **Mehta, M R, Lee, S. & Shah, J.R.** 2006. Service-Oriented Architecture: Concepts and implementation', in: *The proceedings of ISECON conference,* Dallas, November 2-5, 2006. EDSIG: Dallas.
[15] **Perrey, R. & M. Lycett.** 2003. Service-Oriented Architecture, in: *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03)*. IEEE Press: New Jersey, pp 116-119.
[16] **IBM.** 2004. Service-oriented architectures and web services, *IBM Developerworks Technical Presentation*. Texas State University.
[17] **Huhns, M.N. & Singh, M.P.** 2005. Service-oriented computing: Key concepts and principles, *IEEE Internet Computing*, 9(1), pp 75-81.
[18] **Papazoglou, M.P.** 2003. Service-oriented computing: Concepts, characteristics and directions, in: *Proceedings of the Fourth International Conference on Web Information Systems Engineering*. IEEE Computer Society: Washington, pp 3-12.
[19] **Sanchez, F.** 2006. The SOA approach to integration and transformation, *U.S. Banker,* 116 (July), pp 12-13.
[20] **Margulius, D.L.** 2006. Banking on SOA, InfoWorld, July 13. [online] Available from: http://www.infoworld.com/article/06/07/13/29FEwachovia_1.html. Accessed 2 Aug 2008.
[21] **Zimmermann, O., Milinski, S., Craes, M. & Oellermann, F.** 2004. Second generation web services-oriented architecture in production in the finance industry, in: *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM Press: New York, pp 283–289.
[22] **Microsoft**. 2007. *SOA in the real world*. [online] Available from: Microsoft. http://www.microsoft.com. Accessed 2 August 2008.
[23] **Ross, J.W.** 2003. Creating a strategic IT architecture competency: Learning in stages. *MIS Quarterly Executive,* 2(1), pp 31-43.
[24] **Afshar, M.** 2007. *SOA governance: Framework and best practices*. Oracle: California.
[25] **Marks, E.A. & Bell M.** 2006. *Service-Oriented Architecture (SOA): A planning and implementation guide for business and technology*. Wiley: New York.
[26] **Zachman, J.A.** 1987. A framework for information systems architecture. *IBM Systems Journal,* 26(3), pp 276-292.
[27] **Josuttis, N.M.** 2007. *SOA in practice: The art of distributed system design*. O'Reilly Media: California.
[28] **Reekie, H.J. & McAdam, R.J.** 2006. *A software architecture primer*. Angophora Press: Sydney.
[29] **Myers, M.D.** 2009. *Qualitative research in business & management*. SAGE Publications Ltd.: London.
[30] **Bates, J.** 2007. *Algorithmic Trading: The use of algorithms in automated trading*. [online] Available from: http://www.ddj.com/hpc-high-performance-computing/197801615. Accessed 26 July 2008.
[31] **Technical Analyst**. 2007. The new standard in algorithmic trading, *The technical analyst,* 2007 (May/June), pp 40-44.