Software Architecture Coupling Metric for Assessing Operational Responsiveness of Trading Systems

Claudiu VINȚE
Bucharest University of Economic Studies, Bucharest, Romania
claudiu.vinte@ie.ase.ro

The empirical observation that motivates our research relies on the difficulty to assess the performance of a trading architecture beyond a few synthetic indicators like response time, system latency, availability or volume capacity. Trading systems involve complex software architectures of distributed resources. However, in the context of a large brokerage firm, which offers a global coverage from both, market and client perspectives, the term distributed gains a critical significance indeed. Offering a low latency ordering system by nowadays standards is relatively easily achievable, but integrating it in a flexible manner within the broader information system architecture of a broker/dealer requires operational aspects to be factored in. We propose a metric for measuring the coupling level within software architecture, and employ it to identify architectural designs that can offer a higher level of operational responsiveness, which ultimately would raise the overall real-world performance of a trading system.

Keywords: *Software Architectures, Quality and Coupling Metrics, Trading Systems, Operational Responsiveness, Service-Oriented Architecture (SOA), Message-Oriented Middleware (MOM), Cloud Services*

1 Introduction

Over the past two decades, the electronic trading systems have transformed fundamentally the way capital markets work. From the perspective of a broker/dealer institution, that offers direct market access globally to a variety of stock exchanges, and *over-the-counter* markets (OTC), their generically called trading system have been in a continuous race. It is a tough competition to respond exceptionally quick to the investors' needs or, even better, to anticipate them, and to meet, equally agile, the intrinsic exigencies required by the ever-evolving financial market place. Trading architectures are a critical capability in capital markets. The keys to achieving high performance in trading are to have clear objectives of what the trader or the investor want to accomplish on a trading desk, and work upon an integrated design before selecting the underlying technology.

Ideally, a broker/dealer would need only one platform for order and execution management. However, reality is more complicated than that, and it depends of multitudes of factors:

- how broker/dealer firm is organized [1];
- where in the process the trading decision starts - with the portfolio managers or on the trading desk;
- which asset classes are traded, and in which region of the world they are executed;
- what kind of clients (investors) are targeted (institutional, wealthy individual, individual, all), and the volumes foreseen to be handled.

One-size-fits-all platforms may provide economies of scale, but that does not make them necessarily scalable, and they may also miss out on the subtleties needed to execute at the best price in local markets [2].

Normally, in-house traders need order management systems that support their investment strategies, capable of routing orders to traders with the right level of specificity, but having multiple platforms for handling different flows drive up technology costs. Ultimately, there has to be reached a balance between the number of platforms that enable efficiency, and the cost to support and maintain those platforms.

In general business terms, *operational responsiveness* is the ability of business processes and systems to respond to changing conditions and customer interactions as they occur, enabling business leaders to capitalize on opportunities, drive greater efficiencies, and reduce risk [3].

When it comes to trading architectures, traditionally the focus has been on *operational reliability*. That implies powerful, secure, user-friendly software architectures and easy access. It also means having failure-tolerant systems, backup systems, and efficient implementation of system upgrades (releases) [2].

A set of requirements must be specified to guarantee the desired level of quality for the software

architecture on which a trading system is built. In practice, the technical requirements are stated in the *service-level requirements* of an exchange. These include:

a) *Integrity* – security is necessary to maintain data integrity. Huge projects and investments are made in this area so that en exchange can offer the highest degree of security;
b) *Fault tolerance* – trading architecture's tolerance against failure has to be continually checked (e.g. process fail-over, recovery procedures);
c) *Disaster recovery* – facilities must be put in place and exchange teams must be trained to react quickly and flexibly. There must therefore periodically exercise recovery procedures;
d) *Recoverability* – this means data integrity (storability) for system data and messages;
e) *Availability* – this measures the total time that a trading architecture implementation is operationally available for trading; normally, broker/dealer trading architectures must comply with rigors the operational availability requirements imposed to electronic exchanges; for example, the electronic exchanges in Europe must be, and have been, available well over 99.998% of normal working hours per year; that is, they have been down less that 0.002% of the time, which translates into only a few minutes per year [2];
f) *Volume capacity* – this is given in terms of maximum number (a certain percent plus contingency) per day for trades, orders, quotes, and trade reports, along with maximum number of traded instruments (both liquid and illiquid); capacity requirements must reflect both average and peak load numbers for orders per unit of time, both aggregate and per instrument traded;
g) *Scalability and functional expandability* - of critical importance is the scalability of the aforementioned features within a reasonably short period of time; scalability is essential to support further volume growth; functional expandability refers to ability to add and integrate new functional components within the architecture;
h) *Response time* – this is the order request response time and message sequencing in seconds for average and peak broadcast time;
i) *Portability* – in operation, the architecture modules need the ability to provide transparency regarding hardware platform and operating system, physical location, access to resources and the way they may be relocated; using standards where possible is always recommended;
j) *Maintainability* – a simple software architecture which is easy to maintained and at low cost is always desired; however, reduced complexity does not mean simplicity;
k) *Auditability* – ability to meet the demands of regulatory authorities and other auditors.

Additionally, there are overriding criteria regarding the interaction with the clients (investors) of a broker/dealer, including equal treatment, transparency, immediacy, and low cost. These criteria are interdependent. Each is a necessary but not a sufficient condition for market integrity. The different weightings given to these factors in an ever-changing market environment are decisive for the success of a brokerage firm that is striving to attract, to maintain, and to enhance services to their clients at reasonable cost [4].

The empirical observation that motivates our research relies on the difficulty to assess the performance of a trading architecture beyond a few synthetic indicators like response time, architecture expected latency, availability or volume capacity.

Trading systems are complex architectures of distributed resources. However, in the context of a large brokerage firm, which offers a global coverage from both, market and client perspectives, the term *distributed* gains a critical significance indeed.

Offering a low latency ordering system by nowadays standards is relatively easily achievable, but integrating it in a flexible manner within the broader information system architecture of a broker/dealer requires operational aspects to be factored in.

In this paper, we intend to identify the chief trading architecture models that have been, and are being used in the industry, and to analyze comparatively their strengths and weaknesses.

In order to support our trading architecture analysis, we propose a metric for measuring the coupling level within software architecture, and employ it to identify architectural designs that can offer a higher level of operational responsiveness, which ultimately would raise the overall real-world performance of a trading system.

2 Trading Architecture Approaches and Their Characteristics

Traditionally, the components of a trading system architecture have mimicked the corresponding

departments that function within a broker/dealer firm:

- *front office* - responsible with the order placement and routing (separate flows for client, and proprietary orders), order management, and exchange execution capture;
- *middle office* - the place where the exchange executions are processed, and the actual trades a generated;
- *back office* - responsible for client confirmations and where the settlement date is prepared for being sent to clearing agencies, trust bank and other third parties.

The information technology correspondents are illustrated in Figure 1 above:

a) trading applications, GUI based, and their associated programming interfaces;
b) order management system (OMS);
c) dedicated lines to various securities exchanges;
d) trade management system (TMS);
e) client confirmation system (CCS);
f) settlement/clearing system (SCS).

Depending of the approach, the execution management system (EMS - responsible for exchange execution capture chiefly) may be part of OMS or have a distinct representation.

In addition, there are normally necessary modules for market data feeds and their dissemination to the clients and trading desks (prices, companies and other financial instruments updates etc.).

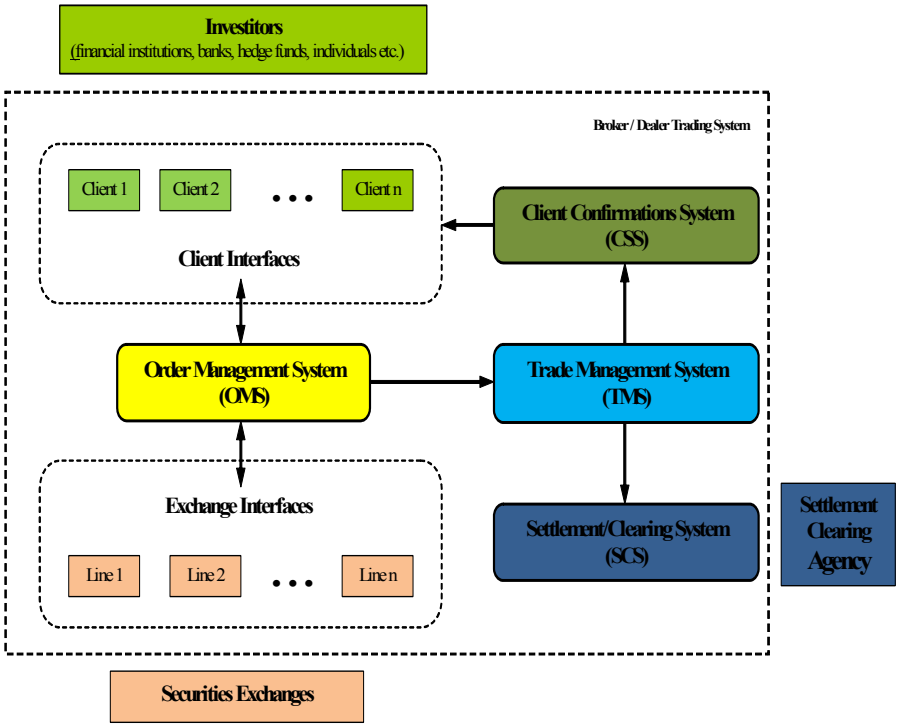

Fig. 1. The main functional components of a dealer/broker trading system

However, these modules dare not part of main trading flow, and we shall not consider them for the purpose of our analysis. Moreover, in many cases real-time market data is provided on a contract basis directly by exchanges (see NASDAQ OMX Global Data Products, for instance [5]) or by specialized firms with global market coverage, like Reuters [6].

It is worth pointing out that, in relation to the external information systems that a brokerage firm needs to interconnect with (remote clients, exchanges, trust banks, clearing agencies etc.) [7], the technology that is to be employed may differ considerably, depending on factors like:

- security measures required for privacy protection of the sensitive data;
- data volume, and frequency of message exchanges;
- data transfer speed necessary for orders to reach the exchange, and the response time for acknowledgments;
- available industry standards and common practices among market participants;
- novelty and accuracy of the market data (see aspects related to price discovery);

We will focus our analysis on the architectural solutions that are to implement the main functional modules presented above into an integrated distributed trading system [8].
The designs that we consider are the followings:

a) *tightly coupled* architecture with *point-to-point* synchronous communication model between components;
b) *loosely coupled* architecture based on an *order request broker* (ORB);
c) *service-oriented architecture* build upon a message oriented middleware (MOM).

The tightly coupled software architecture, as illustrated in Figure 2, implies that each component of the architecture is directly interfaced, and connected to the component that needs to exchange data with [9].

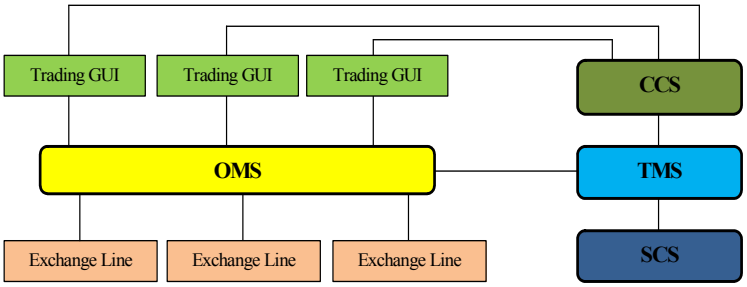

Fig. 2. Tightly coupled trading architecture, based on P2P communication model

This approach is characterized by following aspects:

- fast and straightforward data processing flows;
- the inter-process communication relies on *point-to-point* model (P2P), using TCP/IP sockets;
- employing TCP/IP connections between architectural modules, the communication mechanism is one of a synchronous nature, which implies blocking of the requester processing flow until the reply is received from the service provider; difficult vertical integration of architectural components [10];
- potentially, specific, dedicated *application-programming interfaces* between each pair of distinct modules that make up the software architecture;
- *tightly coupled modules*, due to the intimate manner they are interconnected;
- scalability, functional expandability and maintainability are difficult and costly to achieve;
- failure of a single component may affect multiple processing flows or even bring to a halt the entire trading system;
- complex and potentially unreliable recovery procedures.

The key architectural feature of a trading architecture designed upon an *order request broker* (ORB) is the presence of a dedicated communication component, a *network daemon* (ND) that facilitates data interchange among the architecture processing modules (Figure 3).
An ORB based architectural model has the following characteristics:

- the communication may rely on both TCP and UDP internet protocols, meaning that data may be exchanged between modules using *broadcasting* or *multicasting* mechanisms, along with *point-to-point* connections [10];
- *loosely coupled modules* of the software architecture; they are not directly interconnected, allowing for dynamic scaling of the system, depending on the actual load, and processing capacity of each component;
- the communication layer is separated from the business logic of the applications, and isolated in the *network daemon* (ND); any changes regarding communication are done in a single place [11];

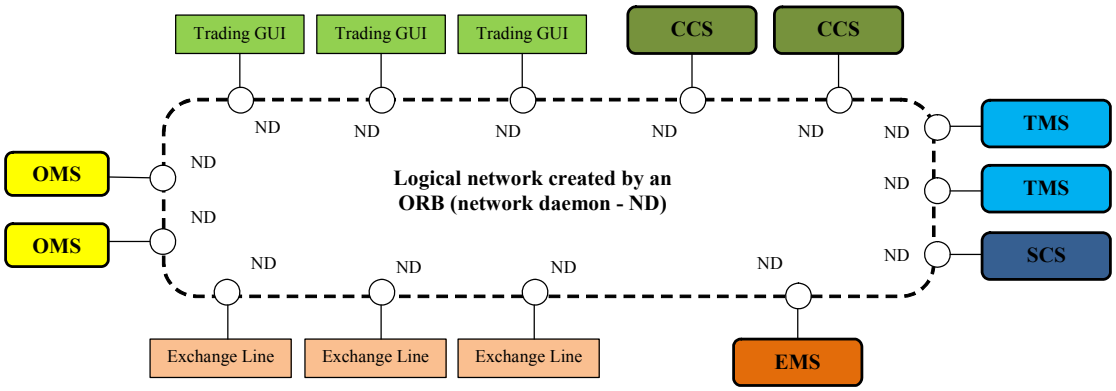

Fig. 3. Loosely coupled trading architecture built upon an ORB

- the applications are to *subscribe* to a set of distributed and shared objects they are interested in, and the *network daemon* ORB provides the delivery mechanism for objects from the application which produce (*publish*) them to the consumer;
- the *network daemon* deployment, and the management of ND versions used on different hardware can be problematic and potentially fault-prone;
- the software architecture is generally flexible and operationally responsive, but there is one major disadvantage: the distributed and shared objects handled by the ND are not normally persisted; there are mechanisms in place to retransmit lost packets at request, but a hardware component failure may generate complicated recovery procedures or even definitive data lost [12];
- the usage of UDP sockets may restrict to logical network area of the architecture to a local network (LAN), due to presence of switches and routers, which makes this architecture not immediately suitable for globally distributes operations [13].

The *service-oriented architecture* (SOA) build upon a message-oriented middleware (MOM) has the following characteristics (Figure 4):

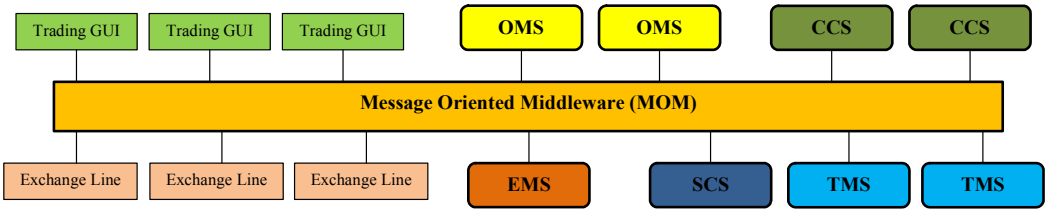

Fig. 4 Service-oriented trading architecture based on a MOM

- each component of the architecture exposes its functionality, as a service provider, to the other components [14];
- requests for services and replies are flowed through a *message-oriented middleware* (MOM). A *message-oriented middleware* makes use of a message provider (message broker) to mediate the messaging operations; in this parading, the elements of a MOM-based architecture are the client applications, the messages, and the message provider; under the broad umbrella of client applications, can be in fact identified certain applications that functionally play the role of a client, and others that have the functional role of a server; all the architecture modules are perceived as clients of the MOM message broker [15]; within a MOM-based system, a client makes an API call by sending a message to a destination managed by the message provider; the call triggers message provider services to route and deliver the message to the consumer; once the message was sent, the producer can continue the processing flow, relying on the fact that the message provider retains (persist) the message until a consumer component is available to process it;
- architecture with loosely coupled components; the implementation of such a software architecture can continue to function reliably, without downtime, even when individu-

al components or connections fail; the client applications are consequently effectively relieved of every communication issue, except that of sending, receiving and processing messages [16];
- distinct messaging patterns, or domains such as *point-to-point* messaging and *publish/subscribe* messaging;
- facilities for synchronous and asynchronous message receipt;
- support for reliable message delivery; messages can be persisted and the delivery guaranteed by MOM design and implementation;
- common and generic *application-programming interface* (API) regarding the message exchange within the software architecture, see *Java Message Service* (JMS) for instance which provides support for common message formats such as text, byte and stream [17];
- unlike an ORB based architecture, the reliance on a MOM provides no restrictions regarding remote accessibility of services, it is a truly wide area network (WAN) solution;
- multiple, and regionally distributed message providers can be interconnected in *clusters of message brokers*, offering natural support for cloud computing.

3 Proposed Model for Software Architecture Coupling Metric

The ability of software architecture to respond to ever-changing operational conditions is an attribute that summons multiple quality characteristics, perceived by the user of the software solution, and that have to be foreseen and taken into account by the designer.

Each software architecture model that we have presented above has its particular strengths and weaknesses.

In practice, the most important impact on an architectural model operational responsiveness is given by the manner in which modules are interconnected with each other, and the business requirements for data flows between them.

The way software components are coupled within an architectural model determines how the design is implemented, maintained, expanded etc.

For example, the P2P model may offer direct and fast connections but, in operation, the flexibility to scale it dynamically is very limited. Moreover, extending such a model, by adding a new functional component, usually requires a *new application-programming interface* (API) to accompany it, in order to facilitate the integration within a given architecture.

There have been proposed software metrics for measuring the complexity of a software function based on the information flows that come in and go out of it. Henry and Kafura proposed software structure metrics based on information flow in 1981 [18]. The metrics that they introduced measure complexity as a function of *fan-in* and *fan-out*:

- *fan-in* of a procedure is define as the number of local flows into that procedure plus the number of data structures from which that procedure retrieves information;
- *fan-out* is defined as the number of local flows out of that procedure plus the number of data structures that the procedure updates.

Such an approach targets the measure of complexity at software function level. At software architecture level, there is no utility in going to the details of the flows between modules, but rather to take into account the nature of coupling, and the number of *application-programming interfaces* that are to be employed.

For evaluating the coupling level of a software architecture, and, consequently, building an objective basis for assessing the operational responsiveness of a trading system implemented upon it, we propose a model to measure the architectural aspects related to coupling.

We define the following elements for constructing a metric for evaluating software architecture level of coupling:

- i – number of distinct *application-programming interfaces* (API) required by software architecture;
- m – number of distinct modules that constitute software architecture;
- I – overall number of API instances employed by software architecture;
- M – overall number of modules that constitute software architecture;

Making a parallel with Halstead's model for assessing the complexity of a software program [19], and identifying interfaces as operators and the modules that make up the architecture as operands, we define the vocabulary (lexicon) of a software architecture as being:

$$AL = i + m \quad (1)$$

Going further, the size of software architecture is given by the following relation:

$$AS = I + M \quad (2)$$

There has to be noted that for distributed software architecture we need to impose the following conditions:

$i_k \geq 1,\ k = \overline{1,m};\ i \geq 1;\ I \geq 1;\ m \geq 2;\ M \geq 2$ (3)

Then, the following relation gives the calculated architecture dimension:

$$\widetilde{AS} = i \log_2 i + m \log_2 m \quad (4)$$

Volume of software architecture is calculated using the following relation:

$$AV = AS \times \log_2 AL = (I + M) \times \log_2(i + m) \quad (5)$$

Therefore, the difficulty to implement the software architecture is estimated by:

$$AD = \frac{i}{2} \times \frac{M}{m} \quad (6)$$

Having stated the above elements, the effort to build the software architecture is given by the following relation:

$$AE = AV \times AD \quad (7)$$

In addition to the above measures, we introduce the following metric for computing the average level of coupling for a software architecture, as being:

$$\overline{ACL} = \frac{\sum_{k=1}^{m}[i_k \log_2(1+ c_k)]}{\log_2 M} \quad (8)$$

where:

- i_k – number of distinct *application-programming interfaces* (API) that module ***k*** needs for being integrated within software architecture;
- c_k – number of modules of type ***k*** employed by software architecture

$\overline{ACL}$ has the minimum value when there is considered a software architecture with two distinct modules, and one *application-programming interface* employed for interconnecting them, see the above condition (3):

$$\overline{ACL} = \frac{\sum_{k=1}^{2}[i_k \log_2(1 + c_k)]}{\log_2 2} = \frac{1 \log_2(1+1) + 1 \log_2(1+1)}{\log_2 2} = 2$$

This minimum value of the average level of coupling for software architecture, gives a good approximation for an empirical observation: that a software architecture needs at least one *application-programming interface* to glue its modules, and one API to communicate with the outside world.

We will compute these metrics for each of the trading architectures presented above, and interpret the results from the operational responsiveness standpoint.

4 Computational Results Using the Proposed Metrics

We begin the computation of the proposed metrics considering the minimal configuration for a trading architecture: 6 modules, and a single instance per module (see Fig. 1).

Table 1 synthesized the number of distinct *application-programming interfaces* (API) per module of the software architecture. These numbers represent the $\boldsymbol{i_k}$ values required for computing the average level of coupling, $\overline{ACL}$.

In Table 2 are presented the proposed metric values for the minimal (reference) configuration of a trading architecture. For ORB and MOM based architectures, beyond the difference previously identified, we consider that the number of distinct interfaces required is the same.

The average coupling level for P2P based architecture reflects the tightly couple nature of this model.

Table 1. Number of distinct *application-programming interfaces* per module

No	Distinct module of software architecture	P2P based	ORB based	MOM based
1.	Trading GUI	2	1	1
2.	Exchange line	2	2	2
3.	OMS – Order Management System	3	1	1
4.	TMS – Trade Management System	3	1	1
5.	CCS – Client Confirmation System	2	1	1
6.	SCS – Settlement/Clearing System	2	2	2
7.	EMS – Execution Management System	3	1	1

Table 2. The reference configuration with only 6 distinct modules

No	Software architecture characteristics	P2P based	ORB based	MOM based
1.	i – number of distinct APIs	8	3	3
2.	m – number if distinct modules	6	6	6
3.	I – overall number of APIs	8	3	3
4.	M – overall number of modules	6	6	6
5.	AL – vocabulary	14	9	9
6.	AS – size	14	9	9
7.	$\widetilde{AS}$ – calculated size	39.5098	20.265	20.365
8.	AV – volume	53.3036	28.5291	28.5291
9.	AD – difficulty	4	1.5	1.5
10.	AE – effort	213.2144	42.7937	42.7937
11.	$\overline{ACL}$ – average coupling level	5.4159	3.0948	3.0948

Then, we increase the overall number of components, but preserve the number of distinct ones. With other words, keep the same vocabulary of the software architecture, but increase its size from 6 to 35, as follows:

- 20 Trading GUI
- 4 Order management systems
- 4 Exchange lines
- 4 Trade management systems
- 2 Client confirmation systems
- 1 Settlement/Clearing system

The results are showed in Table 3 below.

Table 3. Increased only the overall number of architectural modules from 6 to 35

No	Software architecture characteristics	P2P based	ORB based	MOM based
1.	i – number of distinct APIs	8	3	3
2.	m – number if distinct modules	6	6	6
3.	I – overall number of APIs	86	40	40
4.	M – overall number of modules	35	35	35
5.	AL – vocabulary	14	9	9
6.	AS – size	121	75	75
7.	$\widetilde{AS}$ – calculated size	39.51	20.27	20.37
8.	AV – volume	447.75	237.74	237.74
9.	AD – difficulty	23.33	8.75	8.75
10.	AE – effort	10447.46	2080.26	2080.26
11.	$\overline{ACL}$ – average coupling level	6.3421	3.3660	3.3660

If the number of distinct architectural modules is increased from 6 to 7, by including an Execution Management System (EMS) in the software architecture, the number of distinct interfaces required changes accordingly (see the last row of Table 3, containing the underlined EMS module). The corresponding metric values are showed in Table 4. It is worth pointing out that by adding a new component the average coupling level of P2P based architecture increased with about 12%, while for ORB and MOM based architectures the value of $\overline{ACL}$ increased only with 3.6%. In addition, if increasing the overall number of modules made the effort required to implement the P2P based architecture 5 times greater than the effort to implement an ORB or MOM based one, then only adding a new component to the architecture raised the same effort difference to about 7 times. This empirical data test supports the higher level of operational responsiveness provided by loosely coupled software architecture.

Table 4. Increased only the number of distinct modules from 6 to 7

No	Software architecture characteristics	P2P based	ORB based	MOM based
1.	i – number of distinct APIs	10	3	3
2.	m – number if distinct modules	7	7	7
3.	I – overall number of APIs	10	3	3
4.	M – overall number of modules	7	7	7
5.	AL – vocabulary	17	10	10
6.	AS – size	17	10	10
7.	$\widehat{AS}$ – calculated size	52.87	24.41	24.41
8.	AV – volume	69.49	33.22	33.22
9.	AD – difficulty	5	1.5	1.5
10.	AE – effort	347.44	49.83	49.83
11.	$\overline{ACL}$ – average coupling level	6.0556	3.2059	3.2059

Increased both, the number of distinct modules from 6 to 7, and the overall number of modules, from 6 to 39, as follows:

- 20 Trading GUI
- 4 Order management systems
- 4 Execution management systems
- 4 Exchange lines
- 4 Trade management systems
- 2 Client confirmation systems
- 1 Settlement/Clearing system

The proposed metric values are presented in Table 5. Comparing with the reference configuration, the average coupling level of P2P based architecture increased with 38%, while for ORB and MOM based architectures the value of $\overline{ACL}$ increased with about 20%.

Table 5. Increased both the number of distinct modules and overall number of architectural modules

No	Software architecture characteristics	P2P based	ORB based	MOM based
1.	i – number of distinct APIs	10	3	3
2.	m – number if distinct modules	7	7	7
3.	I – overall number of APIs	98	44	44
4.	M – overall number of modules	39	39	39
5.	AL – vocabulary	17	10	10
6.	AS – size	137	83	83
7.	$\widehat{AS}$ – calculated size	52.87	24.41	24.41
8.	AV – volume	559.99	275.72	275.72
9.	AD – difficulty	27.86	8.36	8.36
10.	AE – effort	15599.63	2304.20	2304.20
11.	$\overline{ACL}$ – average coupling level	7.4727	3.7059	3.7059

5 Proposed Architectural Approaches for Increasing the Operational Responsiveness

Taking into account that having more flexibility in responding to operational needs does not entirely compensate for fast and direct data flow required by the ordering system, we have to observe that there is space for improving the overall performance of a trading architecture by combining the coupling model specific to each presented model.

Figure 5 illustrates a trading architecture that retains the P2P coupling model for the ordering flow, and combines it with a MOM based solution for the trade processing needs.

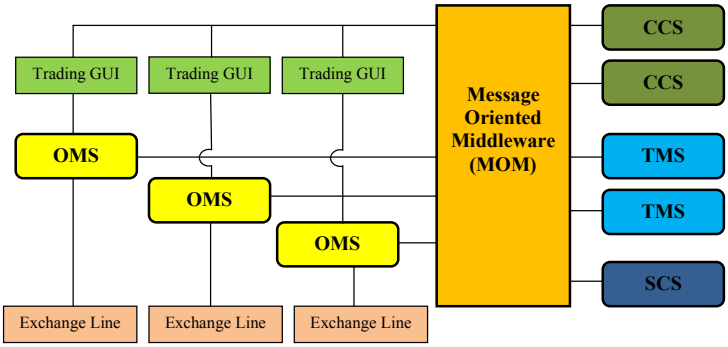

Fig. 5. Mixed architecture, P2P for order flow, MOM based for trade processing

The messaging approach for middle office and back office systems provides the reliability, low maintenance cost, and flexibility to adapt to third party software architectures (client, trust banks, clearing institutions), while having the main focus on the order processing area of the architecture. The messaging services do not necessarily have to be provides by a single message broker, but by a cluster of distributed message providers. There is indeed a logically centralized, physically distributed approach. The application developer does not need to care where the message broker is located and if there is only one message broker that handles messages from a given producer (source) to a given consumer (destination).

On the other hand, for ordering can be identified and isolated specific flows based on asset classes, type of clients, location of the execution points etc. Ordering flow segregation, achieved by creating distinct processing channels for different clients, assets, and exchanges it is natural, desired from business perspective, and increases the operational responsiveness through the usage of multiple instances of the same architectural module [20]. Each ordering flow can be set up in a *hot-stand-by* configuration, with one primary and one secondary servers, *failing over* and *falling back* mechanisms and procedures.

On the trade processing side, there can be easily plugged-in additional instances of a trade managing server, a client confirmation server when needed, along with ease for adding new modules all together, like an execution managing systems (EMS) or a database system (DBS), the latter responsible with the distribution of persisted data. Building clusters of message providers for specific global flows opens the route to cloud computing (Figure 6) [21].

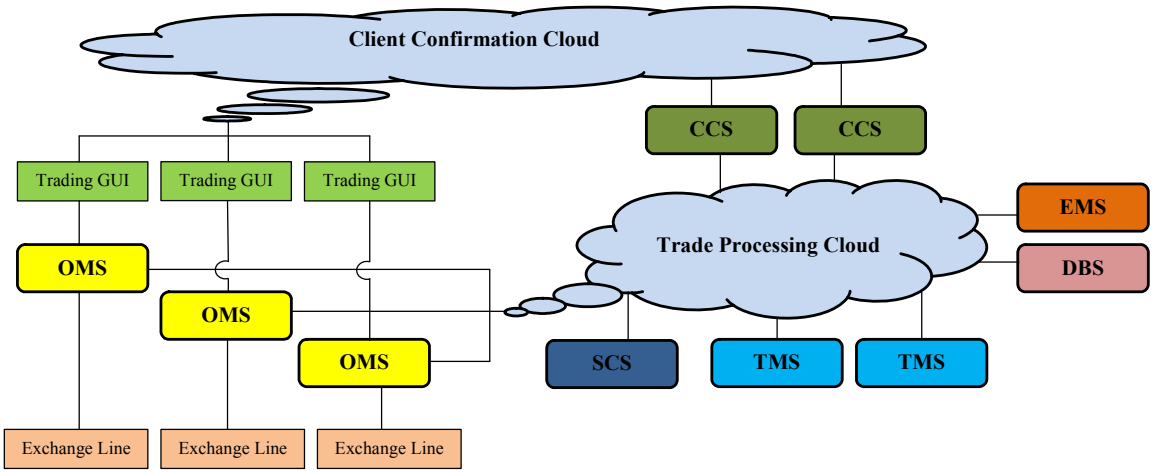

Fig. 6. Middle and back offices supported by services within cloud infrastructures

Table 6 specifies the number of distinct interfaces required by each module within the proposed mixed software architectures: P2P combined with MOM, and P2P with trade processing services in cloud computing configuration.

Table 6. Number of distinct interfaces per module for the mixed architectures

No	Distinct module of software architecture	P2P and MOM	P2P and Cloud
1.	Trading GUI	2	2
2.	Exchange line	2	2
3.	OMS – Order Management System	3	3
4.	TMS – Trade Management System	1	1
5.	CCS – Client Confirmation System	1	2
6.	SCS – Settlement/Clearing System	2	2

Computing the proposed coupling metrics for the mixed software architectures we obtained the following results, showed in Table 7.

Table 7. Coupling metric values for the mixed architectures

No	Software architecture characteristics	P2P and MOM	P2P and Cloud
1.	$\boldsymbol{i}$ – number of distinct APIs	5	6
2.	$\boldsymbol{m}$ – number if distinct modules	6	6
3.	$\boldsymbol{I}$ – overall number of APIs	5	6
4.	$\boldsymbol{M}$ – overall number of modules	6	6
5.	$\boldsymbol{AL}$ – vocabulary	11	12
6.	$\boldsymbol{AS}$ – size	11	12
7.	$\widetilde{\boldsymbol{AS}}$ – calculated size	27.12	31.02
8.	$\boldsymbol{AV}$ – volume	38.05	43.02
9.	$\boldsymbol{AD}$ – difficulty	2.5	3
10.	$\boldsymbol{AE}$ – effort	95.13	129.06
11.	$\overline{\boldsymbol{ACL}}$ – average coupling level	4.2553	4.6422

As expected, the average level of coupling for these two mixed software architecture have increased but, even in the case of multiple clouds to connect to, the effort for implement the architecture is about half of the effort required by a pure P2P approach.

7 Conclusions and Further Research

The empirical observation that motivates our research relies on the difficulty to assess the performance of a trading architecture beyond a few synthetic indicators like response time, system latency, availability or volume capacity.

Trading systems are complex architectures of distributed resources. However, in the context of a large brokerage firm, which offers a global coverage from both, market and client perspectives, the term "distributed" gains a critical significance indeed. Offering a low latency ordering system by nowadays standards is relatively easily achievable, but integrating it in a flexible manner within the broader information system architecture of a broker/dealer requires operational aspects to be factored in.

In this paper, we identified the chief trading architecture models that have been used in the industry, and analyzed comparatively their strengths and weaknesses.

In order to support our trading architecture analysis, we proposed a metric for measuring the coupling level within software architecture, and employed it to identify architectural designs that can offer a higher level of operational responsiveness.

Our ongoing research aims to explore the properties of the coupling metrics model that we briefly introduced herein, and further refine its ability to characterize distributed software architectures.

References

[1] L. Harris, *Trading and Exchanges*, Oxford University Press, Oxford, 2003

[2] A. R. Schwartz, R. Francioni, *Equity market in Action (The Fundamentals of Liquidity, Market Structure & Trading)*, John Wiley & Sons, Inc., 2004

[3] L. Fulton, "Operational Responsiveness", *Progress Software Corporation*, white paper, 2009, http://www.progress.com/en/ opera-

tional-responsiveness.html
[4] McIntyre Hal (editor), "How the U.S. Securities Industry Works - Updated and Expanded in 2004", The Summit Group Press, New York, 2004
[5] NASDAQ OMX Global Data Products, http://www.nasdaqtrader.com/
[6] Reuters Global Market Data, http://www.reuters.com/finance/global-market-data
[7] McIntyre Hal (editor), "Straight Through Processing", The Summit Group Publishing, Inc., New York, 2004
[8] C. Vinţe, "The Informatics of the Equity Markets - A Collaborative Approach", *Informatica Economica Journal*, vol. 13, no. 2, 2009.
[9] A. S. Tanenbaum, M. van Steen, *Distributed Systems - Principles and Paradigm*, Vrije Universiteit Amsterdam, The Netherlands, Prentice Hall, New Jersey, 2002
[10] W.R. Stevens, *UNIX Network Programming, Vol. 1, Networking APIs: Sockets and XTI*, Second Edition, Prentice Hall, 1998
[11] A. S. Tanenbaum, *Computer Networks, Fourth Edition*, Vrije Universiteit Amsterdam, The Netherlands, Pearson Education Inc., Prentice Hall PTR, New Jersey, 2003
[12] C. Vinţe, "Aspecte ale Proiectării unui Order Request Broker (ORB) - Partea I", *Informatica Economică Journal*, Vol. V, No. 2 (18)/2001, INFOREC, Bucharest, 2001
[13] C. Vinţe, "Aspecte ale Proiectării unui Order Request Broker (ORB) - Partea a II-a", *Informatica Economică Journal*, Vol. V, No. 3 (19)/2001, INFOREC, Bucharest, 2001
[14] Erl Thomas (with additional contributors), "SOA Design Patterns", Prentice Hall by SOA Systems Inc., New Jersey, NY, 2009
[15] C. Vinţe, "Upon a Trading System Architecture based on OpenMQ Middleware", *Open Source Science Journal*, Vol. 1, no. 1, 2009, http://www.opensourcejournal.ro/
[16] C. Vinţe, "Upon a Message-Oriented Trading API", *Informatica Economica Journal*, vol. 14, no. 1, 2010, http://www.revistaie.ase.ro/
[17] Richards Mark, Monson-Haefel Richard, Chappell A. David , "Java Message Service (Second Edition)", O'Reilly Media Inc., Sebastopol, CA, 2009
[18] S. Henry, D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering* Volume SE-7, Issue 5, Sept. 1981 Page(s): 510 – 518
[19] Halstead, Maurice H., "Elements of Software Science", Amsterdam: Elsevier North-Holland, Inc., ISBN 0-444-00205-7, 1977
[20] Suzanne Dence, Daniel Latimore, John White, "The trader is dead, long live the trader! - A financial markets renaissance", IBM Institute for Business Value, 2006, http://www-935.ibm.com/services /us/imc/pdf/ge510-6270-trader.pdf
[21] M. Risca, D. Malik, A. Kessler, "Trading Floor Architecture", Cisco Systems, 2007, http://www.cisco.com/ web/strategy/ docs/ finance/TradingFlrArch110707.pdf

Claudiu VINŢE has over fifteen years of experience in the design and implementation of software for equity trading systems and automatic trade processing. In 2007 Claudiu co-founded Opteamsys Solutions, a software provider in the field of securities trading technology and equity markets analysis tools. Previously, he was for over six years with Goldman Sachs in Tokyo, Japan, as Senior Analyst within the Trading Technology Department. Since 2009, Claudiu has joined the Department of Economic Informatics and Cybernetics as adjunct assistant professor. He has been coordinating the course upon *The Informatics of the Equity Markets*, and the seminars on *Software Quality Management* within the Master's program in Economic Informatics. Claudiu graduated in 1994 The Faculty of Cybernetics, Statistics and Economic Informatics, Department of Economic Informatics, within The Bucharest University of Economic Studies. He holds a PhD in Economic Cybernetics and Statistics from The Bucharest University of Economic Studies. His domains of interest and research include combinatorial algorithms, reusable software design, middleware components, algorithmic trading and web technologies for equity markets analysis.