Using Software Architectures and Design Patterns for Developing Distributed Applications

Feras T. Dabous, Fethi A. Rabhi and Hairong Yu
School of Information Systems Technology and Management
University of New South Wales, Sydney NSW 2052 Australia
{f.dabous,f.rabhi,hairong.yu}@unsw.edu.au

Abstract

Although there are a large number of software development methodologies for standalone software, little effort is being paid into investigating specialised methodologies that target the development of Distributed Applications (DAs) in the era of Internet and Web-based applications. Rather than focusing on business models, developers usually spend considerable effort in implementing connectivity between software components that comprise these applications. Since a large number of competing technologies exist, these solutions face serious technology-migration and design reuse problems. This paper advocates approaching the design activity from a business rather than technological perspective by defining a service-oriented software architecture that satisfies the functional requirements in a particular domain. The paper also suggests identifying existing or new design patterns to capture common business process functionalities and fulfil the non-functional requirements. For evaluation purposes, we are applying our approach to Capital Market Systems (CMS) through the development of a prototype system using Web Service technology.

1. Introduction

A decade ago, most software consisted of standalone applications and only minimal forms of interactions through files and data oriented transfer protocols were supported. With the widespread use of the Web and its associated technologies, more and more applications that are "cooperating" over the Web are being developed. Depending on the structure and purpose of these applications, they exist under different denominations such as integrated applications, e-business/B2B applications, Web applications, etc. In the context of this paper we use the term "distributed application" to refer to this category of applications. We see the major problems in developing such applications as follows:

- Rather than focusing on business models, developers usually spend considerable effort in implementing connectivity between software components that comprise the applications. This is extremely difficult, time consuming, and require extensive programming effort. This practice leads to technology-dependant solutions. Since a large number of competing technologies exist, these solutions face serious technology migration and design reuse problems.
- Most existing systems are monolithic and have been developed many years before the emergence of new technologies such as Internet-based middleware technologies. In the business application domain, "legacy" applications were developed with little consideration for integration with other applications across networks. With the recent evolutions, these legacy applications face difficulties in joining the community of interacting applications in a distributed environment.

Several solutions have been proposed but show limitations in finding the right trade-off between the following tensions:

1. How to provide domain-related design abstractions in a way that is independent from the underlying technology? By achieving this, the design will be more stable and reusable across the application domain using different underlying technologies.
2. How do we guarantee that such high-level design abstractions guarantee some minimal "quality requirements"? By achieving this at the design phase, we can avoid costly performance and other quality problems at latter stages in the development process.

The rest of this paper is organised as follows: Section 2 describes some related work and the research problems in this area. Section 3 discusses a possible methodology that can address some of these problems. Section 4 presents

Capital Market Systems (CMS) as the selected case study and a prototype implementation description. Finally, section 5 describes ongoing work and concludes this paper.

2. Background and Problem Area

This paper is concerned with design methods for distributed and Web applications. There are two broad categories of methods. The first (and most popular) category comprises methods that are closely linked to the middleware infrastructure. In fact, they are not strictly speaking design methods but more like programming environments that facilitate particular types of DAs e.g. inter enterprise collaboration across networks. In [6], we survey middleware technologies across a three-level integration model which consists of communication, content, and business process levels. At the communication level, there are technologies that support tightly-coupled (e.g. CORBA or Java RMI) or loosely-coupled (e.g. JMS or SOAP) communication styles. A middleware technology can span over more than one level. For example, Electronic Data Interchange (EDI) supports a loosely-coupled style of interaction at both communication and content levels. XML-based frameworks also vary in their support of interoperability at the different levels. In general, they focus on interoperability at communication and content levels in the context of loosely coupled Internet-based services (e.g. document-based SOAP). Only few efforts in RosettaNet and ebXML provide support for interoperability at the business process level. Workflow-based systems focus on interoperability at the business process level in the context of tightly coupled services. However, emerging efforts in this area mainly by Microsoft and IBM such as BPEL4WS [1] — have the potential of promoting Web Service as an integration platform that will be able to integrate complex interactions between applications in the context of automated business processes.

The second category includes methods that are part of a formal software engineering methodology. Although there has been design methods specifically proposed for concurrent and distributed systems, their impact has been small in practice. Most of them target real-time applications and systems. Examples include CODARTS (Concurrent Design Approach for Real-Time Systems) [10], Octopus [2] and ROOM [27]. Object-oriented methodologies (currently dominated by UML [24]) have also been proposed for the design of DAs due to their support for modularity, flexibility and reusability. Some convergence between concurrent and object-oriented methods is happening. For example, Real-Time UML [8] describes an extension for real-time systems and COMET (Concurrent Object Modeling and architectural design mEThod) [11] is one of the first methods to explicitly target the design of distributed applications alongside real-time applications.

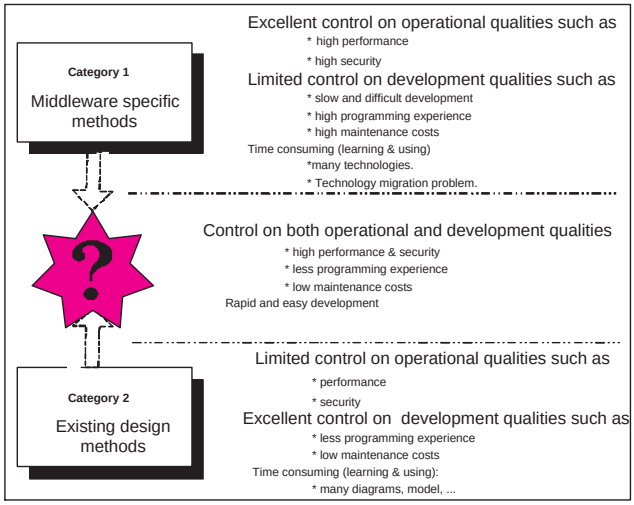

Figure 1. Approaches of developing software

When comparing the two categories, methods in the first category do well in the flexible management of DAs operational qualities such as performance and security but they are costly in terms of development qualities such as time to market (they require high programming expertise) and maintenance costs (caused by the rapid evolution of middleware technologies). In addition, the resulting designs are technology-dependent. As a consequence, migrating to new technologies encompasses reengineering most parts of the software. On the other hand, methods in the second category support development qualities such as time to market and low maintenance costs but do not have a firm control over operational qualities such as performance and security. They are of little use in practice because they do not show clear connections to existing technologies. They involve a long learning curve and require an excessive number of models and diagrams. In addition, they are more suitable for designing new systems from scratch rather than integrating existing software components into the design. This is because most existing design methodologies focus on modeling each sequential activity of a DA separately using known techniques while outside interactions are usually left much later in the design process or only addressed at the implementation phase [23]. Most of existing supporting tools are general-purpose and only fit standalone applications. These tradeoffs are illustrated in figure 1.

3. Towards new methodologies for developing distributed applications

As shown in figure 1, there is a need for methods and tools that cater a “middle-ground” compromise. An ideal

middle-ground should consider strengths and avoid weaknesses of the two major approaches discussed earlier. For example, it should facilitate control over both operational and quality requirements. This is a very challenging task since improvement on one quality attribute may affect other attributes badly. One possible approach is to utilise the concepts of software architectures and design patterns in a more systematic way to address these problems. In specific situations, a solution has to fit in a particular domain which has known legacy applications. A possible methodology can operate in two steps:

Step 1 Based on the selected domain and its functional and quality requirements; we investigate and define a suitable *service-based* software architecture (SA) that facilitates interoperability by abstracting away the underlying technology. This SA must integrate key legacy applications and business processes for that domain.

Step 2 We address quality (non-functional) requirements through the extraction and identification of candidate design patterns at different abstraction levels. These patterns are solutions to recurring problems that are across different business processes whose SA is derived from 'Step 1'.

Onces such patterns are identified and evaluated against requirements, then any DA in that particular domain that utilises one or more of these patterns is guaranteed to have qualities as prescribed in the patterns' specifications. These two steps are now explained in more detail.

3.1. Domain-Specific Service-Oriented Architecture

A service Oriented Architecture (SOA) is an approach for DAs integration where the software functionalities are encapsolated into services. These services abstract the implementation details and provide standard interfaces that can be discovered and invoked in a loosely coupled style by other applications and services [15]. An SOA conforms to incremental applications integration. It also facilitates the business managers' perspective for DAs development where the driver is the business needs and not the technology [28, 3]. In this type of architecture we distinguish two types of services which are basic (or elementary) and integrated (or composite) [22]. A basic service is entirely processed within one architectural component. The component itself is not necessarily a single application: it could be an entry point for an enterprise workflow or ERP system, but from architecture's perspective it is viewed as a single component. A basic service can typically be a legacy system with a software wrapper that makes the correspondence between the system and the service interfaces. An integrated

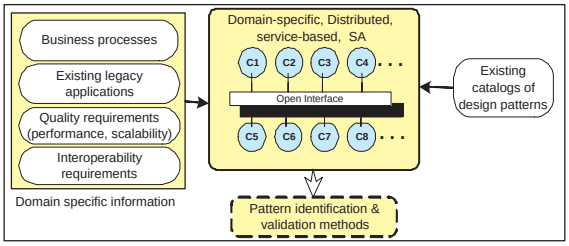

Figure 2. Domain-specific SA identification process

service is a service offered by one component, but it invokes the interaction between this component and several other components that provide "outsourced" services. The architecture is domain-specific, meaning that it offers common services to a number of business processes within a particular domain (e.g. health or finance). As illustrated in Figure 2, the process of identifying this architecture takes into account the following considerations:

- it must address the business processes associated with the domain (usually in the form of composite services)
- it must integrate existing and quality proven legacy applications in that business domain (usually in the form of basic services)
- it must as much as possible comply with interoperability and quality requirements (we can use existing catalogues of design patterns such as the GoF [9]).

3.2. Domain-Specific Design Patterns

Design Patterns (DPs) imply the fulfilment of functional and quality requirements in determining the solution of the recurring problem in a given context [9]. The "discovery" process is important since patterns are not created but architects and designers discover them by mining the knowledge and practices of expert developers who have been dealing with similar problems. The literature shows a number of patterns classifications that comprise levels of abstractions such as [4, 13, 25]. In our approach we adapt the pattern description at three different levels of abstractions which are business process, architectural and technology patterns.

At the highest level, we use the term Business Process DP (BPDP) to describe a DP that identifies the shared functionality of a number of interrelated business processes in the highest abstract level that can be understood and interpreted by most business personnel. This is very useful as a first step in bridging the gap between business and technical perspectives so that business people can participate in the

verification of the BPDP. The quality requirements are not considered explicitly at this level. At the middle level, we map the BPDP into Service-Based Architectural Patterns (SBAPs) each of which focuses on achieving the quality requirements such as performance and scalability for one or more of the identified business processes (see figure 3).

- By service-based, we mean that this pattern uses a combination of services identified in Step 1 of our approach. This combination of services is formulated in such a way that addresses the description of the BPDP.
- By architectural pattern, we mean that the pattern description is abstracted from the choice of the middleware. Therefore, incorporating different technologies will not affect this level of pattern abstraction.

At the lowest level, the SBAP is further mapped into Technology-Based Patterns (TBPs) that represent implementations using selected technologies. These technologies are selected based on the level of interoperability support offered to implement the SOA. Tradeoffs are considered since more interoperable technologies may affect other qualities such as performance. Therefore, each TBP demonstrate possible conflicting forces in achieving quality attributes while supporting interoperability. This will be illustrated in the case study.

Most of the identified basic services in the SOA are wrappers for existing legacy systems. In this case, any implementation of the described BPDP would not affect the way such legacy systems are interacting with each other and with existing customers. Such integration is normally tightly coupled and technology dependent where users may resist complete replacement. On the contrary, our method creates a new alternative of using and interacting with such legacy systems. This would attract more users and open new integration possibilities with other emerging systems that are based on a service oriented model. The challenge here is to reduce overheads of SOA abstractions on quality attributes such as performance.

Figure 3 demonstrates our proposed patterns identification process. The resulting design patterns should address recurring design problems within that particular domain. Reusability opportunities arise because business processes in a given domain share many features and functionalities but are usually implemented with different technologies. Amongst different solution patterns, there are those that offer the best compromise between different (and often conflicting) quality requirements. For example, a pattern may offer good performance at the expense of security, a second one good security at the expense of performance and a third one a compromise between both. These tensions are all well documented in the patterns literature usually in the form of conflicting "forces". The ultimate choice refers back to the stated domain-specific quality requirements. We have an

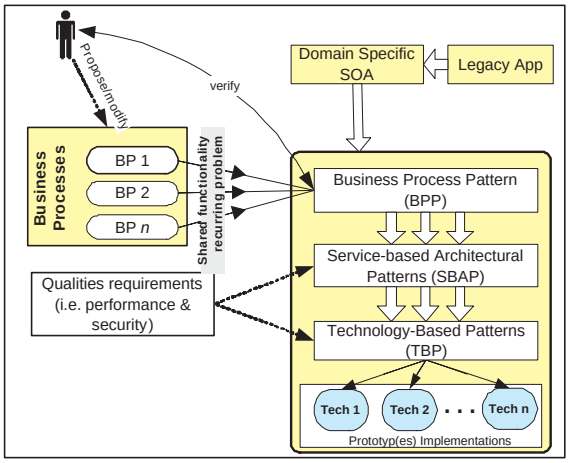

Figure 3. Domain-specific design patterns determination process

additional constraint in that all patterns must be defined in accordance to the definitions of the SA defined in Step 1. This will involve several iterations between the two steps.

The candidate design patterns at the three levels of abstractions that are identified will evolve into formal patterns though an evaluation process. This process will guarantee that the candidate patterns fulfil the interoperability constraints and other quality requirements for the business processes that they represent. As shown in Figure 4, the cycle of evaluating a candidate design pattern starts with a refinement process that checks that the business processes functional requirements are met. Then, the next refinement cycle uses one or several technologies of choice to implement a prototype. This prototype is tested and modified until interoperability and other quality requirements are met. Even though there is a number of proposed SA evaluation methods such as those in [14],we have selected prototyping because it can be used to quantitatively measure metrics for many of quality requirements especially in areas of high risk or uncertainty. Related work on skeletons and patterns reported in [20, 18, 19, 5] also uses prototyping to validate closely related methods.

In case one or more quality requirements are not met, then we need to iterate over the previous two steps. For example, the software architecture can undertake a process of transformation using existing catalogues of patterns and architectural styles. Changing the architecture to meet certain quality criteria may affect other qualities criteria so this new trade-off needs to be reflected again in the corresponding pattern. Besides being part of the evaluation process, experience in developing the prototype will enrich the corresponding pattern documentation and give crucial guidance as to how to realise middleware-specific implementations.

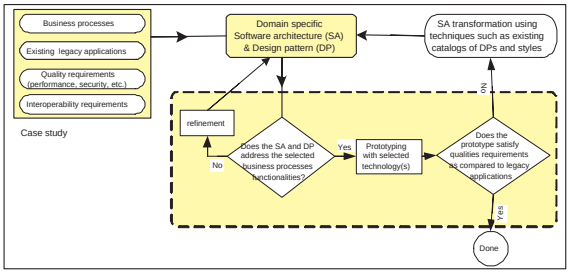

Figure 4. Domain-specific pattern validation process

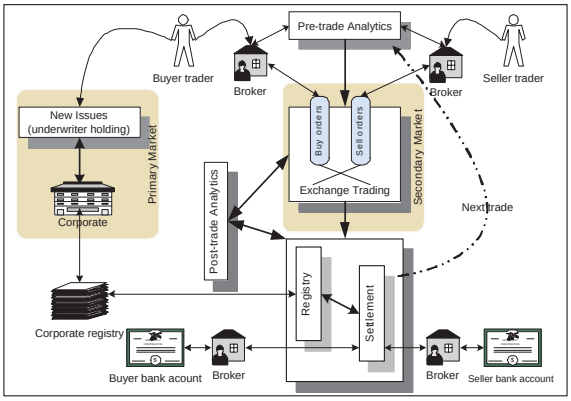

Figure 5. Overview of the trading cycle in capital markets

4. The Case Study

We apply this approach in the e-finance business domain, in particular Capital Market Systems (CMSs). CMs are places where financial *instruments* such as equities, options, and futures are traded [12]. Trading in CMs comprises a cycle of a number of phases which are: pre-trade analytics, trading, post-trade analytics, settlement and registry (see figure 5). Current work on this case study focuses on the performance as the major quality attributes in addition to supporting interoperability. In the following is a justification for the crucial importance of performance and interoperability in our domain problem.

Interoperability: Interoperability is a crucial factor for any distributed environment especially in e-business domains. In CMs, e-services are emerging with the purpose of adding extra value of automation of the business processes. Having high levels of interoperability at the infrastructure, content, and service levels facilitate the integration and the automation at the business process level. This pattern descriptions will be discussed in the context of the prototype description presented next section.

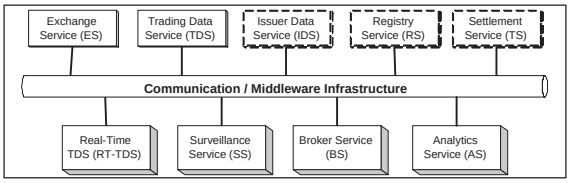

Figure 6. High-level overview of CMSs SOA

Performance: Based on our review of some legacy systems in capital markets and based on feedback from the industrial partners, performance is a crucial quality factor for the success of the automation process in capital markets trading. Historic trading data shows incremental growth on every particular capital market. This implies that the number of trading orders posted by investors and the number of trades generated are increasing rapidly. For example, the ASX had shown an average of 300K orders per day in 2000. This average has increased to about 1.5m orders and trades per day in 2002. Therefore, the ASX has conducted a major upgrade to its infrastructure [31]. The same requirement applies to the problem of researching historic trading data which has been duplicated many times over few years. Surveillance is also affected by this fact as it needs to cope with huge volumes of historic and real-time data in order to support real-time detection of illegal trading behaviour.

Other important quality attributes such as security are currently under investigation and out of the scope of this paper.

4.1. An SOA for CMSs

Figure 6 presents our preliminary SOA for CMSs [22]. Current research work extends to the identification and the integration of basic and composite services that utilise existing commercial financial systems. Our focus is on four major services. Figure 7 illustrate some of the major interactions between these services which are described in more detail next.

4.1.1 Surveillance Service

The Surveillance Service (SS) is supported by a commercial system called SMARTS [34]. Its objective is to expose some of the SMARTS surveillance system functionality such as the dissemination of real-time alerts as Web services by wrappers implemented using a technology of choice. SMARTS is a Unix/Linux based system that receives real-time transactions from the exchange trading engine, processes and correlate them with historic data as they

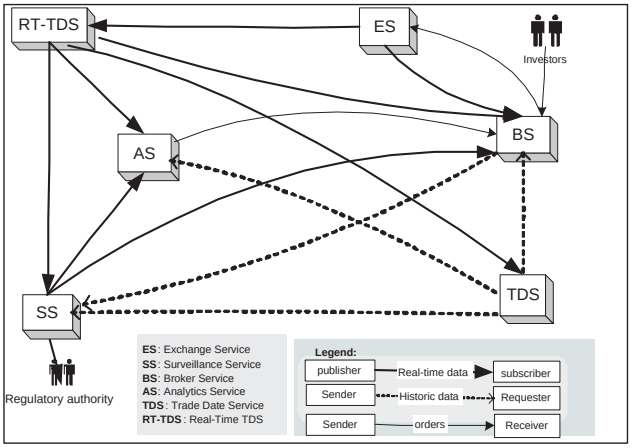

Figure 7. Connectors between major services in the identified SOA

arrive, checks against illegal trading behaviour rule (e.g. insider trading and market manipulation), and then issues and disseminates alerts (if any) to market analysts. Analysts can then utilise a number of associated tools to investigate further actions.

4.1.2 Trading Data Service

The Trading Data Service (TDS) allows access to a large repository of financial data held by SIRCA [33]. Our work on the TDS is to provide a unified way to access a number of data sources for capital market trading data. Every data source stores market data in its own format and provides its own API to access the data. Some data sources contain only quotes, trades and quotes (such as TAQ [35]), or trades and all orders (such as SMARTS and SIRCA). Most of these internal data sources are designed for optimal retrieval performance and therefore, outperform the performance traditional database management systems.

4.1.3 Exchange Service & Real-Time Trading Data Service

The Exchange Service (ES) is backed by a commercial trading platform called X-STREAM [32]. Its objective is to expose some of the X-STREAM securities exchange system functionality of placing, amending, deleting, querying orders, which are used extensively by the market brokers, as services by wrappers implemented using a technology of choice.

On the other hand, the the Real-Time Trading Data Service (RT-TDS) facilitates access to real-time market informationthe. Its objective is to provide real-time dissemination of market information to market participants. Market information could include orders, trades, quotes, etc. Market participants include brokers, market analysts, surveillance, investors, etc. X-STREAM is a Unix/Linux based system that receives real-time trading orders (bids, offers, amends, cancels, etc.) from the market makers (e.g. brokers) and processes them as they arrive. It also maintains orderbooks, applies matching rules, and keeps logs of orders and trades that are used by other CMSs.

4.2. Business Processes Identification

There are a number of interrelated Business Processes (BP) each of which covers one or more tasks within the trading cycle. CMs are constantly introducing new instruments for trading in their efforts to attract investors. Therefore, many business processes consist of the same or a subset of trading phases with some variations which depend on the instrument type can be identified. The geographical location and the difference in regulations across markets also result in variations for the same trading instrument. Each phase in the trading cycle is naturally performed in a geographically dispersed location running different legacy applications and possibly owned by different companies. These business processes that are embedded in the SOA identified in section 4.1, will form the core for identifying our BPDPs.

In this case study, we will be concerned with a simple BPDP. We have identified five business processes that have common functionality and address a recurring problem in the CMs trading. Figure 8 only shows the data flows in each of these BPs Each BP represent a minimal autonomous process of the overall trading cycle that is presented in figure 5. In the following discussion the terms “source” and “target” are referred to either another service or a simple user (e.g. through Web interface). Each source represents a possible starting data flow point for the BP while each target represents a receiving point data flow. Depending on the BP, any source or target can be a starting or receiving end of the control flow. In the following is a brief description of each of these BPs as presented in figure 8.

BP1: Orders/trades placement/notification. This BP starts with order placement by the brokers into the exchange. One a match (i.e. trade) is occurred then the corresponding brokers are notified with that trade details. The brokers are both sources and targets for this BP. Real-time trading data is also a possible target that represents the central service in BP3.

BP2: Historic trading data dissemination. This BP starts with a request for market meta-data by targets such as brokers, analytics (which represents the central service in BP5) or surveillance (which represents the central service in BP4). The query request is then satisfied by one or more historic data sources (such as

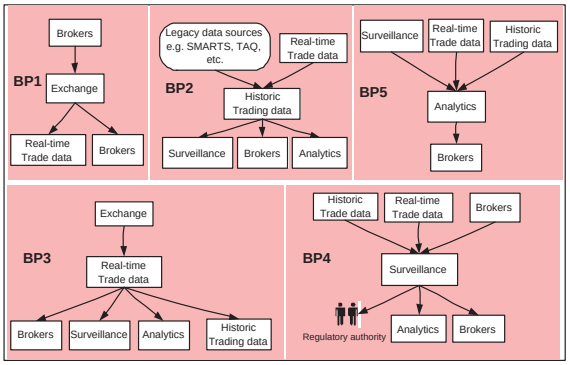

Figure 8. Illustration of data flows in the identified BPs

SMARTS or TAQ [35]) based on the specification of the query. Real-time trading data (which represents the central service in BP3) is also a possible source for current day data.

BP3: Real-time trading data dissemination. This BP is concerned with disseminating real-time market data that includes all source (i.e. exchange) events to targets (i.e. subscribers) of data. Targets includes brokers, surveillance (which represents the central service in BP4), analytics (which represents the central service in BP5), and historic data service. Each target is subscribed for events of its interest.

BP4: Surveillance alerts dissemination. This BP is concerned with disseminating surveillance generated alerts to targets such as market analysts, regulatory authorities, and brokers. The alerts are triggered based on the sources such as the exchange historic data, real-time data, and brokers data

BP5: Trading data analytics This BP is concerned with analysing market data from sources such as historic data, real-time data and surveillance alerts to support trading decisions and plans. Once a trading plan is computed, it is posted to the target (i.e. broker) so that an order can be placed into the exchange.

These business processes can be combined to form more complex business processes. For example, assume that an investor needs to sell a large volume of shares of one company. In order not to have a market impact (i.e. casue an adverse price movement), BP5 can be invoked by the broker. Once a trading plan is computed, BP1 is invoked as many times as required by the trading plan.

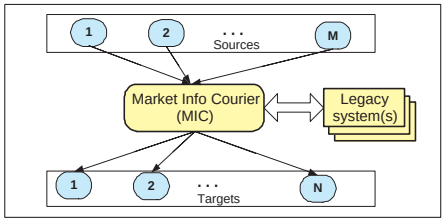

Figure 9. Overview of the MID Business Process Design Pattern

4.3. Business Process Design Pattern Description

The context of this proposed pattern is to develop DAs that are capable of receiving a diversity of market information from different sources, process this information (possibly via underlying legacy systems), and disseminate this information to an arbitrary number of observers (i.e. subscribers and/or requesters). The proposed pattern should also be in the context of loosely coupled environment because the underlying legacy systems are dispersed and mostly owned by different companies. The candidate pattern should operate across trading different instruments such as equities, options and futures and across different capital markets that have possibly different market microstructure regulations with associated legacy or distributed computer systems that are implemented using different technologies.

Figure 9 illustrates the common functionality of the BPs identified earlier. We call this candidate pattern Market Information Disseminator (MID). An essential component of this pattern is the Market Information Courier (MIC) which represents a service that has three roles: receiving information from a number of sources (services and/or customers), passing this information to the underlying legacy systems (if any) for processing, and disseminating the received and/or the processed information to a number of targets (services and/or customers). The receiving and disseminating of information from/to sources/targets are either on demand (e.g. Historic trading data) or by notification (e.g. real-time trade data and announcements). The MIC itself can be a target for data from the data sources. Therefore multi level dependencies and/or loops of Sources-MIC-Targets can be produced by instantiating any number of the identified BPs descriptions.

4.4. Service-Based Architectural Pattern

We need to refine the identified BPDP into architectural components based on the identified SOA and refine the connectors between these components to reflect the best relationships that satisfy quality requirements such as perfor-

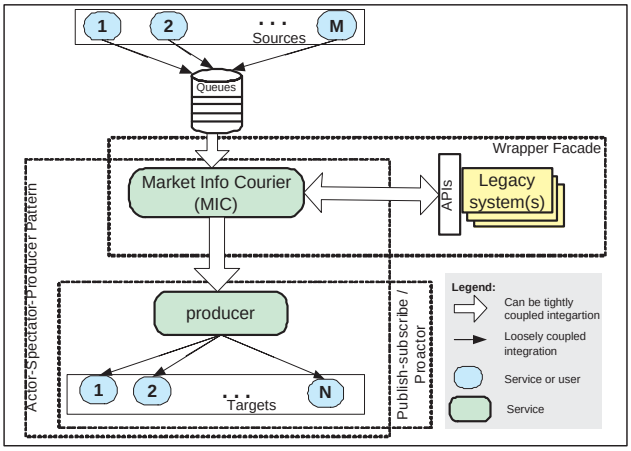

Figure 10. Overview of a Possible Service-Based Architectural Pattern

mance. We identify three types of relationships presented by the connectors: publish-subscribe push model (e.g. RT-TDS trades dissemination to SS), requester-sender or pull model (e.g. AS requesting historic information from TDS), and sender-receiver (e.g. BS submits trading order to ES). Each identified BP descriptions is to be refined from the BPDP descriptions into an SBAP that describes the appropriate connectors styles and legacy systems. Although a number of different SBAPs can be identified, the scope of this paper is to address only one SBAP. This pattern addresses real-time constraints which are clear in most parts of the identified BPs. For example BP1, BP3, and BP4 should satisfy real-time constraints that vary based on the type of the relationship between connectors (see figure 7) and the quality attribute of each BP.

In order for this pattern to address these qualities, we extensively use existing design patterns catalogues. The original descriptions for such patterns are mostly specified in the context of tightly coupled object oriented environment. However, we extend their usage to address distributed services interactions in a loosely coupled context. Figure 10 presents an overview of this pattern that combines a number known patterns: Publish-Subscribe [9], Actor-Spectator-Producer [21], Wrapper Facade and Proactor [26]. The core issue in this pattern is to maximise the performance of the MIC by minimising the connection coming to and out of it. To achieve this, our work in progress is to use queuing concept to limit the entry points from sources ans we use the concept of producer [21] to limit the exit points to targets.

4.5. Technology-Based Patterns

These patterns refine the identified SBAP into implementation descriptions. A number of TBP catalogues are emerging such as [17] that address best practices based on a particular technology. Such patterns descriptions could vary from one technology to another based on their support for interoperability. A compromise with other qualities in particular the performance can be achieved by following this simple strategy for every BP:

- If interoperability is more important, then choose the most loosely coupled technology that satisfies the minimum requirements for performance (i.e. that can accommodate the expected transactions rates).

- If performance is more important, then choose the best performance technology (highest transactions rate) that satisfy the minimum interoperability requirements.

Therefore, we need measures (metrics) for performance and interoperability:

- Performance: mainly throughput and latency overhead as compared to transactions rates experienced in the real situation and as compared to the legacy systems performance.

- Interoperability: Many studies have ranked technologies based on their interoperability support e.g. [16] in addition to our work in [6]. Moreover, the prototype development process will demonstrate the integration at the service level as compared to the current add-hoc practices of point-to-point legacy system integration.

There have been several attempts to implement infrastructure for an SOA vision such as CORBA, J2EE and DCOM. Recent literature argued that these technologies are not successful enough for heterogeneous applications integration mainly because they support tightly-coupled integration using complex binary wire protocols and different interface languages. On the other hand, Web Services are currently the most promising implementation platform for SOA mostly because they support loosely-coupled integration using a universally accepted and vendor neutral XML-based interfaces and messages over the simple wire protocol SOAP that can run over the universally accepted Internet. A TBP can be identified for each of these technologies. The technology selection would affect the way the SBAP is mapped into TBP. For example, some technologies support issues like session handling and queuing while others need extra programming effort to do similar things.

4.6. Prototype Implementation

Our ongoing work on prototyping exposes services offered by commercial legacy systems through wrappers.

Therefore, this section provides discussion on how to satisfy interoperability requirements through proper technology selection. It also discusses implementation issues of the four major services and performance evaluation of their prototypes in the context of the identified BPs.

4.6.1 Configuration and technology of choice

In this paper, we describe one TBP prototype using Web Services which is most likely to be the future for services implementation in our domain case study. A number of SOAP implementations are available from different vendors or as open source. We have used two Web Services implementation: C based (gSOAP [29]) and Java based (Apache Axis [30]). The gSOAP toolkit has been selected as one technology for the implementation of the pattern architecture for the following reasons: gSOAP is implemented in C++ and provides a C/C++ binding, and therefore we use it to integrate C/C++ legacy systems such as the SMARTS for the SS and the X-STREAM for the ES. gSOAP does not suffer from the performance problems associated with Java-based implementations. gSOAP also provides a transparent API as its compiler generates stubs interfaces and thus providing more interoperable solutions [29]. We also use the RPC-style Web services because it is a mature style and supported by all SOAP implementations. Java-based approach has been used for implementing Trade Data Service (TDS) to prove to our business partners that concept of Web Services abstracts the implementation details. The performance evaluation of the implemented services in this case study considers two major metrics: latency (round-trip time) and throughput (transactions rate) of the Web services correlated with the SOAP message size and compared to those for the legacy code itself.

4.6.2 Services Performance Evaluation

Surveillance Service: Performance evaluation findings about this service has been published in [7]. It suggested two recommendations. The first one is that the latency of SOAP marshalling, binding to HTTP at the requester side and vise versa at the service side is insignificant when the legacy code invokes a computationally-intensive process such as processes that access large volumes of trading data over a certain period of time. The second one is that SOAP's performance overheads should not be considered a bottleneck for services that are not performance demanding such as real-time alerting services.

Trading Data Service: Preliminary evaluation findings have shown significant enhancement to existing retrieval procedures which in many cases (e.g. SIRCA and SMARTS trading data) incorporate the intervention of a programmer to extract complex analysts' requests. Currently, this service is being deployed in the real context to be available for use by market researchers and analysts. Our ongoing work is to finalise this service and to have comparable performance measures.

Exchange Service: We have conducted comprehensive benchmarking testing mainly on throughput and hardware utilization for different market types. Our ongoing work is to investigate the performance overheads of the wrappers and the Web services applicability for a number of CMs based on up-to-date historic trading data. The result of this performance evaluation and benchmarking will be published soon. Currently, this service is being deployed as a teaching tool together with trading agents software for finance students so they can have easy and simple access to a real exchange system.

Real-time Trading Data Service: This service implementation is an ongoing work as it incorporate other issues such publish-subscribe push model. This issue is currently not supported by Web Services implementations. However, we are achieving this model by having two services interacting with each others.

5. Conclusion

The world of software development is witnessing tremendous new developments. It is moving away from traditional software engineering methods towards technology-specific methods due to pressures such as fast time to market, stringent quality requirements, the need to integrate legacy applications and the need to support interactions over networks. New approaches such as service-oriented architecture and design patterns provide solutions but their use is still ad-hoc and detached from known software engineering design principles.

This paper investigates an approach that addresses the limitations of existing methodologies especially in the case of distributed applications. First, a service-oriented architecture captures existing legacy systems functionalities in the form of basic services and business processes in the form of composite services. Then a set of design patterns based on this architecture are identified. Their main role is to address both functional and quality requirements. The key idea behind patterns is not to produce new techniques (as much of the existing literature in this subject does) but to provide a framework for the reuse of well-proven solutions that have addressed similar concerns (possibly from different domains). Finally, there is an evaluation process which provides useful feedback for enhancing the architecture and associated patterns (particularly in the patterns implementation part).

Future work will concentrate on identifying other patterns in this domain and applying this approach to other domains. We are also looking forward to the provision of supporting tools especially for defining and generating technology-specific implementations. Such tools can assist on developing quality guaranteed distributed applications for the domain of choice based on the set of identified patterns and their possible configuration.

Acknowledgment

The authors would like to thank the Capital Markets Cooperative Research Center (CMCRC) (www.cmcrc.com) for its financial support. We would also like to thank our industrial partners SMARTS [34] and Computershare [32] for providing access to their systems.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Lui, D. R. D. Smith, S. Thatte, I. Trickovic, and S. weerawarana. Business process execution language for web services (BPEL4WS) specification. *http://www.siebel.com/bpel*, 2003.

[2] M. Awad, J. Kuusela, and J. Ziegler. *Object-Oriented Technology for Real-Time Systems*. Prentice Hall, 1996.

[3] J. Bloomberg. Service-oriented architecture: Why and how? *http://www.zapthin.com*, 2003.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[5] H. Cai. *A Skeleton-Based Approach for the Design and Implementation of Distributed virtual Environments*. Phd, Dept. of Computer Science, University if Hull (UK), 2002.

[6] F. T. Dabous, F. A. Rabhi, P. K. Ray, and B. Benatalla. Middleware technolofies for b2b integration. *The International Engineering Consortium (IEC) Annual Review of Communications*, 56, 2003.

[7] F. T. Dabous, F. A. Rabhi, and H. Yu. Performance issues in integrating a capital market surveillance system. In *Proceedings of the 4th International Conference on Web Information Systems engineering*, Rome, Italy, Dec. 2003.

[8] B. P. Douglass. *Real-Time UML*. Addison Wesley, 2nd edition, 1999.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Object Oriented Software*. Adison-Wesley, 1995.

[10] H. Gomaa. *Software Design Methods for Concurrent and Real-Time Systems*. Addison Wesley, 1993.

[11] H. Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison Wesley, 2000.

[12] L. Harris. *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2003.

[13] R. Helm. Design patterns keynote address. In *OOPSCA95*. ACM, 1995.

[14] R. Kazman, M. Klein, and P. Clements. Evaluating software architectures for real-time systems. *Annals of Software Engineering*, 7:71–93, 1999.

[15] J. McGoven, S. Ambler, M. E. Stevens, J. Linn, V. Sharan, and E. Jo. *Practical Guide to Enterprise Architecture*. Prentice Hall, 2003.

[16] B. Medjahed, B. Benatalla, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. B2B interactions: Issues and enabling technologies. *The VLDB Journal*, Apr 2003.

[17] T. J. Mowbray and R. C. Malveau. *CORBA Design Patterns*. John Wiley and Sons, 1997.

[18] B. O. Osaba. *A Skeleton-based Parallel Multigrid Implementation using BSP*. Phd, Dept. of Computer Science, University of Hull (UK), 1999.

[19] M. E. Outram. *An Environment for Parallel Fault-tolerant Systems Design*. Phd, Dept. of Computer Science, University if Hull (UK), 1999.

[20] P. J. Parsons. *Generating Paraller programs from Paradigm-based specifications*. Phd, Dept. of Computer Science, University of Hull (UK), 1997.

[21] F. Rabhi and T. Devillers. A software architecture for distributed virtual environments: the actor-spectator-producer pattern. In *Proc. The First Asian-Pacific Pattern Languages of Programming Conference (KoalaPLoP'2000)*, RMIT, Australia 2000.

[22] F. A. Rabhi and B. Benatalla. An integrated service architecture for managing capital market systems. *IEEE Networks*, 1, 2002.

[23] F. A. Rabhi, H. Cai, and B. C. Tompsett. A skeleton-based approach for the design and implementation of distributed virtual environments. In *5th International Symposium on Software Engineering for Parallel and distributed Systems*, pages 13–20, Limerick, Ireland, Jun 2000. IEEE Computer Society Press.

[24] J. Rambaugh, G. Booch, and I. Jacobson. *The Unified Modelling Language Reference Manual*. Addison Wesley, 1999.

[25] D. Riehle and H. Zullighoven. Understanding and using patterns in software development. *Theory and Practice of Object Systems*, 2(1):3–13, 1996.

[26] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2000.

[27] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modelling*. John Wiley & Sons, 1994.

[28] A. O. Toole. Web services-oriented architecture: The best solution to business integration. *http://www.capeclear.com*.

[29] R. A. van Engelen and K. A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *Proceedings of IEEE CC Grid Conference*, 2002.

[30] Apache axis web services. *http://ws.apache.org*.

[31] Australian stock exchange (ASX): An emerging financial hub powered by IBM technology, an IDC e-business case study. *http://www-8.ibm.com/e-business/au/case_studies/pdf/ASX_Case_Study.pdf*, 2002.

[32] X-STREAM system. *http://www.computershare.com.au*.

[33] SIRCA research center. *http://www.sirca.com.au*.

[34] SMARTS surveillance system. *http://www.smarts.com.au*.

[35] Trades and quotes markets data (TAQ). *http://www.nyse.com*.