A Reference Infrastructure for Electronic Commerce

Yu LI, Weihong LI, Hee Beng Kuan TAN
ICIS, School of EEE
Nanyang Technological University
Singapore 639798
{P147204023, P148854264, IBKTAN}@ntu.edu.sg

Abstract
Electronic Commerce is all the rage these days. Current challenge of Electronic Commerce for suppliers is to allow building dynamic supply link and to meet various and wide needs for the customers, furthermore to build more reasonable electronic market places at run time. In this article we aim to provide a common reference infrastructure contains an instantiation of the ISO 7 layer model for Electronic Commerce and uses modern software engineering design paradigms. A way of identifying distributed building primitives for large-scaled integration and multi-enterprise transactions is shown.

1. Introduction

Electronic Commerce (E-commerce or EC) is all the rage these days [1]. The first-generation EC systems have already created marketplace, reduced transaction costs, and benefited both consumers and suppliers in a variety of business models. Current E-commerce research focuses on realization and standardization of infrastructure issues such as secure online payment, electronic product catalogues, user profiles and digital delivery [7][8] and also on migrating business workflow to web-based E-commerce [9][10][11]. However, in parallel there are challenges associated with the spread of electronic commerce:

Firstly for global Successful E-Commerce in Society, the ease for building on-line web store increases the web overload. This emerging trend also fragments the internet-enabled marketplace. The Internet's potential is imperiled by the risking specter of this digital anarchy.

Secondly for supplier system, the variability of each value-added step in EC need further study to make EC successfully transfer from traditional commerce to this on-line open business via the Internet. And electronic commerce offers the platform of fair competition and collaboration for various suppliers.

Finally from Customer Viewpoint, different customers have different preference. Customers expect EC system can provide them whatever combination they need, good products and services with reasonable price under various circumstances.

1.1. Literature Review

Many frameworks for Electronic Commerce is build, but most are not aimed to reuse architecture.

XPECT [7] is a general framework for E-commerce. The prototype application is a simple pay-per-view document server that involves the third party – a basic banker agent. This system does not consider the future changing in the framework itself, neither about the integration and customization for different systems.

Eco system [12] from CommerceNet is concerned about inter-operating several proprietary systems. They provide a solution which effort to reuse the modules across industry product line to build a framework. The weakness for Eco system is that it does not consider the reuse problem both from architecture and component point of view because sometimes E-commerce modules in product line are different enough to inter-operate, as well as it does not consider about
evolution in framework itself.

DASHER [13] project focuses on a service infrastructure that supports distributed applications and federations of loosely affiliated software components. To achieve its objective, dasher’s research includes work on middleware and a framework for modular services, as well as readily usable and configurable services that can be put to use in multiple information repository application. Actually this framework consider fully about both reuse and evolution of the system. But we expect e-commerce application systems with more open architecture and more dynamic relationship.

1.2. Our Solution for Electronic Commerce

Therefore, it is urgent to allow mixing and matching of products and services that bundle formerly disparate components into a distinct one to serve a single-customer’s basis [14]. Furthermore, the design and architecture patterns for E-commerce need to be always integrity although some new capabilities and some internal features are customized. And the future of E-commerce is to integrate and customize [15]. Given this trend, flexible and open architecture, which can provide dynamic relationships among previously unknown parties and does not require both suppliers and customers to understand all the complexities inherent in the global E-commerce context, is needed. As well as, the design and architecture patterns for EC developers to quickly build specific E-commerce application systems by reusable components and customized services are also needed.

Our approach takes a different approach to integrating and customizing systems by using design patterns based on reference architecture. So when various systems are integrated, there is no need to conform a single standard. When a company adds new capabilities to the e-commerce platform, over time, the boundaries blur and eventually melt away completely, and we are back to a single monolithic more open and more dynamic e-commerce application platform.

In this article, we will describe our reference architecture rich with design patterns at multiple stages of creating E-commerce system. We start by analyzing the existed requirements complexities and using architecture patterns to guide the creation of abstractions necessary to accommodate future changes yet maintain architecture integrity. Then we design our reference model that identifies seven main interfaces at which building blocks (services), technologies and standards really matter. Furthermore, we use patterns to help us achieving reuse in the implementation stages by favoring object composition or delegation over class inheritance. Finally, we also discuss the patterns help document strategies properties of the electronic commerce at a higher level than the source code.

2. Reference Infrastructure for Electronic Commerce

We design a Reference Infrastructure for electronic commerce intended to enable rapid and effective integration and customization of EC application. We focus on those that simplify the integration and development of EC application, by hiding complexities associated with EC technology and exposing the necessary variability in the whole problem domain. Also we focus on abstractions necessary to identify reusable components and on the variances necessary to provide the customizations required both by specific EC application end uses and by application developers.

2.1. EC Concept and Complexities

At a simplistic level, the difference between two EC applications boil down to what service the application provides and how the application provides. Both [12] and [13] give architectures based on services. Fortunately, we can abstract service requirement in the context - EC problem domain.

However, the result that the EC application returns in response to a end user’s requirement is variable depending on the state of dynamic combination of building blocks of the whole infrastructure in the context of a E-commerce application.

Several factors contribute to complexity in integrating and developing EC applications. Firstly, services in EC domain are always changing over time. The internal strategy may be evolved so that the traditional economy can be transferred to real E-commerce smoothly. Even, new service, new features, or new capabilities will be added when integrating various EC applications and developing new
application systems. Secondly, EC technology is inherently distributed and asynchronous and requires adhering to well-defined handshaking protocols or standards, but because of the complexity of the interoperation, no single standards will suffice. Thirdly, the distributed and client-oriented [16] nature of EC make it possible for the end users to initiate and action during a transaction process so that the application must either disallow or defer any action until all previous transaction processing can be completed. Finally, EC-programming interfaces typically require a service of calls to accomplish a single transaction task. The uncertainty associated with high-accuracy transaction forces application to constantly monitor and deal with errors or abnormality while active services communicate with applications. Therefore the application must therefore build its own infrastructure to direct and dispatch messages or tasks at the process level.

2.2. Framework Abstraction - Overview of Reference Architecture

The advantage of reference architecture for software engineering lies in the reusability of entire architecture at design level [17]. Since it is possible to separate requirement [18] in EC problem domain, so an abstract description of various reference requirements can be fulfilled in an application-specific building block of the reference architecture. Then the integration of specific E-commerce application and the design of a new application just involve combining those building blocks that finish the given requirements.

Three-tier [19] architecture, also called three-schema architecture becomes increasingly common in electronic commerce system. It decomposes electronic commerce system to three layers.

User Interface. This top level contains the desktop applications, implementing the presentation of information and interaction with the user. These clients are implemented platform independently. They use "business object" to receive input of transaction from and return the result of transaction to users.

Business logic. This middle level implement most parts of the application functionality: the business objects and application objects. These objects implement services. Sometimes, this layer can be extended or sub-divided into further levels, such as third party or intermediate party [20] to clear the business process for increasingly complex processing logic.

Data Storage. This level contains data storage components. Possible systems in this level are distributed database systems, information archives or data warehouse services.

![Reference Architecture for Electronic Commerce](image)

Figure 1. Reference Architecture for Electronic Commerce
But three-tier architecture is inadequate and can not effectively integrate various e-commerce systems because building the third layer adaptive to every system is costly and need to know much more details on knowledge of each business domain model and make the learning curve quite steep for integration complex system. Even those architectures which have third party travel agent or trusted payment party also can’t do work very well. Although 3-tier architecture is not suitable for integration, it can also be our basis for reference architecture because it is possible to isolate individual functions, which in turn enhances control of the system, both for construction phase and for maintenance phase.

Our reference architecture has referred to architecture pattern – application facade [19] and some successful standard model, such as Posix Open System Reference Model [21]. An application facade schema [19] is also a layered architecture pattern in which application is split into presentation and application logic. Each presentation thus has a simple interface to the domain model that minimizes any processing. Application facades are best understood from a fairly complex and abstract cosmos domain model, which can handle a wide range of cases. Posix is a reference model that does not have any layers and identifies the main interfaccs at which standards really matter.

We factors out the business logic into separate layers to build our reference architecture that identifies main interfaces at which technologies and standards really matter. Figure 1 is our reference architecture with maximum flexibility and reuse. It creates seven layers for Electronic Commerce like ISO 7 layers standard. Following are the layers:

- User Interface
- Domain-Specific Layer
- Function Session Layer
- Cooperation & Coordination Layer
- Communication Layer
- Basic Business Layer
- Database & Directory Service Layer

2.3. Details of Each Layer

User Interface. User Interface is provides a user with the interface to the brokerage services (Input Services) and alerts users as to the availability of information (Display Services). Although it is HCI (Human Computer Interface) research area that has sample standards of look and feel such as Motif, Window 98 and Macintosh, we consider how the business objects (such as online catalog and user identification) and application objects (such as query and order) entered to the transaction process. Sample standards are HTTP, FTP, SMTP.

Domain-Specific Layer. This layer gives presentation and manipulation of structured data for the benefits of application program (Presentation Services). It isolates the basic notion of a form for the user interface and underlying layers. The main service is delivering information in file formats as real-time continuous data streams to and from both customers and systems. Sample standards are HTML, XML, CGI, ED1 and SGML.

Function Session Layer. This layer organizes, synchronizes, and manages the exchange of data. The principle for creating this separate layer is to handle function that are manifestly different in the process performed or the technology involved and to collect similar function into the same layer. This lay includes many functional services: Search Services, Locate Services, Order Service, Payment Services, Tariff Services and Configure Services. Sample standards are standard application program languages, such as C++, Java.

Cooperation & Coordination Layer. This layer focuses on scheduling access to resources and it relieves higher-level entities from any operation concerns between them. This layer includes many services: Negotiation Services which realize a set of function from different situations, Schedule Services that operate plan process, enact workflow, define priorities in case of conflicts, solve conflict and conflict. And it also includes Allocation Services that allocate, de-allocate and optimize functions from the lower lay such as recovery from breakdowns or technological problems and information integration which introduces and integrates new information in members' views after some function. And the most importance for this layer is that some travel agents and intermediate agents lie in this layer. In summary, this layer has the services: Schedule Services, Allocate Services and Negotiation Services. Sample standards are extension of standard application program languages, such as DCOM/ActiveX, CORBA/IOP, JavaBean and Java RMI.

Communication Layer. EC system must interact with standard communication equipment by Communication Services. This layer also provides meta-data collection.
Sample specification at this level is TCP/IP, UDP that provide the communication service and data conversion to overcome the heterogeneity of transmission, as well as Resource Reservation Service which has the relevant Protocol is RSVP that provides resource setting up service. Resource Reservation or Collection Services is very important. For example, some open systems will act as the final destination of data and other open systems may act only as intermediate nodes. Suppliers oriented content, such as which locations and suppliers have the needed products and services, will be grouped in this layer from the successful experience. Thus, this communication layer will provide a connection path between a pair of supplier entities, including the case where intermediate locations are involved.

**Basic Business Layer.** This layer provides the service of the performance, reliability, and security to meet mission-critical business needs. It can be viewed as a virtual machine that provides services to all the other frameworks in a sufficiently abstracted form so that the services can be implemented in alternative ways. The basic services include System Management Services, Transaction Services and Security Services. System management services provide mechanism of starting, stopping or configuring and fault and event management such as quality-of-service management and delivery receipts. Transaction services provide monitors and mechanisms for non-standard transactions. Security services that include authentication and authorization mechanisms for users and for components, such as authenticated packets and smart firewall, to ensure only those authorized business partners can pass packets so that it can provide the security policy of its particular security domain. This layer need monitor standards such as SNMP and CORBA-based monitors for distributed systems such as OrbixManager from Iona Technologies.

**Database & Directory Service Layer.** This level contains data storage components for both business objects and other objects in application and it include Database Services and Directory Services that are the repositories for business information and process. Sample standards are DB2, ORACLE, SQL.

Our architecture is an abstract combination with maximum flexibility in standards. Each layer encapsulates the functional services in abstract terms from the layer above it and the layer under it. From EC vendors' viewpoint, they would rather agree on a meta-standard than a single standard. That's because a single standard would require most to abandon rival technologies in which they have a substantial investment. “Just tell me what protocol you prefer, and I'll accommodate it if I can” is a favorite philosophy for EC vendors. Since today's computer can support multiple standards, negotiation is a practical way of realizing integration. Fortunately much more work has done to facilitate the negotiation among standards or technologies of the same functionally, for example between XML and SGML [22], between DCOM/ActiveX and Java/JavaBean [23].

Although services are arranged hierarchically as in Figure 1, the actual implementation is flat so that any object can request a service from any other. This is convenient because situations do frequently arise where object lower in the hierarchy requires services from the above one. For example, Network services may require security services; cooperation and coordination services always require communication services.

### 3. Business Primitive and Interaction Primitive

The services between adjacent layers in the reference architecture are expressed in terms of business primitives and layer interaction primitive. A primitive specifies the function to be performed. The actual form of a primitive is implementation-dependent. In the following of this part, we will first introduce the interaction primitive and business primitive.

#### 3.1. Layer Interaction Primitive

Layer interaction primitive refers to the control of thread of execution between two layers. Four types of interaction primitive are defined in Table 1.

Figure 2 suggests the execution of interaction primitive. This sequence of events is referred to as a confirmed service, as the initiator receives confirmation that requested service has had the desired effect at the other end. If only request and indication primitives are involved, then the service dialogue is a non-confirmed service, the initiator receives no confirmation that the requested action has taken place.
### Table 1. Interaction Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST</td>
<td>A primitive issued by a service user to invoke business primitive and to pass business primitive to fully specify the requested service.</td>
</tr>
<tr>
<td>INDICATION</td>
<td>A primitive issued by a service provider to either: 1. indicate that a procedure has been invoked by the peer service user on the connection and to provide the associated parameters 2. notify the service user of a provider-initiated action</td>
</tr>
<tr>
<td>RESPONSE</td>
<td>A primitive issued by a service user to acknowledge or complete some procedure previously invoked by an indication to that user.</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>A primitive issued by a service provider to acknowledge or complete some procedure previously invoked by a request by the service user.</td>
</tr>
</tbody>
</table>

![Figure 2. Execution of Confirmed Interaction Primitive](image)

#### 3.2. Business Primitive

Business primitive defines the abstraction of requirement for electronic commerce system. Since the requirements for this single problem domain - electronic commerce system are the same, so the variance among different domain is to be expected, and the cost of integration and customization associating with change can be reducing by preparing it. At the same time, these business primitives define those aspects of an application that must be kept flexible for different adaptations of the framework. In this problem domain electronic commerce, business primitives are often the concerns that both buyers and suppliers care about. So the successful architecture must clearly define those primitives to ensure integration and construction of specific e-commerce application at flexible scale. Following are those primitive in electronic commerce systems.

Following illustrates the core business primitive:

**Item and Stream Delivery Management**

Item and stream delivery management primitive gets requirement from user interface and delivers file-structured items and real-time multi-media data streams to and from the customer. The important business object is:

**Customer profile**

Customer profile is the gateway for customers to enter the electronic commerce system and it is also the beginning of the transaction. Different customers have different defined user profiles. In EC system, customer accessing commerce site, get useful information, delegating his/her task to agents, ordering products are all based on user profiles. When we integrate various electronic commerce systems, it is the first primitive to be considered.

**Online catalog**

The product data shown on virtual catalog are retrieved from different suppliers, so we must consider two problems: the heterogeneity of catalog structures [24] and the heterogeneity of data semantics. For example, the money currency is US dollar in some e-commerce system, but UK pound in others. We must explicit describe the heterogeneity among various system. The rules to translate the product description to a global one will be set up gradually as more items are encountered. After some system being integrated, the system tend to be complete. Another advantage is that it enables easy product-comparison from different suppliers.

**Contracts**

The contract between suppliers and buyers is the key of transaction. Commonly, contracts include who buy, what to buy, from where to buy, how to pay, how to deliver, etc. Different e-commerce system has much more difference
on contracts. How to uniform the heterogeneous contracts is the basis of future transaction in the integrated e-commerce system to effective exploit the future union of various subsystems.

**Search**

Search primitive accepts requests to carry out a search for products that fit a particular user description. It returns lists of identifiers or products or even locations that fit the description. Through searching, supplier’s provision and customer’s requirements are matched. In electronic commerce system, mobile agent is responsible to query on remote host. There are so many mobile agents in the e-commerce system, how to allocate and de-allocate these agents to complete the transaction is also the important business operation in e-commerce system.

**Auction**

Dynamic price is the content of online auction that challenges the traditional economic market. Auctions involve not only consumers, but also business. An auction can perform dynamically and enable the exchange of goods much as stock exchanges manage the buying and selling of securities [25].

**Metadata Collection**

Metadata collection primitive collects information about product description and where they are available. Metadata will provide convenient way to the future product searching.

**Order**

Order primitive manages negotiation between a customer and a supplier, in order that agreement may be reached on the terms of availability of a particular product or a group of products. Following the negotiation phase, the order primitive accepts purchase commitment from the customer and forwards them to the supplier. It returns a notification of the status of the order action.

**Payment and Authentication**

E-commerce system must consider payment and security service to authorize and finalize the true transaction process. Flexible payment option and secure payment will signal the termination of transaction stage. Authentication primitive must provide mechanism that allows a user or components to prove identity to the brokerage system or among different brokerage system. It also needs to provide the mechanism that monitors the system and charges of using the system, such as tariff or tax for the society.

**Delivery and Feedback**

Flexible, fast delivery and the feedback from customers are the post-service for e-commerce. They are also important for both business corporations and customers. Complete e-commerce application must provide the relevant electronic delivery and feedback in its application.

Above are the business primitives in electronic commerce system. Using the reference architecture to identify the relevant building block can make development, integration and customization flexible. Since the development of business primitive in our architecture is application-independent, following we discuss abstraction of business primitive.

### 3.3. Abstraction of Business Primitive

In our architecture, we use software design patterns to abstract business primitive and to facilitate reuse of design and code and decouple or integrate the major component of the system. A software design pattern documents a recurring problem in a given context and a proven solution to that problem [26]. Design patterns, properly applied, can make a system more robust to change by allowing some aspect of the system to vary independent of other aspects [27]. Design patterns are abstract enough to help application software ride over the bumps of requirement change but concrete enough to be understandable. Abstract business primitive as a set of design patterns is an effective means of achieving a high level of communication between different domain and also evolution for electronic commerce system. In our paper, we use design patterns to hide the complexities of business primitive for creating electronic commerce applications. Table 2 summarizes some of the design patterns we used and the reason we used them.

We use Facade [26] pattern to provide a unified, abstract interface for a set of interface provided by each e-commerce system. Each facade selects information from the underlying domain and simplifies it into the exact form that the uniform family system need. Therefore, the facade provides operation for the whole problem domain, and a reusable community, essentially a new kind of collection, is usually the best solution.
### Table 2. Design Patterns Used in Building the Electronic Commerce

<table>
<thead>
<tr>
<th>Design Patterns Used</th>
<th>Reason for Using the Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object-Oriented Patterns</strong></td>
<td></td>
</tr>
<tr>
<td>Facade</td>
<td>Provide a set of abstract interfaces supported by the various electronic commerce systems. Abstract business primitive concepts to facilitate the use of a different technology.</td>
</tr>
<tr>
<td>Adapter (Wrapper)</td>
<td>Enable the use of previous e-commerce components whose interface did not match the standard interface. Create a reusable class that cooperates with unrelated interface that don't necessarily have compatible interface.</td>
</tr>
<tr>
<td>Interpreter</td>
<td>Because of much more heterogeneity in integrated user profile, catalog and contracts, interpreter is necessary to define a representation for global grammar along with an interpreter that uses the representation to interpret sentence in the language.</td>
</tr>
<tr>
<td>Observer</td>
<td>Define a dependency between one business primitive and other business primitives and allocate other operation in the system when changes occur so that customer requirement can be notified to the other operation.</td>
</tr>
<tr>
<td>Singleton</td>
<td>Create exactly one instance of some primitive for per transaction since it begins to operate at the process level.</td>
</tr>
<tr>
<td>Strategy</td>
<td>Define a family of auction algorithms, encapsulate each one, and make them interchangeable. Strategy let the algorithms vary independently from clients that use it.</td>
</tr>
<tr>
<td><strong>Agent Task Patterns</strong></td>
<td></td>
</tr>
<tr>
<td>Master-Slave</td>
<td>Define a scheme whereby a master agent can delegate a task to a slave agent.</td>
</tr>
<tr>
<td>Plan</td>
<td>Provide a way of defining the coordination of multiple tasks to be performed on multiple hosts.</td>
</tr>
<tr>
<td><strong>Distributed Patterns</strong></td>
<td></td>
</tr>
<tr>
<td>Asynchronous Completion Token</td>
<td>Allow application to efficiently associate state with the completion of asynchronous operations.</td>
</tr>
<tr>
<td>Service Configurator</td>
<td>Decouple the implementation of services from the time when they are executed without having to know who its observers are. Any number of observers can subscribe to receive notifications.</td>
</tr>
</tbody>
</table>

Singleton [26] pattern is important in integrating e-commerce system. It guarantees a sole instance of the e-commerce subsystem for each transaction process. Sometime, single instance is necessary to communicate with the e-commerce subsystem by sending requests to the other object, which forwards the requests to the appropriate subsystem objects.

We use Adapter (or Wrapper) [26] pattern to enable the use of existing classes whose interfaces did not match the whole application system. For example, we used Adapter to create a speech-enabled GUI window that multiple inherits from both an abstract speech client class and specific GUI window class in one application. Thus, we were able to encapsulate speech-aware behavior in abstract framework classes so that GUI did not have to be modified to exhibit speech-aware behavior.

Observer [26] pattern describe how establish the dependency between one business primitive and other business primitives and allocate other operation in the system when changes occur so that customer requirement can be notified to the other operation. This kind of interaction is also known as publish-subscribe. The subject is the publisher of notifications. It sends out these notifications without having to know who its observers are. Any number of observers can subscribe to receive notifications.

Interpreter [26] defines a representation for much more heterogeneity in integrated user profile, catalog and contracts, interpreter is necessary to define a representation for global grammar along with an interpreter that uses the representation to interpret sentence in the language.

Strategy [26] defines a family of auction algorithms, encapsulate each one, and make them interchangeable. Strategy let the algorithms vary independently from clients that use it.

Asynchronous Completion Token (ACT) [28] pattern allows applications to efficiently associate state with the completion of asynchronous operations. Applications must respond to many types of events, ranging from user interface, other operation to storage database messages. For example, delay-sensitive events often perform long-running operations asynchronously to avoid blocking or abnormality.

Service Configurator [29] pattern decouples the implementation and configuration of services, thereby increasing an application's...
flexibility and extensibility by allowing its constituent services to be configured at any point in time. We implemented this pattern again using the speech profile concept, which set up the configuration information for each application.

Especially we use Agent Task patterns. Using the principle embodied in the master-slave and plan patterns, we separated agent definition from agent execution for each application to formula our new system.

The agent master-slave patterns allow a master agent to delegate a task to a slave agent. The slave agent will move to a destination host, perform the assigned task, and return with the possible result of that task.

The more complex plan pattern adopts a workflow concept to organize multiple tasks to be performed in sequence or in parallel by multiple agents. The plan encapsulates the task flow, which is then hidden from the agent. The agent merely provides the mobility capabilities needed to perform tasks at specific destination. The plan promotes reusability of tasks, dynamic assignment of task to agents, and even composition of tasks.

In our infrastructure, we use design patterns to differentiate one system from another to build unified product model that includes all commonality and variation across various systems. These patterns can be incorporated into any architectural style and will let developers build product family architectures, from which they can not only integrate various sub e-commerce system but also derive a new system’s architecture.

3.4. Execution of Business Primitive

Figure 3 gives the execution of business primitive. The collaboration between the business primitives is as follow:

According to some trigger rules, the business primitive in layer N-1 creates many business primitives in layer N.

Business primitive in layer N initialize and dispatch task to business primitive to local or remote destination according to standard package.

The business primitive in layer N+1 does the job and returns with the result of the task to layer N and subsequently returns to layer N-1.

4. Conclusion

Ultimately, based on ISO multi-layered architecture, using a common set of abstractions across the analysis, design and implementation phases and the use of design patterns to capture these abstraction led to better communication and a more efficient for handling changes in the E-commerce frameworks evolved. To achieve the above objectives, the future work will focus on modulated middleware, as well as on readily usable and configurable module that can be put to use in multiple information repository applications. Transparently, users can access parts and modules with appropriate capabilities for particular tasks that they have in mind. The global system deployed various vendors of those goods to customize the need for customers. Thus, the global concerns of the
effort are to increase the speed with which customized, task-oriented information repositories can be formed. Not only strategies for respective design patterns but also the relationship between them are to be perfected. Mechanism that simplifies the search and negotiation process for users' task should adopt the advanced algorithms.

In summary, the prospective customer will be able to access the service in a wide range of products and services. Suppliers can establish a commercial partner relationship and get more customers from their partners without non-monopolistic. For e-commerce developer and implementers, it gives the guideline and orientation for them and makes it easy for them to design and implement components quickly in many diverse domains. And the most importantly, global E-commerce system will become increasingly structured and controlled.

Reference
[18] Martin Fowler, Analysis Patterns: Reusable Object Model, Addison-Wesley, 1997