

A Proposal for Enterprise Application Integration
in a Distributed Mass Customization Printing Environment

By
Ben March

An Alternate Plan Paper Submitted

In Partial Fulfillment

Of the Requirements for the

Masters of Science

In

Computer Science

Minnesota State University, Mankato

Mankato, Minnesota

December 2008

Date _____

This alternate plan paper has been examined and approved.

Examining Committee:

Dr. Christophe Veltsos, Chairperson

Dr. Cyrus Azarbod

Dr. Michael Wells

ABSTRACT

A Proposal for Enterprise Application Integration
in a Distributed Mass Customization Printing Environment.

Benjamin P. March. Masters of Science in Computer Science.

Minnesota State University, Mankato. Mankato, MN. 2008. 77 pp.

The custom printing industry in the United States has seen a large fluctuation in sales over the last four decades. Changing cultural norms have increased the sales of formal announcements for weddings, graduations and births, while the increasing trend for offices to go paperless along with the personal printer has greatly decreased the need for personalized stationary and business cards.

The changes in the custom printing market have had great impact on the look of printing corporations' footprints. Mergers, divisions and acquisitions have been the key for printing corporations to successfully react and remain profitable in the constantly changing printing market. This reactive approach to the market has caused a high degree of compartmentalization and distribution in custom printing corporations.

This research will investigate the technical infrastructure in the printing industry and how proper enterprise application integration can be introduced to keep custom printing plants loosely coupled allowing corporate flexibility while increasing data visibility and empowering leaders to make informed business decisions.

TABLE OF CONTENTS

Chapter

I.	Introduction	1
	1.1 Problem Definition.....	1
	1.1.1 Disconnected Systems	2
	1.1.2 Access to Data to Make Informed Decisions.....	2
	1.1.3 Tightly Coupled Systems.....	3
	1.1.4 Slow Reaction Time and Limited Flexibility.....	3
II.	Review of Literature.....	5
	2.1 Mass Customization and the Printing Industry	6
	2.2 What is Enterprise Application Integration?.....	7
	2.3 Why use Enterprise Application Integration?.....	8
	2.4 EAI Integration Styles and Patterns	8
	2.5 EAI Topologies.....	10
	2.5.1 Hub and Spoke Message Brokers	10
	2.5.2 Distributed Agents	12
	2.5.3 Enterprise Service Bus.....	14
	2.5.4 Point to Point.....	15
	2.6 Enterprise Resource Planning (ERP) Systems.....	16
	2.7 EAI vs ERP.....	17
	2.8 Implementation Considerations	18
	2.9 Legacy EAI Technologies.....	22

2.9.1 CORBA.....	22
2.9.2 DCOM.....	23
2.10 Critical Success Factors and Pitfalls.....	24
III. A Proposed EAI Architecture.....	26
3.1 Explanation of Business Requirements	26
3.1.1 Fast Turnaround Times	26
3.1.2 Market Adaptability.....	27
3.1.3 Flexible and Modular Systems.....	27
3.1.4 Data Access and Business Intelligence.....	28
3.1.5 Managing Resources and Costs	29
3.2 Technical Requirements.....	29
3.2.6 Summary of Technical Requirements.....	32
IV. Simulation of Proposed Architecture	33
4.1 Overview.....	33
4.2 Simulation Process.....	33
4.2.2 Test Details	36
4.2.3 Environment.....	37
4.2.4 File Formats	37
4.2.5 Console Applications	39
4.2.6 Result Format.....	41
4.3 Hypothesis.....	41
4.4 Simulation Results	42
4.5 Results Summary	42

4.5.1 Number of Orders Produced	42
4.5.2 Longest Production Duration in Milliseconds	44
4.5.3 Average Production Duration in Milliseconds	47
V. Conclusions	50
5.1 Summary	50
5.2 Research Limitations	51
5.2.1 Further Topology Comparisons	51
5.2.2 Initial State Considerations	51
5.2.3 Scalability	51
5.3 Suggestions for Future Work	52
References	53
Appendix A: Glossary of Acronyms	59
Appendix B: Use Case Models	60
Appendix C: Simulation Configurations	62
Appendix D: Code Listing	65

CHAPTER 1

INTRODUCTION

Distributed manufacturing systems are becoming a common occurrence in today's economy. Corporate mergers, divisions, and acquisitions naturally add to the degree of distribution seen in manufacturing environments. As the degree of distribution rises in a corporation, so does the complexity of managing a business based on data in disparate systems. Enterprise Application Integration or EAI is a methodology of connecting disparate business systems. Through EAI, business systems can be connected to share data and business logic embedded in separate systems. By sharing data and business processes in disparate systems, a business can become more responsive to changes in the market and make intelligent business decisions much faster than without EAI. This paper focuses on proposing technical requirements for an EAI system specifically in a distributed printing environment.

1.1 Problem Definition

There are many challenges to confront in a distributed manufacturing environment. A distributed custom printing corporation shares those challenges and more. The custom printing industry is one example of a unique type of manufacturing called mass customization. In mass customization, a business attempts to produce customized goods or services at near mass production efficiency and cost.

There are many obstacles that prevent efficient mass customization. The allocation of resources may be inefficient. Implementation of business rules across the

enterprise may be difficult or virtually impossible. An enterprise's large size can actually hinder it in the market if it has many poorly integrated distributed systems. The remainder of this section details some of the imposing challenges a distributed printing corporation faces.

1.1.1 Disconnected Systems

To increase efficiency in distributed manufacturing systems, orders may be decomposed and re-distributed to an appropriate specialized producing plant [43]. If a plant does not have an electronic interface to receive their part of the decomposed order then the request must be physically delivered and manually entered before it can be produced. The physical delivery and manual entering of order information may not be a viable option in the printing industry. To be competitive in the market an order must typically be produced and shipped in less than twenty-four hours. Due to this time restriction and the inherent liabilities of manual entry, a printing plant that is electronically disconnected from the collective of connected printing plants cannot participate in decomposed order fulfillment.

1.1.2 Access to Data to Make Informed Decisions

The factors that can be taken into account when making business decisions are virtually limitless. Changes in the economy, inventory in a warehouse and even an employee's hours worked for the week can greatly affect how a business is able to meet their customer's needs. Knowing which factors are the most important, and how to manipulate them is key in making business decisions. A business leader may define new

business rules and redefine existing rules frequently to try to gain an advantage over the competition. An enterprise that is able to successfully implement important business decisions the fastest will usually out-perform their competitors in the marketplace and win customers.

1.1.3 Tightly Coupled Systems

Business rules often change faster than they can be implemented. Implementing business rules across an entire corporation can be a daunting task when many different systems require updating. The concept of virtual factories is becoming more commonplace in distributed manufacturing environments. A virtual factory or virtual enterprise is a system of applications or processes that operate a physical factory or group of physical factories [1]. While virtual factories are common in manufacturing environments, the method of integrating the systems must be performed diligently to make it cost effective.

Sometimes distributed systems are interconnected, but they are coupled tightly which leaves the systems rigid. In a tightly integrated environment, modifying a program in one system requires other connected systems to also be modified [24]. It is common for fast growing corporations to have tightly coupled systems. Mergers and acquisitions are often performed under a tight deadline. The tight deadlines may cause systems to be integrated hastily. Systems which are integrated hastily are occasionally not built with flexibility in mind.

1.1.4 Slow Reaction Time and Limited Flexibility

Tightly coupled systems lead to slow reaction time to the market. Even if business decisions are made quickly, the business rules that are created to enforce them must also be implemented quickly. Successful corporations must be able to swiftly recognize and react to any changes that may impact their business.

Small businesses are typically quick at adapting to business decision changes because their systems and processes are relatively small and contained. It is easy to understand a problem, how it is embedded in existing systems, and how the systems need to be modified in order to address a problem.

A company must also be able to ramp up or down its level of production to match the demand for a particular product or service. Eventually a successful company will mature and grow requiring more staff and equipment to fill its customer's needs. A successful organization may eventually turn into a corporation containing many companies acquired through growth, division, acquisitions and mergers. Corporations soon find themselves far from the well known environment in which their small business started. Even more alarming is that they are no longer able to quickly adapt to changes in the market. Small understandable systems and processes are now giant legacy systems of confusion. Corporations become large barges unable to make swift turns to stay with the changing flow of the market.

CHAPTER II

REVIEW OF LITERATURE

Enterprise Application Integration or EAI is a very generic term. The concept of EAI is interpreted differently by many people [52]. For the purpose of this paper, Enterprise Application Integration or EAI is considered the concept of interconnecting many business systems in order to facilitate knowledge sharing and business logic between systems.

There are many *tools* which can be used to implement EAI; tools such as frameworks, patterns, architectures, protocols, message formats, message translators, etc. Each of these tools has their own advantages and liabilities. These tools must be researched in order to make an educated proposal for an EAI implementation for any manufacturing enterprise. This research will be limited to some of the most widely accepted EAI components.

Most manufacturing enterprises have similar requirements for implementing corporate wide integration systems. It is very important that the implementation of EAI aligns with corporate goals. Typical goals revolve around increasing profits through waste reduction and increased sales. The custom printing industry also shares these typical goals. However, the custom printing industry also has some aspects that make it exceptional.

The custom printing industry differs from many traditional manufacturing companies in that it specializes in mass customization with extremely short turnaround

times. Customers expect their product within a few days whether ordering online or in person and they also want their items to be personalized to fit their need.

While many mass customization manufacturers such as Dell allow customers to customize an item by selecting a finite number of options for their products, the custom printing industry offers an infinite number of configurations for products, limited only by the customer's imagination. A quote taken from *Future Perfect* by Davis, "Every buy is customized, every sale is standardized" [10], epitomizes the EAI infrastructure required in the custom printing industry. The supporting EAI of a custom printing corporation must be flexible enough to allow any custom order to be received, while implementing standards everywhere possible. This section is devoted to the discussion of research surrounding EAI and its potential applications in the custom printing industry.

2.1 Mass Customization and the Printing Industry

Mass customization has turned the printing industry into a very unique industry. By personalizing products which a consumer may purchase, it gives a manufacturer the ability to differentiate themselves in the market [11]. Many tools such as Enterprise Resource Planning (ERP) systems have been introduced into this industry to improve efficiency by improving consistency and reducing bottlenecks in distributed business processes [11]. ERP systems tend to be very generic in order to be utilized by many different industries [33]. However, customization of these kinds of systems is required to realize the full potential of these systems in a mass customization industry [11].

The production floor in most mass customization environments is very dynamic [11]. Due to the dynamic nature of the environment, it can be particularly difficult to maintain production floor resources and capabilities.

There are many other risks specific to the mass customization division of the printing industry. For instance, if a timeline slips a month in an industry which builds ocean liners, that may be an acceptable setback which can be absorbed and compensated for by the business and the production processes. However, in the custom printing industry, short turnaround times are demanded by customers [11]. A setback of a few hours could have drastic implications to production facilities and their profit margins.

2.2 What is Enterprise Application Integration?

Enterprise Application Integration (EAI) is the sharing of information and processes between disconnected systems [28]. Originating in the mid-1990s, this was deemed a new approach at connecting existing systems which are typically legacy systems, and enhancing their functionality by connecting them to new systems which provided additional functionality [27, 11].

Lam [26] states that EAI tools are typically made up of an integration broker, messaging middleware and adapters that are pre-built to connect applications to the messaging middleware. The advantage of Enterprise Application Integration is that it provides a common interface layer through which systems communicate rather than traditional integration which required point-to-point connections [27].

Thomas Gulledge [17] describes two classifications of EAI. The terms Big I and Little i are used to label the two types of integration. Big I defines integration systems that are all encompassing, much like an ERP system. All data is stored and shared between business processes in a single system. No external interfacing is required in Big I.

Gulledge defines Little i as integration which connects multiple systems which were never intended to be connected through electronic processes. Since single piece Big I systems (ERP systems) are likely not sufficient in most enterprises due to the constant flux in today's marketplace, Gulledge speculates that pockets of Big I will most always be conjoined through Little i [17].

2.3 Why use Enterprise Application Integration?

EAI answers the problem of how to connect disparate systems built over many years on varying platforms and languages [28]. Corporations typically have applications that use both legacy and modern technologies [26]. Legacy systems were typically built to each serve a single purpose without the intention of connecting to another system [27]. In order to remain competitive in the marketplace, an enterprise may acquire a new system to leverage its functionality, but it must integrate with the legacy systems which contain the data it must operate on [27].

2.4 EAI Integration Styles and Patterns

According to Hohpe and Woolf's book Enterprise Integration Patterns, there are 4 styles of EAI integration patterns [21]. The four styles are: remote procedure invocation,

messaging, shared database and file transfer. Each of these styles has their own advantages and disadvantages. Integration of an application may make use of one or any combination of these styles.

Remote procedure invocation involves calling a piece of business logic which resides in another system [49]. Invoking procedures remotely can be expensive depending on the method used to invoke the procedure [42]. An advantage of remote procedure calls is that each method has a named signature with typed parameters which allows for client side invocation code to be generated automatically based for calling the procedures [16].

Messaging involves the submission of a request for an action to take place. The systems involved in the message transaction can be involved via many mechanisms such as publish/subscribe, solicitation/response, or even fire and forget [22]. Hohpe calls messaging “a pragmatic reaction to the problems of distributed systems” [21].

Messaging offers high scalability and integration into heterogeneous networks [51]. It can also reduce the impact of a failure at a single node in the architecture [51]. Messaging is considered asynchronous which inherently introduces delays into the handling of requests. This produces a disadvantage because it is slower relative to other solutions.

A shared database style of integration requires building interfaces so that all integrated systems update a common data store behind the scenes [21]. There can be a fair amount of work required to update a set of systems so that they share a common data

store. ERP systems lend themselves to this type of integration since they only need to be configured to use the existing data store.

Finally, the file transfer style revolves around generating some sort of file from the source system that will be consumed by the destination system [21]. In the past these kinds of files have been text based [21]. EDI and CSV files are both examples of files that get transmitted from system to system. The current trend is to use XML files for transmitting information between systems [21].

File transfer shares the same disadvantage of messaging in that this method of integration is typically asynchronous. This makes it slow relative to other methods of integration. The main advantage of having an asynchronous style of delivery is that a system is not dependent on another system being available to immediately receive the message [21]. This also produces loose coupling making the design more modular which is desired [21].

2.5 EAI Topologies

There are several popular topologies relating to EAI. Common topologies are hub and spoke, distributed agents, enterprise service bus and point to point integration. The following section describes some of these popular topologies in more detail.

2.5.1 Hub and Spoke Message Brokers

A message broker (Figure 1- Traditional Integration compared to a Message Broker Integration Architecture) is classified as an Enterprise Application Integration

(EAI) technology [23]. Message brokers are systems that receive and then re-distribute messages from a centralized location. Implementing a single centralized message broker produces a hub and spoke style of architecture. The hub contains the metadata relating to routing as well as messages received while the spokes act as adapters which connect to the systems on the outer ring [22].

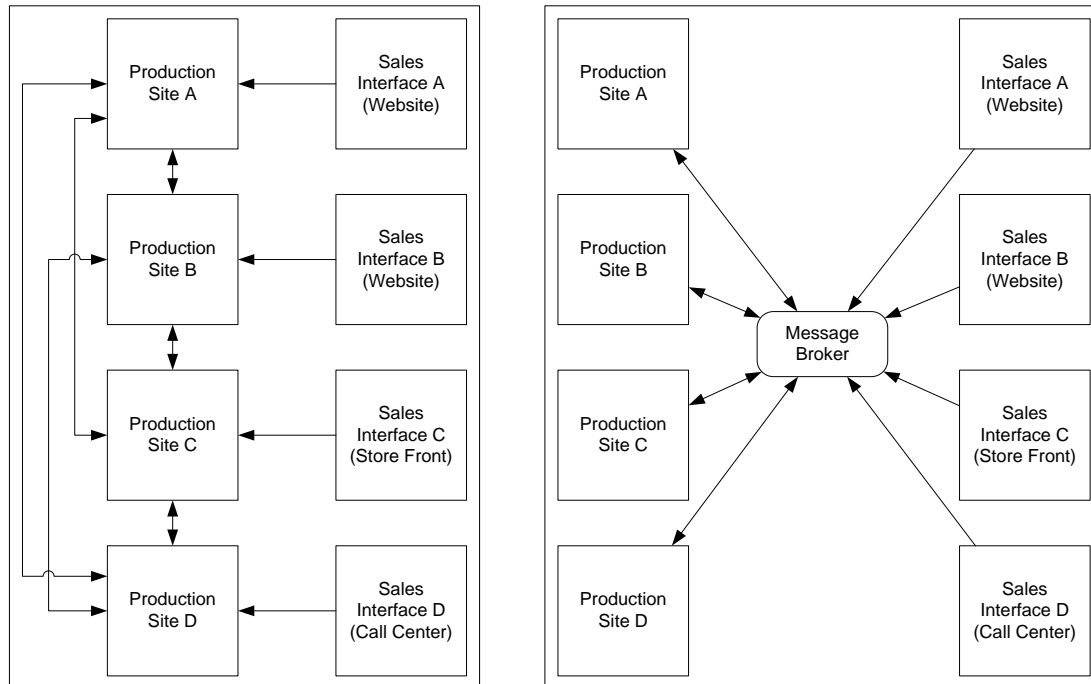
A message broker helps reduce complexity by reducing the number of interfaces between a complex system of applications. The reduction of interfaces for a system means there are fewer systems to update when an update must be made. All interface changes occur in the message broker [23].

Messages can be distributed based on rules set up in the message broker. The decision on where to route the message may be based on any number of factors such as message content, system load or available resources [22].

While routing messages, a broker may also add additional functionality such as the ability to transform messages or save the data to a warehouse [22]. This architecture allows customization of XML document transmission and receiving methods to interact with a wide variety of existing message processing systems [39].

There are many advantages to using a message broker. A message broker provides a lot of flexibility between systems [39]. Another advantage of using a message broker system is that the message sender need not know the specifics of the receiver's implementation. The message broker handles negotiating the communication and data translations that are necessary between sender and receiver [22].

Figure 1- Traditional Integration compared to a Message Broker Integration Architecture



A disadvantage of using a message broker is that the technical complexity and related costs can be quite high [39]. Although the costs can be high, the related maintenance and support costs are lower [5].

2.5.2 Distributed Agents

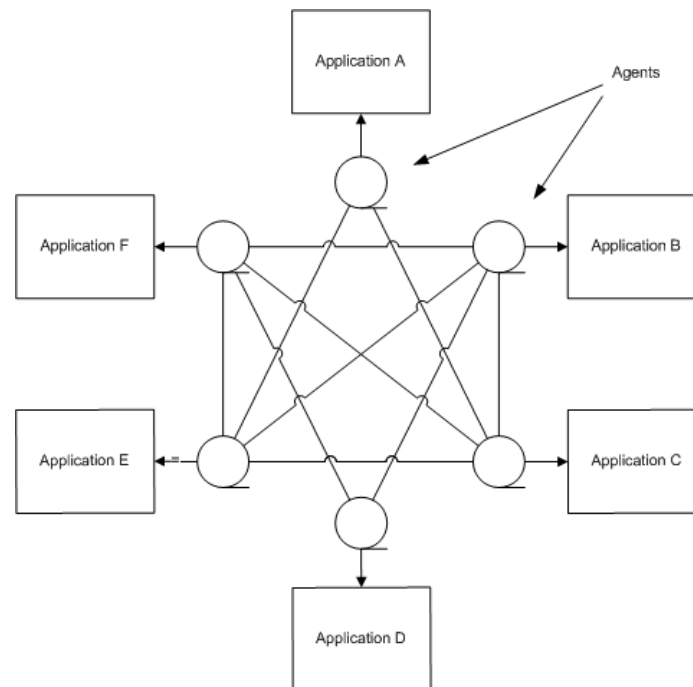
One approach of integrating distributed systems involves using multiple agent programs sometimes called Multi-Agent Systems or MASs (Figure 2 - Distributed Agent Architecture). Agents are programs with specific skills that collaborate with other specialized programs [1]. Distributed agents are becoming a leading edge technology in respect to data sharing [11].

Agents are typically embedded at each conceptual node in a distributed environment. The agents need to understand their own capabilities as well as the capabilities of other agents which are part of the distributed network [11]. Each agent can request and provide feedback to other agents through their common interfaces [11].

The advantage of using agents is that they can communicate with each other to perform tasks such as allocating resources for tasks. There is no centralized agent in the distributed agent architecture. Each agent can operate as a client or a server depending on the direction of the request. Agents are only aware of the systems they can interact with and the environment they have direct influence over [45].

There has been a large amount of research done in the last ten years in the field of agent based integration; however there is a lack of support in the tools for developing and deploying agent based applications [38].

Figure 2 - Distributed Agent Architecture

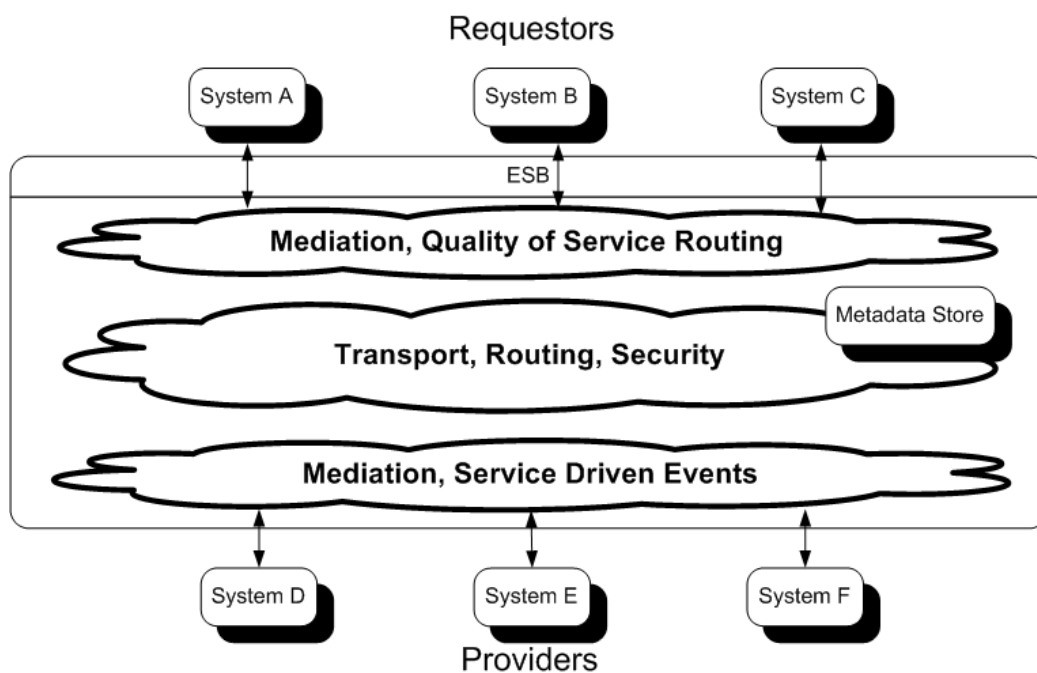


2.5.3 Enterprise Service Bus

An Enterprise Service Bus or ESB (Figure 3 - Enterprise Bus Model (Adapted from Bieberstein et Al, 2005, pg. 44)) provides a method of delivering a message or request from a source to a destination. While nodes on the ESB do connect to a central data pipe, the intelligence does not lie at the central pipe. Each node contains adapters and integration engines [32]. The particulars of the source node are hidden from the destination node and vice versa. The ESB takes care of the mediation details between the two entities [6]. Transport, quality-of-service-based routing, mediation and gateway services are all infrastructure services provided by an ESB [6].

A mediation step is an extra step which can occur on the ESB to provide extra manipulation of a message while it is in transit [37]. Two example mediation manipulations are message translation or encryption [37]. This is done on the bus and the requestor and consumer have no knowledge of the mediations performed by the bus making the bus virtually transparent to the parties involved [37].

Figure 3 - Enterprise Bus Model (Adapted from Bieberstein et Al, 2005, pg. 44)



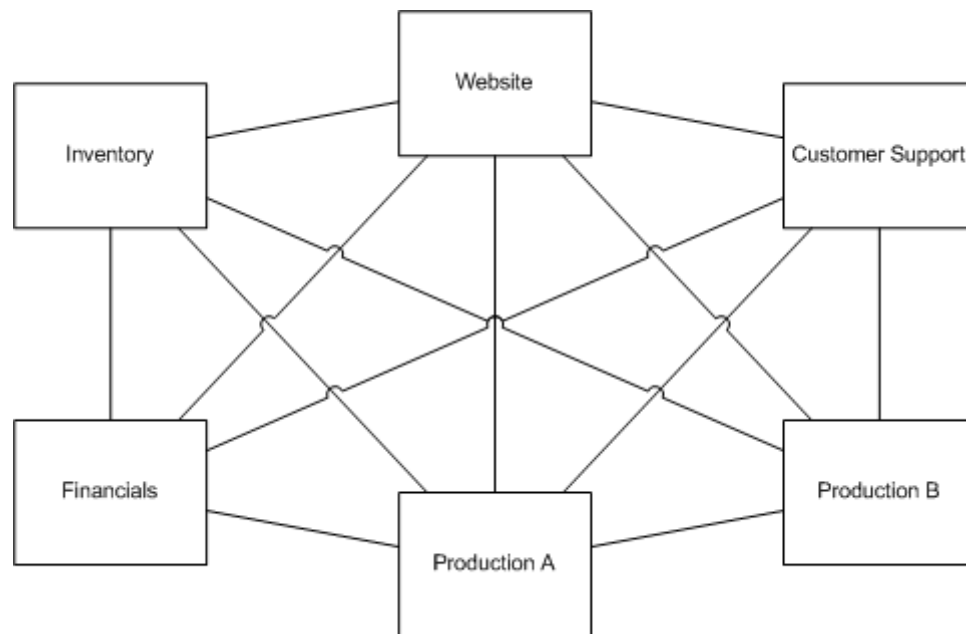
Requests placed on the bus are treated as messages to be delivered to a destination and the bus guarantees the delivery of the message once it has been received [37].

2.5.4 Point to Point

Point to Point or P2P integration (Figure 4 - Point to Point Integration) directly binds one system to another system. This style of integration is typically fast and

efficient because the systems are tightly bound [31]. (P2P) integration, sometimes referred to as traditional integration, is the most expensive form of integration [17]. This design requires a single interface between every pairing of systems that need to work together. As the number of endpoints in a P2P architecture grow, so does the number of interfaces required between systems. P2P integration is usually easy to implement between two systems, however the amount of overall complexity as additional interfaces are created [31].

Figure 4 - Point to Point Integration



2.6 Enterprise Resource Planning (ERP) Systems

Enterprise Resource Planning (ERP) systems are designed to integrate the most common business processes in an enterprise [2]. The systems that ERP applications

typically replace deal with financials, human resources, inventory and all other systems that are in use at a company [2].

While the terms ERP and EAI are occasionally used interchangeably there is a fundamental difference. EAI can at times contain ERP components in situations where multiple ERP systems exist in a single corporation [17]. Also, ERP process integration requires more effort than EAI process integration requires [27].

The use of ERP systems boomed in the 1990s [2]. Ake, Clemons and Cubine attribute ERP's success partially to the mania that went along with the new trend of converting to ERP systems [2]. Companies migrated to ERP systems without concrete proof of return-on-investment [2]. ERP does offer many advantages over existing patchwork applications, but the advantages do not always outweigh the costs.

It is estimated that ERP systems have spoken for 60% to 70% of IT costs since the 1990s and those costs were estimated to be over \$300 billion [2].

Once an ERP system is implemented, its success is vital to an enterprises' survival and profitability [2]. ERP and EAI share many common critical success factors and risks which are discussed later in this chapter.

2.7 EAI vs ERP

While EAI and ERP provide similar results, they do so in very different ways. EAI is best described as a manner of connecting existing systems via an electronic

middleware interface [21]. On the other hand, ERP systems are designed to replace existing systems with a single new system which has all encompassing functionality [27].

The difference between these concepts can be confusing. Both EAI and ERP systems software can be purchased. EAI software differs from ERP software in that it is usually a suite of functionality which allows a development team to perform their own enterprise application integration [21]. The EAI suites can provide adapters for many common systems as well as a means for translating message formats from one to another [22]. Some examples of EAI suites are IBM Websphere MQ, Microsoft BizTalk, TIBCO and WebMethods [47]. Examples of ERP systems are SAP, Peoplesoft, Epicor Enterprise, Microsoft Dynamics AX [48].

While ERP systems replace disparate applications with a single overarching solution, EAI provides a means of integrating existing systems [27]. Both ERP and EAI achieve the same results in providing integrating of business data and processes, but they do so in fundamentally different ways [27].

2.8 Implementation Considerations

Many details must be considered when integrating applications. Failure to identify the risks and rewards of an implementation can put an entire enterprise in jeopardy. Risks must be measured and deemed to be acceptable compared the projected rewards [40].

2.8.1 Cost

Changing an application can be very expensive since “in addition to changing application logic, we need to test, integrate, and redeploy the application within the enterprise” [28]. It is typically more cost effective to connect existing systems as opposed to build new all encompassing systems [28]. According to Bass and Lee, there are three types of costs associated with integration: architecture, integration and operations [5].

The initial cost of developing the solution, purchasing licenses for any tools needed and implementing the solution all fall under the architecture costs [5]. The architecture investment is much higher in an EAI solution compared to a custom integration solution [5].

Bass and Lee also state that integration costs are usually 25 to 40% lower with architected EAI solutions than custom integration solutions [5]. This is because there are many reusable components such as adapters, message translators, guaranteed message delivery functionality, etc. Also, fewer interfaces are needed with an architected EAI solution because there is usually a common data format which messages are translated to. Learning how to effectively use a new EAI architecture in an organization will take time [32]. Time spent on learning new standards, interfaces and best practices all directly relate to the cost of implementing a new EAI architecture [32].

Finally, Bass and Lee classify any cost which is related to ongoing interface development and maintenance as an operation cost [5]. They estimate that operational costs are 50 to 80% lower in an architected EAI solution compared to a custom

integration. This is due to the common data format which drives reusable adapters and translators.

2.8.2 Service Oriented Architectures

Service Oriented Architecture or SOA is becoming increasingly favored in enterprise level development [11]. There are several ways to implement SOAs such as using distributed objects or web services. Web services have become a favored method of implementing SOA's because they are not platform specific.

A service placed between two systems allows those systems to interact without knowing the implementation details of the other [19]. SOA also gives the ability to run different versions of the same service in tandem so that systems using an old service can eventually be converted to use a new similar service independently of each other. You can start off small while thinking about the big picture [8].

2.8.3 Data Formats

There are many standard formats for transmitting information over the Internet. Some corporations use open standard formats such as Electronic Data Interchange or EDI while others opt to use proprietary formats in order to meet the specific needs of their business. Open standards are becoming more important as companies seek to improve communication in their intercompany systems [48, 11].

Open standard formats like EDI formats have been in use since the 1980s [46]. EDI has four major specifications: UN/EDIFACT, ANSI ASC X12, TRADACOMS, and ODETTE [46]. These specifications are much more rigid than self describing XML

formats. EDI also requires advanced technical knowledge of the specification in order to be used properly [7]. Most VANs charge fees to trading partners based on the amount of traffic that goes through their mailbox which gets to be expensive [4].

EDI was originally designed to be compact due to size concerns of messages that were transmitted through expensive VANs. Due to this fact, EDI is machine-readable, but very hard to decipher by a human. EDI specifications have complex and inflexible formats in order to maximize efficiency of data exchange which makes it hard to understand [48]. Also, because EDI is so complex to read, understand, and become knowledgeable about, it is hard to train and retain programmers [36].

More recently eXtensible Markup Language or XML was created and has been gaining in popularity [48]. XML based formats such as commerce XML or cXML have been standardized for transmitting the most common electronic documents related to electronic commerce [9]. The use of XML standards such as cXML has been spreading much faster than EDI did [48].

Also, proprietary formats are perceived to have more value than standardized formats in the market [3].

2.8.4 Data Translation

A network of distributed interfaces is defined here as not the number of distributed plants involved, but by the number of total interfaces in the systems that are involved. In a distributed environment described as such, with n nodes involved, there

can be up to $n/2 * (n-1)$ lanes of communication in the network [20]. If the lanes of communication are directional, then the number of lanes jumps to $n * (n-1)$ [20].

In order to minimize the number of data translations required for all systems to be able to interact, a universally known data format can be described that all other formats can map to. In a healthcare study relating to EAI, it was found that the most important barrier to EAI adoption was the lack of a common standard [25].

2.8.5 Communication

The communication between two systems plays a large role in the system's ability to be flexible and easily modified. Communication between systems should be "loosely coupled" [44]. Being loosely coupled means the specifics of a systems implementation should be able to be updated without affecting other systems that interact with the updated system [19]. By making systems loosely coupled, there is less of a chance for maintenance to one system to require maintenance to another system it connects to [19].

2.9 Legacy EAI Technologies

It is very important to study the history of EAI to understand how certain technology worked, why it was successful, and more importantly why it failed. Understanding the weak points of EAI implementations will assist in speculating how ideal integration system should be implemented.

2.9.1 CORBA

The Common Object Request Broker Architecture (CORBA) is a standardized protocol developed by the Object Management Group (OMG). CORBA allows interoperability between systems and is used for distributed object computing [30]. This allows developers to easily build heterogeneous applications. Up until CORBA was invented, developers needed to manually develop protocols to achieve interoperability between systems. In the late 90s companies began building systems to work over the web with HTTP, Java and Enterprise Java Beans (EJB). This technology started out very strong in the mid 1990s, but after 15 years it is “a niche technology that exists in relative obscurity” [18]. CORBA was late to keep up with the changing market but was able to put out the CORBA Component Model (CCM) in 1999. The delay of meeting the market’s demand helped contribute to its decreased usage.

CORBA also required the user to open a port in the enterprise’s firewall for each service exposed. Other serious problems with CORBA such as its bloated size, development and runtime costs, complexity and type safety limitations contributed to its diminished usage. [18]

2.9.2 DCOM

Microsoft attempted to capture the integration market by developing the Distributed Component Object Model (DCOM). DCOM is a Windows only implementation used in a similar fashion to CORBA.

This model, like CORBA, failed to capture the middleware market. DCOM could only run on Windows server. This was a limitation meaning many legacy systems could

not be integrated with newer enterprise components. The integration of legacy systems has been determined to be a critical part of EAI as discussed earlier. A further limitation of DCOM is that it performed more poorly than CORBA [29].

Microsoft later came to understand the limitations of DCOM. Instead of giving up on the middleware market, Microsoft set to change it. In partner with DevelopMentor, Microsoft developed a specification called SOAP. SOAP is a protocol that is based on eXtensible Markup Language (XML). SOAP was then handed off to W3C for standardization. The development of SOAP and XML in combination with the downfall of .coms in 2001 caused most enterprises to rethink how they were implementing their integration systems. This led to a drastic reduction in the use of the expensive CORBA system and added in the market's migration toward SOAP enabled systems [18].

2.10 Critical Success Factors and Pitfalls

While ERP and EAI systems are very similar, there is one very key difference. Since ERP systems usually replace existing systems with an all-encompassing system, this usually requires the business processes which exist in a corporation to be modified to align with the limitations of the ERP system before the system is implemented [26].

A limitation of EAI is that it requires the players implementing a solution to have an intimate knowledge of the existing business processes, data and systems which are currently in place [26]. If the individuals implementing the EAI solution do not have the skill set required to define the necessary integration points, they could be a major stumbling block which and the project would have a higher chance of failure.

There are so many EAI tools and patterns that looking into every system will undoubtedly cause analysis paralysis where the team never gets past the planning stage. It is important to consider what functionality your business requires and then apply that knowledge as a filter when selecting an EAI solution to expedite the planning stage [32].

Many studies have attempted to identify the critical success factors in implementing an ERP system in order to increase the chances of a sustained and successful implementation. Some common critical success factors identified in these studies involve: setting corporate strategic goals, high commitment from all employees, proper software selection, comprehensive training, business process reengineering, and executive leadership [43, 13].

The process of implementing an ERP system is complex and risky [40]. It requires a considerable amount of resources during the planning and implementation process [40]. Also, those resources are put at risk during the implementation phase [15]. Due to the high costs of software and the amount of programming required by ERP systems, EAI emerged as a lower cost alternative which required less programming [27].

CHAPTER III

A PROPOSED EAI ARCHITECTURE

This chapter proposes an EAI architecture for a distributed custom printing enterprise. The technologies selected should present the best possible solution for sharing data and processes in a distributed printing environment. It is important to keep in mind the unique aspects of a distributed production environment. By using the research presented, this paper will attempt to propose the best possible EAI architecture for the distributed printing enterprise relating to mass customization.

3.1 Explanation of Business Requirements

Business requirements mold the look of an EAI implementation. Requirements such as peak throughput during busy seasons, the number of business partners, and even the degree of distribution an enterprise has can dictate the best EAI installation for an enterprise. The following items are typical business requirements in a distributed printing environment.

3.1.1 Fast Turnaround Times

In the realm of mass customization, turn-around times for a product are extremely competitive between printers. Customers are empowered by the use of the Internet to be very selective in the printer they choose for their custom printed product. An EAI implementation must be designed to minimize the time in which an order takes to be received and produced. If a customer has a negative experience with placing their order,

there is no shortage of competitors in the online market that the customer can take their business to.

3.1.2 Market Adaptability

Over the past 30 years, the Internet has drastically changed the look of the custom printing industry. Changes such as the replacement of traditional printing presses with higher capacity digital presses are constantly occurring in the printing industry. Many enterprises have adopted the business model of acquiring startups which are successful in newly developing fields in order to stay competitive in their markets [12, 34, 35, 41]. The constantly changing structures of enterprises as well as the fast paced fluctuations in the market require that businesses can adapt. Some do so by shrinking, expanding, eliminating and acquiring systems in order to fulfill the needs of customers. This creates the need for flexible and modular systems.

3.1.3 Flexible and Modular Systems

The architecture for a distributed printing enterprise should lend itself to simple and fast *flexing* in the business. We will describe flexing as the increase or decrease of a certain aspect of a system. In this context, flexing occurs in the capacity of an individual production facility as well as the number of production facilities in the enterprise.

In the custom printing industry, there are many seasonally produced items. Graduation announcements, holiday cards as wedding invitations all have a busy season which requires workforce and capacity to be flexed to meet demands of production.

A business must be able to quickly react to busy seasons, especially when the current capacity does not match the market's demand. Depending on the market, a business may need to ramp up or ramp down its capacity to produce work. This is done via workforce, inventory and even equipment adjustments.

Flexible and modular systems give plants the ability to modify their structure in response to business demands. Changes to an enterprise's structure should only occur if facts can be provided to justify the change. This requires visibility to data and implementing business intelligence based on knowledge gleaned from the data.

3.1.4 Data Access and Business Intelligence

It is critical for every enterprise to have access to the appropriate data in order to shape its core business decisions. It can be a challenge accessing data across many distributed databases when database platforms differ and the structure of the database tables is inconsistent. An EAI implementation should help enable systems to access necessary data across a business. It should also help foster an environment where data driven business decisions are automated.

Through the proper application of business intelligence to make informed business decisions, a business can operate dynamically to meet business needs. The custom printing industry is like many production industries in that product pricing needs to be competitive. Product prices are affected by operating costs. To reduce operating costs, it is important that a business only reserves the minimum required resources to meet market demand. Effectively managing resources in a flexible production

environment ensures production costs are minimized which allows decreased product prices and maximized profit margins on products.

3.1.5 Managing Resources and Costs

A successful printing enterprise has the ability to access knowledge from data and the ability to flex its systems in order to adapt to the market, and even more importantly the ability to know when it is appropriate to do so. Managing resources in a dynamic enterprise is a very powerful action. Resources come in many forms such as inventory, presses, personnel and orders. Properly allocating these resources at distributed printing plants may help drive down production costs and product prices while creating higher profit margins.

3.2 Technical Requirements

This section will outline the proposed technical requirements of an EAI system that can be implemented in a distributed custom printing production environment. The justification for each technical requirement is described with each section

3.2.1 EAI or ERP

The proposed solution will implement EAI instead of ERP. The justification for this decision is based on the fact that EAI architectures allow systems to be integrated over time instead of requiring the replacement of an entire component all at once as with ERP. The notion of EAI allowing systems to evolve compared with ERP requiring systems to be replaced is very important.

EAI allows an enterprise to migrate its existing systems into an EAI style of architecture. ERP on the other hand requires more wide-sweeping changes to be made when an ERP system replaces entire business components with a prebuilt package.

ERP requires companies to conform to the existing functionality of the ERP system [27]. An acquisition would need to replace their business systems with the ERP modules in use at an enterprise while EAI allows an acquisition to interface into the enterprise without needing to adjust their internal processes. In many cases, a company is acquired because of their outstanding knowledge in a new field related to custom printing. Requiring a business to retool to meet the requirements of an ERP system would redirect their attention from the niche market they were typically purchased for being knowledgeable about.

Another advantage of EAI is that it can be implemented a system at a time instead of through broad sweeping replacement of existing systems. This helps manage the costs of system integration by connecting only select integration points that are deemed to provide the best return on investment.

3.2.2 Data Format

Based on the benefits outlined in the research conducted, the proposed EAI system will have a common data format. The common data format will be XML because customers submitting imprint information for their products may have very different imprint information. XML is extensible which will allow the dynamic format of imprint information to be passed inside of it. The XML can still be validated against a schema to

ensure the document has a proper format while being flexible enough to contain the dynamic imprint information.

3.2.3 Data Translation

The XML message format will lend itself to being easily translated to different XML formats through the use of EAI suites such as Microsoft BizTalk. Tools similar to BizTalk use eXtensible Stylesheet Language (XSL) translations, also known as XSLT to convert between XML formats. BizTalk also allows XML messages to be translated into EDI and flat file message types such as CSV which external systems may require.

3.2.4 Communication

Research suggests that loosely coupled systems allow for more flexible system integration [10]. System integration flexibility benefits the business requirements of fast turnaround times, market adaptability and having flexible and modular systems. Research confirms that service oriented architectures allow for systems to be integrated without systems knowing the implementation details of each other [50]. Therefore, communication via services will allow data and business processes to be shared between systems while remaining loosely coupled and flexible. The type of service implemented should not require systems operate on the same operating system platform because that limitation contributed to the downfall of CORBA and DCOM as outlined in Chapter 2.

3.2.5 EAI Topology

The proposed EAI architecture will have a message-broker style topology. As companies are acquired, they can join this EAI architecture by utilizing exposed interfaces. Existing companies can also be easily removed from the architecture without affecting other company's integration interfaces. The number of interfaces required by this topology compared with traditional P2P integration make it a better choice [20].

3.2.6 Summary of Technical Requirements

The technical requirements proposed for an EAI implementation have been driven by the business requirements outlined and integration research conducted. Here is a summary of the technical requirements.

Enterprise Application Integration will be used in conjunction with a message broker topology. Communication will take place via a service-oriented architecture. A universal data format will reduce the number of translations required between systems. The hub will provide functionality to translate messages. The hub will also provide functionality to route messages. The hub will distribute copies of relevant data to a data warehouse. Data mining will allow informed business decisions to be made which will produce business rules that will be implemented in the hub and manifest themselves through message translation and routing functionality.

CHAPTER IV

SIMULATION OF PROPOSED ARCHITECTURE

4.1 Overview

The simulation outlined here will compare a message-broker based topology against a traditional point-to-point (P2P) topology. This simulation should help approximate the performance of each topology. Simulating these models under varying conditions will help identify if this topology is a good fit for a distributed production environment.

4.2 Simulation Process

This section will outline the specifics of the simulation relating to the implementation of the simulations and how results will be gathered from the tests.

4.2.1 Simulation Overview

The message broker topology used in the simulations is shown in Figure 5 - Message Broker Topology. The traditional P2P topology is shown in Figure 6 - Traditional P2P Topology. All six simulations conducted use the same number of customer and plants.

Four customers will be created: Customer1, Customer2, Customer3 and Customer4. Each customer is able to produce the same items in all simulations. Only three items are used in all simulations and they are given arbitrary values of A, B and C.

Four plants will also be created in each simulation. The plants will be given

Figure 5 - Message Broker Topology

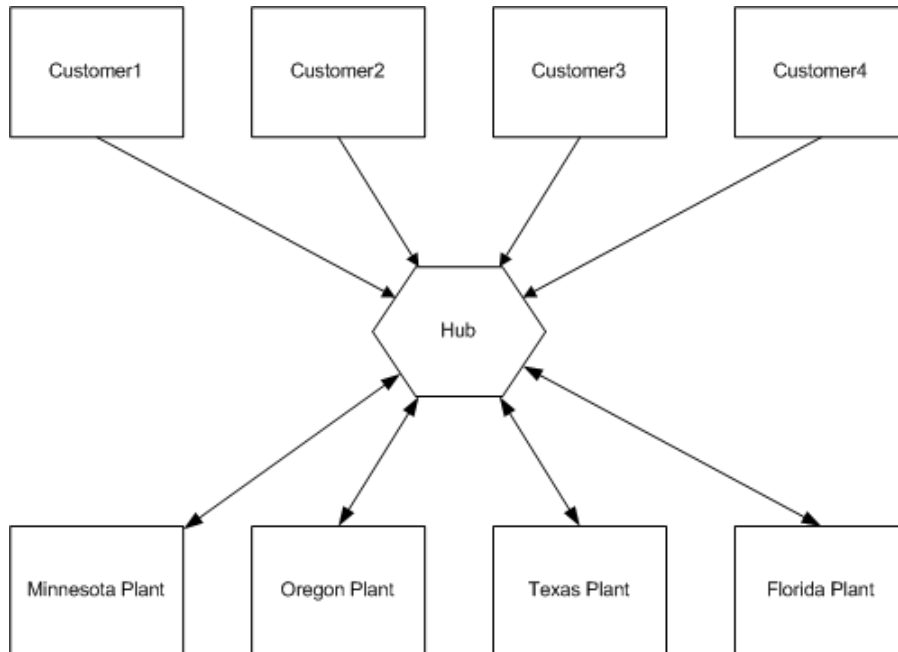
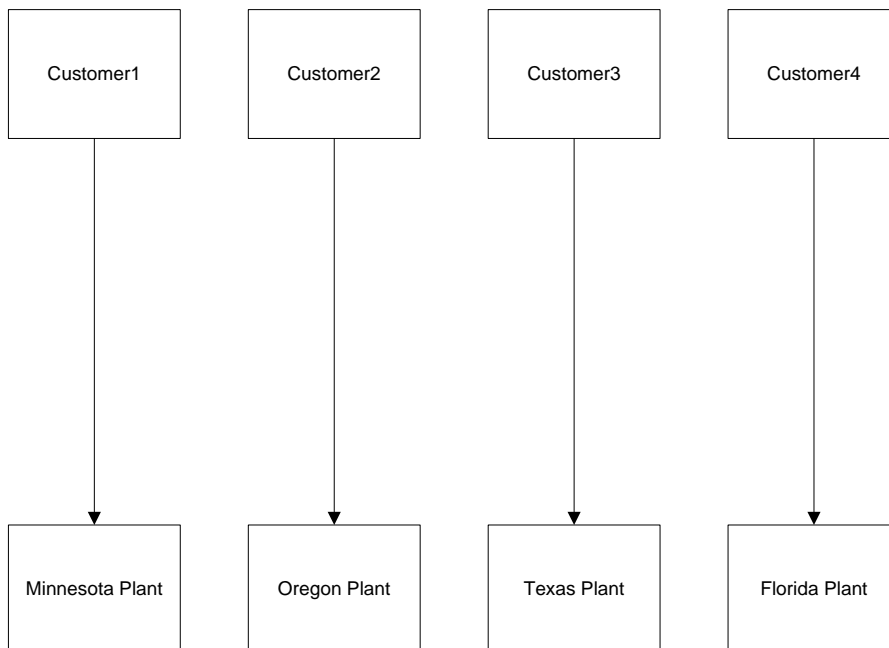


Figure 6 - Traditional P2P Topology



the arbitrary names Minnesota, Oregon, Texas and Florida.

Finally, as shown in Figure 5 - Message Broker Topology, a hub will be constructed to mitigate the delivery of orders between customers and the production facilities. In the message broker topology, the plants must request work when they would like to receive additional orders. This differs from the P2P topology in that the plants have full access to all orders the customer has sent to them. The plants do not have access to each other's orders, however.

In the hub, orders will be distributed on a first come first served basis by retrieving the list of all orders ready to be processed and selecting the one with the earliest creation date.

Metrics will be collected relating to the flow of orders through each plant. These metrics will help us draw conclusions about the performance of each model. It will also help create assumptions of each topologies performance in a real world setting.

The following metrics will be collected:

- The number of orders produced at a plant.
- The average duration of time in milliseconds required to produce an order.
- The total duration of time for all orders to go through production at the plant.
- The timestamp of the first order created.
- The timestamp of the last order produced.
- The longest duration of time for a single order to be produced.

4.2.2 Test Details

Three different situations will be simulated for both the message broker and P2P topologies. The three simulations are outlined next.

The first simulation will test both topologies ability to handle normal workload. In these simulations, we will assume that an enterprise is operating at *normal* production levels when the combined output of all plants operating near capacity are able to produce the combined order requests of all customers at nearly the same average rate at which customers are submitting their orders over an extended period of time. It is also assumed that an enterprise's capacity is driven by the average workload created by its customer's orders over an extended period of time. It is assumed that an enterprise would maximize its capacity in this manner in order to minimize its operating costs and maximize its profits.

The second simulation will test both topologies ability to handle a load of work greater than can be produced at the same rate as the customer submits work. In the real world, this could occur due to an advertising campaign or a reduction in an item's cost. This simulation will be completed after the normal rate of production has been established. The rates that customer's submit orders in the normal production scenario will be increased. The numbers of orders that a customer submits will also be increased to simulate a heavier workload. The rates will be increased the same for both the message broker and P2P simulations.

The final simulation will compare both topologies ability to handle an otherwise normal workload when a production facility has been impaired. In both the message broker and P2P topologies, the Oregon plant will have its ability to produce orders severely impaired. This situation could occur in the real world if equipment breaks or if there is a reduction in personnel. The results of this simulation should provide details regarding an enterprises ability to handle severe production impairment and the impact it has on the turnaround times for customer's orders. To simulate an impairment at the Oregon plant, the capacity of the plant will be severely reduced.

4.2.3 Environment

The simulation of the two models will be performed with .NET console applications created with the .NET 2.0 Framework using the c# programming language. Both topology simulations will be conducted on a single computer. The test computer has Windows XP Professional with Service Pack 3, 2GB of RAM and an Intel Core2 CPU T7600 processor @ 2.33GHz. In order to reduce the I/O delay of writing files to disk, simulations will be conducted on a virtual hard drive which resides in RAM. The size of the virtual drive is 40MB. The configurations which were tested required only 5MB of the 40 MB virtual drive which was created.

4.2.4 File Formats

Orders will be produced by a customer and sent either to the hub or directly to the plant depending on the topology being tested. Each order will have only one item. For

the purposes of this simulation orders will be comma separated value (csv) flat files in formats described below (Table 1- Fields in an Order).

Sample Order Text: 456,58,A,2008-11-02 09:32:01:953,2008-11-02 09:32:27:562.

Table 1- Fields in an Order

Ordinal	Sample Value	Explanation
1	456	Customer Id
2	58	An order number unique to the customer
3	A	The item id for the item being requested.
4	2008-11-02 09:32:01:953	The timestamp relating to the creation of the order in the format Year-Month-Day Hour:Minute:Second:Millisecond
5	2008-11-02 09:32:27:562	The timestamp relating to the production of the order in the format Year-Month-Day Hour:Minute:Second:Millisecond

Windows console applications will be created to simulate a customer and plant in both the message broker and P2P simulations. The message broker simulation will have an additional console application which will regulate the flow of messages between a customer and a plant. In the message broker implementation, a customer will submit orders to the hubs inbox regardless of item requested in the order. When a hub receives a request for work from a plant, it will distribute work to the plant based on the plants capabilities. A plants request for work will be a csv file in the format shown below (Table 2 - Fields in a Work Request).

Sample Order Request Text: A|B|C,C:\Texas\In

Table 2 - Fields in a Work Request

Ordinal	Sample Value	Explanation
1	A B C	This is a pipe delimited list of items that can be produced by a plant. The values A, B and C are fictitious item names.
2	C: \Texas\In	This is the file directory path to the inbox for a plant. The example shows the inbox path for the Texas plant.

In the message broker architecture, the hub will receive orders from a customer and group them into folders relating to the item requested in the order. The order will then be moved from that holding queue and placed into a plant's inbox when that plant places an order request and is capable of producing that orders item. All orders are processed in a first-come first-served basis.

4.2.5 Console Applications

The customer, plant and hub console applications each have parameters which affect how they function.

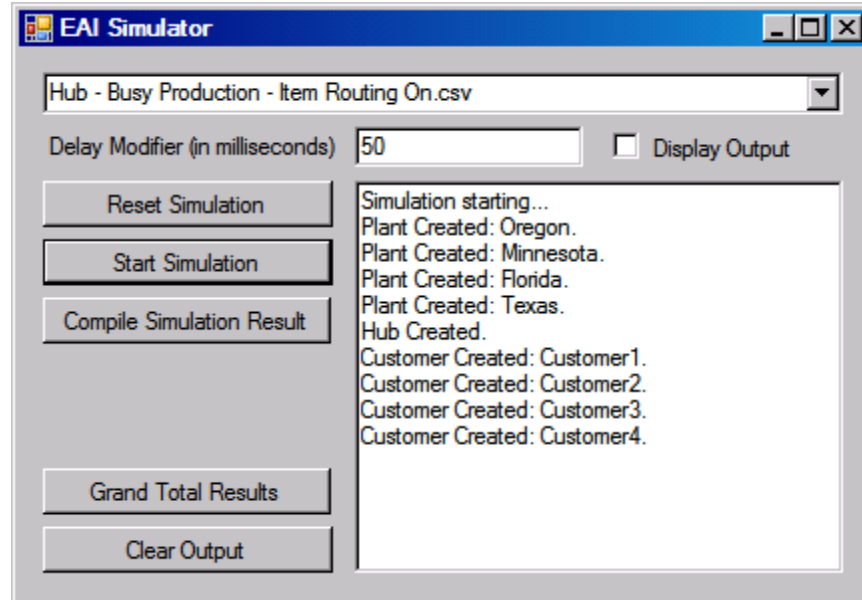
The customer console application takes in a customer's name, id, items which they could possibly request, the unit of time delay between each order, the maximum number of orders a customer will place, and the location a customer will send the orders to. For each order, a customer will randomly choose one of the items listed that they

could possibly request. The customer application will generate an order containing its customer id, an integer unique to the customer for the order id, and a timestamp indicating when the order was created.

The plant console application requires a parameter of H or T describing whether the plant is configured to operate with a Hub or implements traditional integration respectively. The plant console application also requires the following values as parameters: a plant's name, id, items which the plant can produce, the capacity for concurrent work at the plant, the unit of time delay required to produce each item, the path to the plant's order inbox, and the location the plant should submit work requests to if it is part of a hub simulation. Upon production of an order, the orders order produced column is filled in by the plant.

The hub console application takes in one parameter. It is the units of time delay that the hub should delay when attempting to process work from a customer as well as submit work to the plants.

Configuration files will be created to expedite setup time for simulations. Each configuration file lists the console applications which will be started and the parameters they should run with. The details of each configuration file are listed in Appendix C. There will be one windows application which will be used to initiate tests. A screenshot of the simulation program is shown below.



4.2.6 Result Format

Upon completion of each test the orders will be dissected to gain information about the performance of the model under the given test conditions. The listed in the introduction of this chapter will be gathered for each plant to compare the overall performance of the enterprise between the two models. This information will then be graphed to make the information easier to compare.

4.3 Hypothesis

The message broker topology should perform very comparably with the point-to-point topology under normal and busy workloads. There may be a slight performance decrease due to the fact that the hub will need to distribute work to a plant based on the capabilities of the plant. If performance is degraded, it should be only slightly. The benefits of the message broker with intelligent work distribution should become very

clear in the limited production test. In the limited production capabilities test, the hub model is anticipated to greatly outperform the traditional model by reducing the maximum turnaround time for an order.

4.4 Simulation Results

Equilibrium for *normal* production was found by creating an order every 10 units of time in the P2P topology. Each plant's inbox was monitored to see if the plant was keeping pace with the customer's requests without building up a backlog of work while also maintaining its workload near capacity. A plant with capacity to produce 5 items concurrently must produce an item every 50 units of time to keep up customer demand while maintaining its workload near capacity. The message broker topology was then run with the same capacity and delay parameters as the P2P topology at normal workload. The message broker was also able to produce the items without creating a backlog of work while keeping running near capacity. The simulations were run three times each and the averages of the results were used to create the following charts.

4.5 Results Summary

After running the three simulations, information was tallied for each plant of each simulation. Information was gathered on the number of orders produced at each plant, the average duration of time in milliseconds that an order took to be produced, and the longest duration of time that an order took to be produced per plant. The following charts outline the results of the simulations.

4.5.1 Number of Orders Produced

The comparison of the number of orders produced shows that the P2P topology routed every order to its respective plant as expected (Figure 7 – Normal Scenario - Number of Orders Produced). In the *limited* capacity simulation and the *normal* simulation, the P2P architecture produced 1,000 orders at each plant for each simulation and 1,250 orders at each plant in the *busy* production scenario.

In the *normal* (Figure 7) and *busy* (Figure 8) simulation, the hub routed orders almost exactly the same as the P2P model. However, hub topology distributed the workload very differently in the limited capacity simulation (Figure 9). This was due to the fact that the Oregon plant was unable to meet the production need of the customer. Plants Florida, Texas and Minnesota all consumed additional orders to meet the demand of the market.

Figure 7 – Normal Scenario - Number of Orders Produced

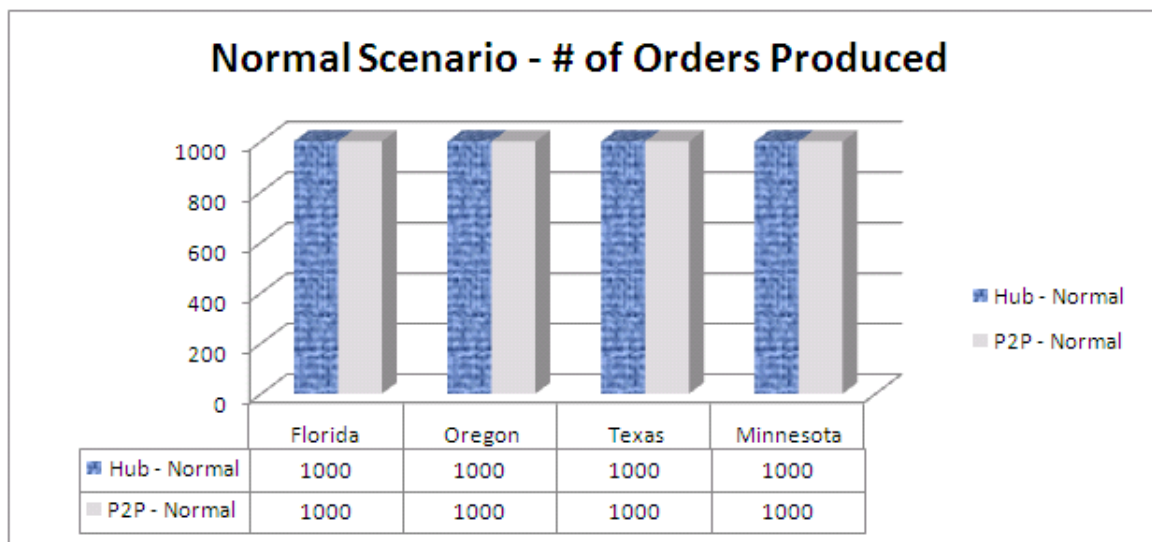


Figure 8 - Busy Scenario - Number of Orders Produced

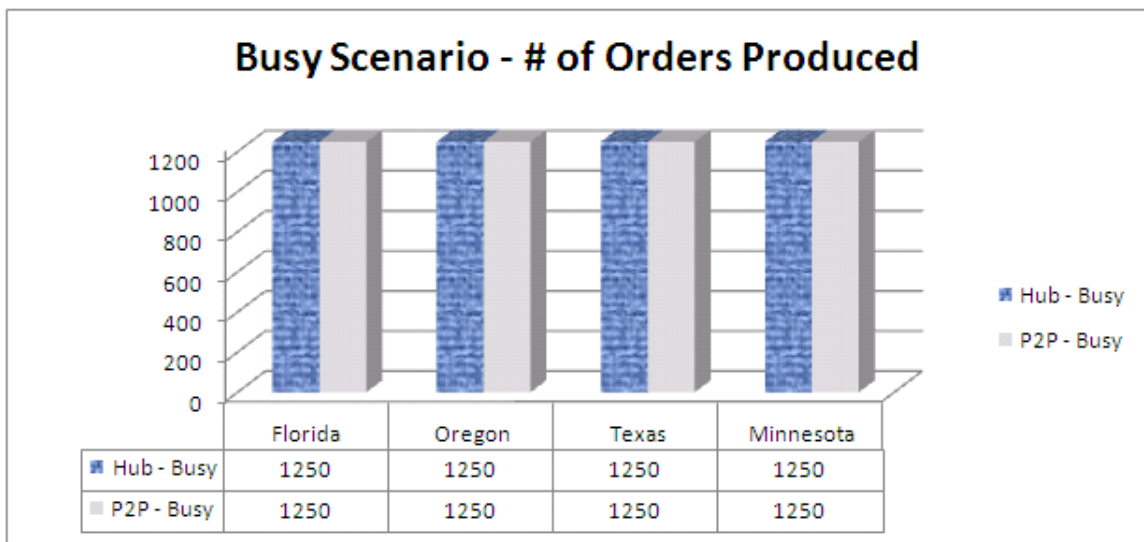
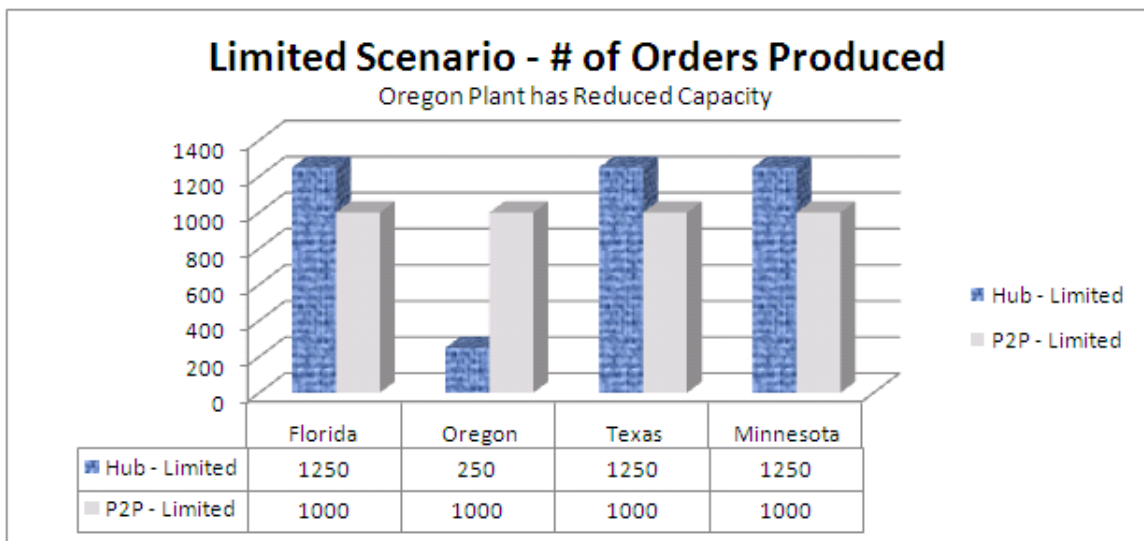


Figure 9 – Limited Scenario - Number of Orders Produced



4.5.2 Longest Production Duration in Milliseconds

The chart shown in Figure 11, which describes the longest production duration in milliseconds, shows that the *busy* simulation for both the message broker topology and the P2P topology seemed to cause the longest wait times on average compared to the

other two scenarios shown in Figure 10 and 11. The P2P topology produced the worst performance relating to the longest production duration in the *busy* simulation.

The *normal* workload (Figure 10) simulation shows that the traditional model performed slightly better than the message broker model. The time separation between these models is very minimal.

The *limited* capacity scenario (Figure 12) showed that the hub did spend slightly more time processing orders, but the time was distributed between all plants. The P2P chart on the other hand shows the limitation at the Oregon plant greatly affected the duration of time that it took for the slowest order to be produced.

Figure 10 – Normal Scenario - Longest Production Duration in Milliseconds

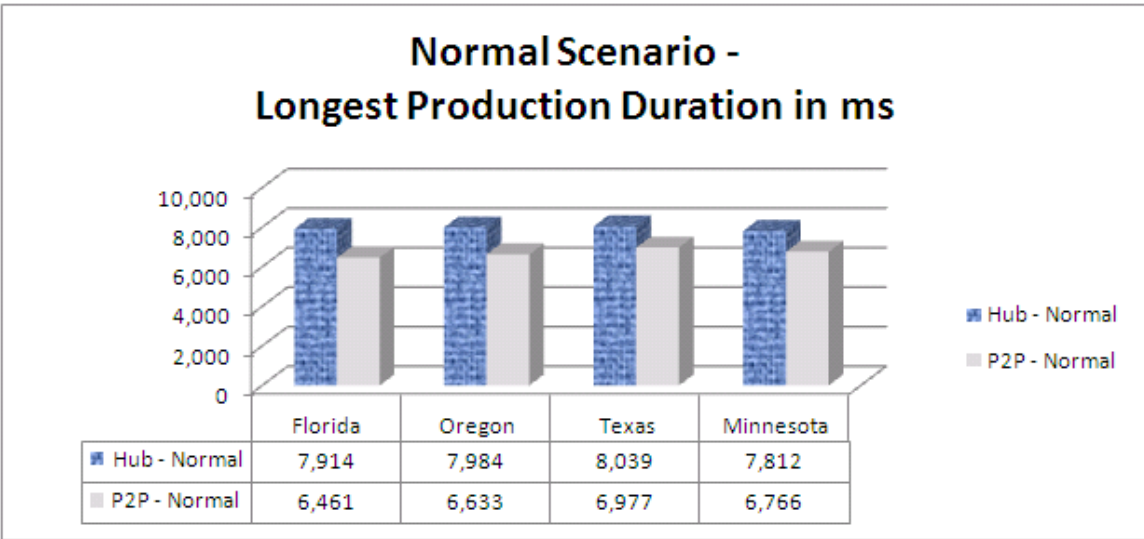


Figure 11 - Busy Scenario - Longest Production Duration in Milliseconds

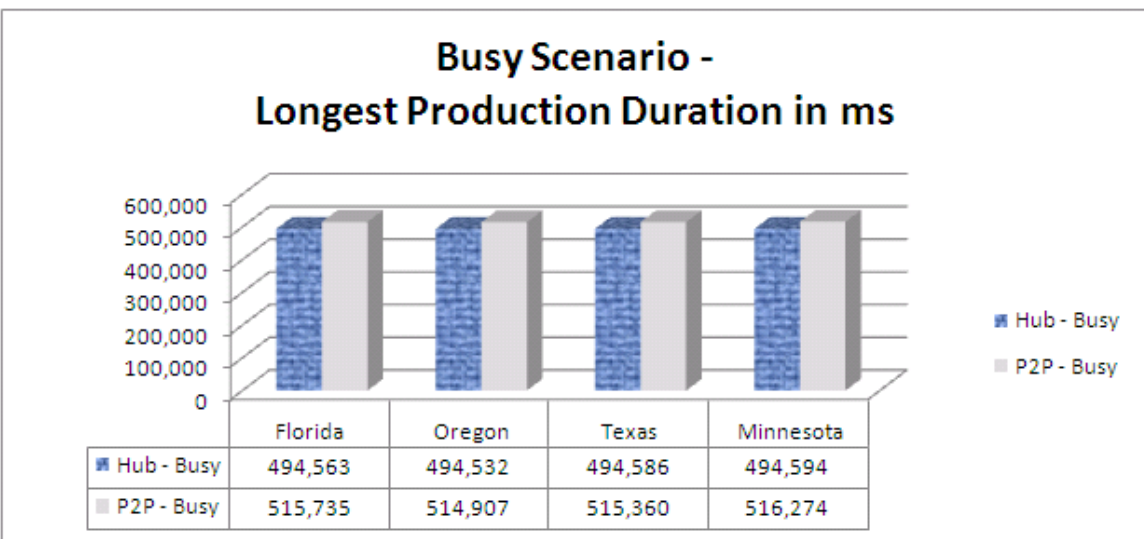
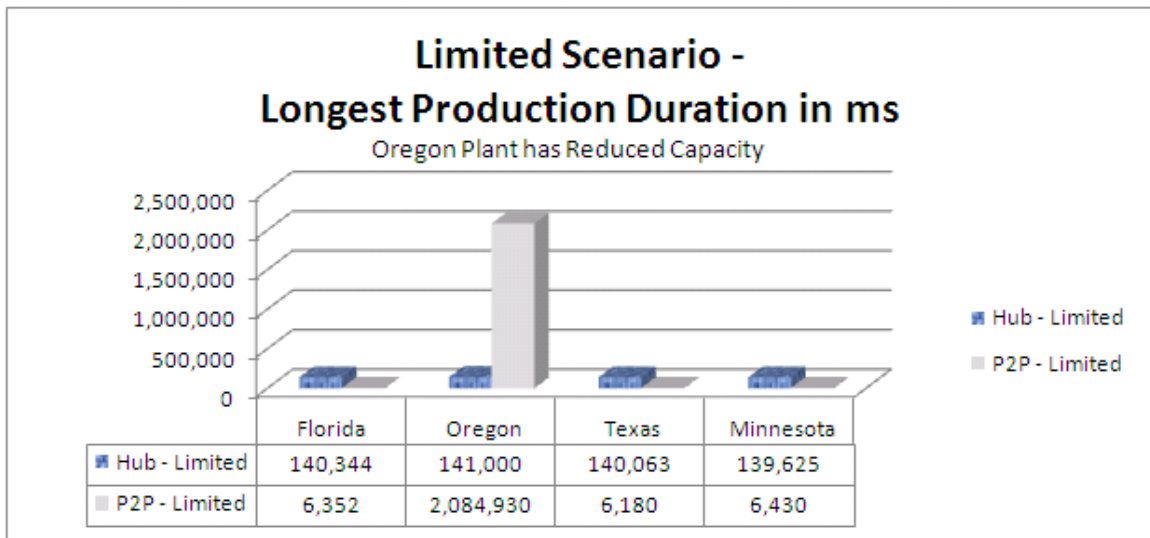


Figure 12 – Limited Scenario - Longest Production Duration in Milliseconds



4.5.3 Average Production Duration in Milliseconds

Figures 13, 14 and 15 show the average production duration in milliseconds for all three scenarios. The *normal* production simulation shows that the P2P topology produced items slightly faster than the message broker topology (Figure 13).

The *limited* capacity scenario shows that the message broker topology produced items slightly slower when comparing the Florida, Texas and Minnesota plants (Figure 15).

Figure 13 – Normal Scenario - Average Production Duration in Milliseconds

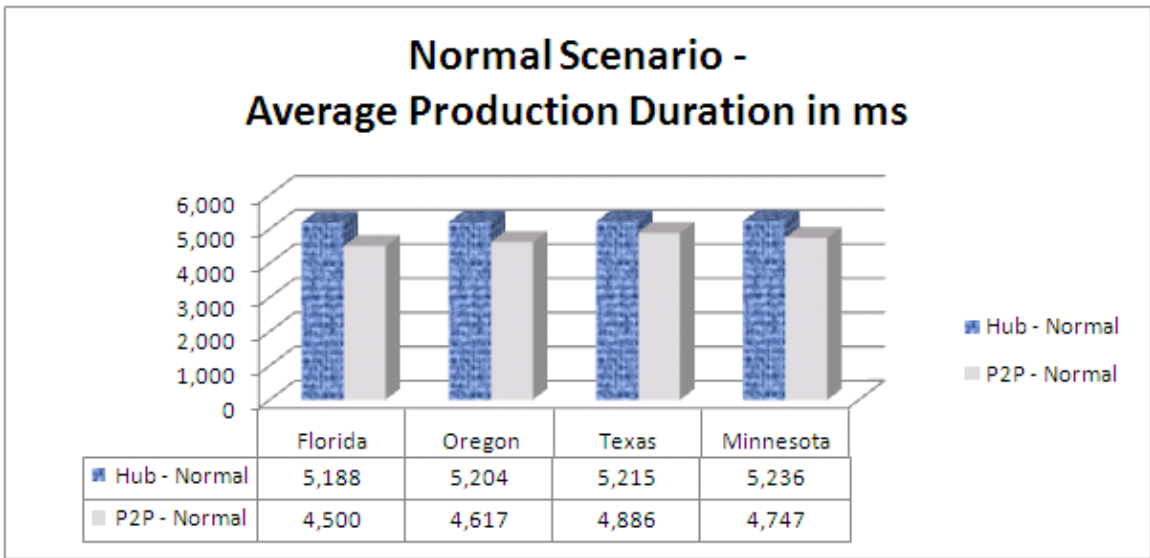


Figure 14 - Busy Scenario - Average Production Duration in Milliseconds

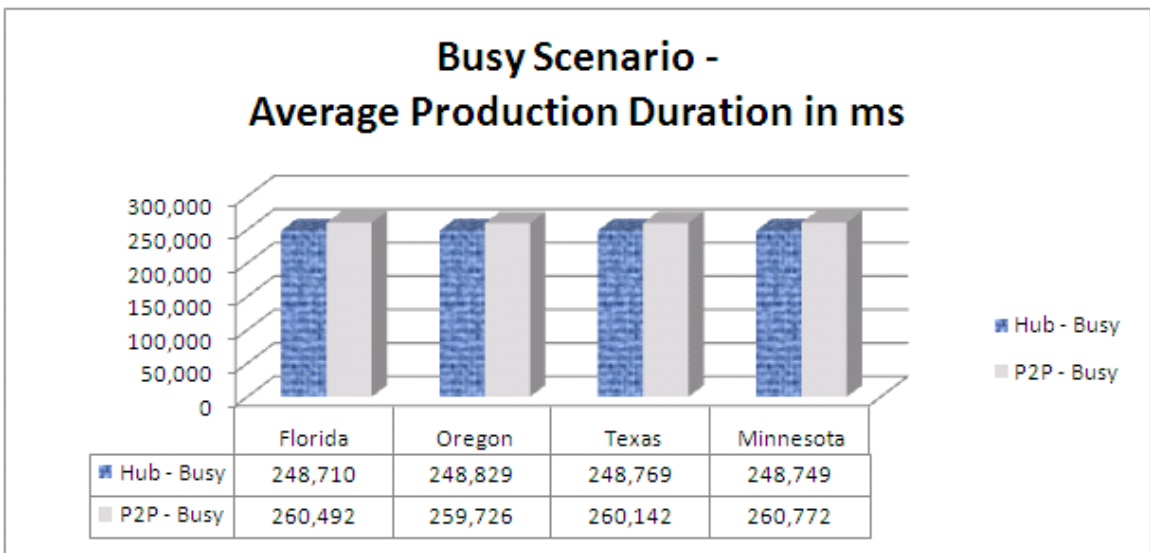
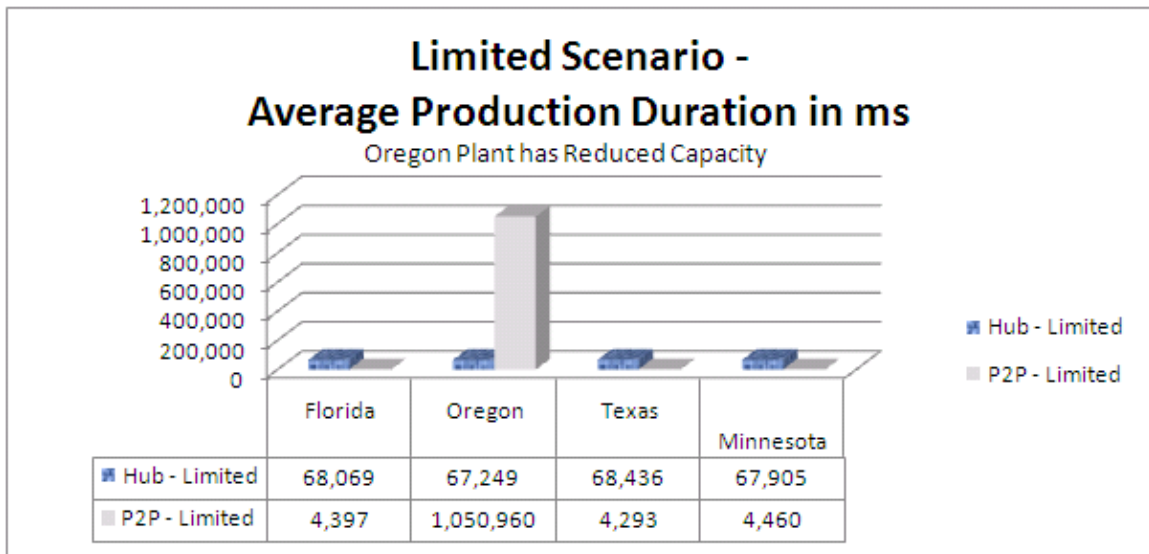


Figure 15 - Limited Scenario - Average Production Duration in Milliseconds



However, the Oregon plant in the P2P model took exceptionally more time to produce all of its orders. The message broker topology shared the work between all four plants which accounts for the slightly elevated production times. The difference between the performances at the Oregon plants is easy to see when viewing Figure 15.

Finally, the *busy* workload (Figure 14) simulation shows that the P2P production plants took longer to produce items than the message broker topology did.

CHAPTER V

CONCLUSIONS

5.1 Summary

Reviewing the results outlined in Chapter 4 shows that the production capabilities between the message broker and P2P topology are very comparable. The greatest difference is highlighted by the production capabilities of the Oregon plant when the plant's production capabilities were set to simulate reduced capacity. The delays at the Oregon plant would most likely be unacceptable from a customer's perspective. When many custom printing plants tout their abilities to produce customized items quickly, the customer expects to receive their item quickly.

The delay in production at the Oregon plant could potentially cause customers to go to a competitor in the future which is not an acceptable risk. The small performance hits relating to the average production duration that were seen under *normal* production did not seem to be a risk which outweighed the benefits of the message broker topology.

As outlined in the problem statement, custom printing enterprises can become large cumbersome companies through years of acquisitions and mergers, which potentially spawn tightly-coupled systems that make an enterprise inflexible and brittle. Printing enterprises have the potential to become faster and maneuverable by combining technologies that support loose coupling and an EAI topology. The message broker's ability to outperform the P2P model under heavier load in respect to average production

duration in milliseconds seems to make it a better choice, but an implementation is needed to confirm or refute this observation.

5.2 Research Limitations

This research attempts to compare the performance of the message broker topology against the traditional point-to-point integration topology while keeping in mind the business requirements of a custom printing enterprise. There are many factors which are not taken into account with the research and simulations which were conducted. These limitations are outlined here and may lead into areas for future research.

5.2.1 Further Topology Comparisons

Additional topologies should be considered and compared to the message broker topology. The message broker appears to be a better choice by minimizing the risk of a customer's order being seriously delayed in a limited production scenario, but there may be another topology which can minimize this risk while also performing better under normal conditions.

5.2.2 Initial State Considerations

The message broker topology and the EAI implementation suggested here assume that the custom printing enterprise is currently using a P2P topology and tightly coupled systems. Additional work should be done to consider the proper EAI implementation for enterprises which have a different initial state.

5.2.3 Scalability

The research conducted here was based on four production facilities with four customers submitting orders. The scalability of this solution was not measured by creating additional plants and customers. A more thorough simulation could be conducted by running scenarios with more varying numbers of plants and customers.

5.3 Suggestions for Future Work

To improve the simulation, the file system delivery method could be replaced with web services to closer simulate a real world environment. Placing the customer, plant and hub console applications on separate computers for testing could also remove any performance impact they may inflict on each other.

Enhancing this simulation to allow multiple items per order would also provide more information on the impact of order decomposition in a message broker environment in comparison with the necessity to submit multiple orders in the P2P topology if a single plant is unable to produce all items requested.

Adding additional functionality to take into account inventory levels at the distributed plants and quantity of items requested would most likely also prove the message broker to be a better fit for a distributed printing environment.

Another aspect of the simulation which could be enhanced would involve making the hub more intelligent to allow it to route orders to production facilities based on their physical location to a customer to reduce shipping costs. Shipping costs could be compared between traditional P2P and the message broker implementation.

REFERENCES

- [1] A. Aarsten, D. Brudali, G. Menga, L. Mosconi, "Using Agent Technology in Virtual Factories," *Communications of the ACM, Special Issue on Agent Technology*, 1996.
- [2] K. Ake, J. Clemons, M. Cubine, *Information Technology for Manufacturing*. Boca Raton, FL: St. Lucie Press, 2004.
- [3] N. Aggarwal, D. Qizhi, E. A. Walden, "Do Markets Prefer Open or Proprietary Standards for XML Standardization? An Event Study." *International Journal of Electronic Commerce*. 11(1), pp. 117-136, 2006.
- [4] A. Asher, "Developing a B2B E-Commerce Implementation Framework: A Study of EDI Implementation for Procurement," *Information Systems Management*. 24, pp. 373-390.
- [5] C. Bass, J. M. Lee, "Building a Business Case for EAI," *EAI Journal*, pp. 18-20, 2002.
- [6] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, R. Shah, *Service Oriented Architecture Compass*. Upper Saddle River, NJ: IBM Press, 2005.
- [7] P. Chau, "Inhibitors to Edi Adoption in Small Businesses: An Empirical Investigation," *Journal of Electronic Commerce Research*. 2(2), pp. 78-88, 2001.
- [8] G. Coticchia, "Seven Steps to a Successful SOA Implementation [Electronic version]," *Business Integration Journal*, Sept/Oct 06, pp. 10-13, 2006.
- [9] cXML.org, "Commerce Xml Resources," [Online]. Available: <http://www.cXML.org>. [Accessed: March 2, 2008].
- [10] S. M. Davis, *Future Perfect*. Reading, MA: Addison-Wesley Publishing Inc, 1987.

- [11] A. J. Dietrich, S. Kirn, I. J. Timm, "Implications of Mass Customisation on Business Information Systems," *International Journal of Mass Customisation*. 1(2-3), pp. 218-236, 2006.
- [12] B. Elgin, "Google Buys Android for its Mobile Arsenal," [Online]. Available: http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm. [Accessed: November 4, 2008].
- [13] P. Fremantle, S. Weerwarana, R. Khalaf, "Enterprise Services: Examining the emerging field of Web Services and how it is integrated into existing enterprise infrastructures," *Communications of the ACM*. 45(10), pp. 77-82, 2002.
- [14] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA: Addison-Wesley Publishing Inc., 2003.
- [15] N. García-Sánchez, L. E. Pérez-Bernal, "Determination of critical success factors in implementing an ERP system: A field study in Mexican enterprises," *Information Technology for Development*, 13(3), pp. 293-309, 2007.
- [16] D. K. Gifford, N. Glasser, "Remote Pipes and Procedures for Efficient Distributed Communication," *ACM Transactions on Computer Systems*, 6(3), pp. 258-283, 1988.
- [17] T. Gulledge, "What is Integration?" *Industrial Management and Data Systems*, 06(1), pp. 5-20, 2006.
- [18] M. Henning, "The Rise and Fall of CORBA," *Communications of the ACM*, 51(8), pp. 53-57, 2008.
- [19] G. Hohpe, "Conversations between Loosely Coupled Services," [Online]. Available: <http://www.infoq.com/presentations/hohpe-soa-conversations>. [Accessed: October 27, 2008].

- [20] G. Hohpe, “*Hub and Spoke [or] Zen and the Art of Message Broker Maintenance*,” [Online]. Available http://www.enterpriseintegrationpatterns.com/ramblings/03_hubandspoke.html. [Accessed: February 2, 2008].
- [21] G. Hohpe, B. Woolf, *Enterprise Integration Patterns*. Boston, MA: Addison-Wesley, 2004.
- [22] D. Jefford, K. B. Smith, E. Fairweather, *Professional BizTalk Server 2006*, Indianapolis, IN: Wiley Publishing, Inc., 2007.
- [23] P. Johannesson, E. Perjons, “Design principles for process modeling in enterprise application integration,” *Information Systems*, 26 (3), 2001.
- [24] R. Kay, “QuickStudy: Service Oriented Architecture (SOA),” [Online]. Available: <http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=90530>. [Accessed: October 22, 2008].
- [25] K. Khoubati, M. Themistocleous, Z. Irani, “Investigating Enterprise Application Integration Benefits and Barriers in Healthcare Organizations: an Exploratory Case Study,” *International Journal of Electronic Healthcare*, 2(1), pp. 66-78, 2006.
- [26] W. Lam, “Investigating success factors in enterprise application integration: a case-driven analysis,” *European Journal of Information Systems*, 14(2), pp. 175, 2005.
- [27] J. Lee, K. Siau, S. Hong, “Enterprise Integration with ERP and EAI,” *Communications of the ACM*, 46(2), pp. 54-60, 2003.
- [28] D. Linthicum, *Enterprise Application Integration*. Reading, MA: Addison-Wesley, 2000.

- [29] R. Levine, "EAI Roads to Riches," [Online]. Available: <http://www.bijonline.com/index.cfm?section=article&aid=120>. [Accessed: February 2, 2008]
- [30] M. Marinov, "Using frames for knowledge representation in a CORBA-based distributed environment," *Knowledge-Based Systems*, 21(5), pp. 391-397, 2008.
- [31] R. O'Brient, "Integration Architecture Explained, [Online]. Available: <http://hubpages.com/hub/Integration-Architecture-Explained>. [Accessed: October 30, 2008]
- [32] S. Ortiz Jr., "Getting on Board the Enterprise Service Bus," *Computer*. 40(4), pp. 15-17, 2007.
- [33] A. Presley, "ERP Investment Analysis using the Strategic Alignment Model," *Management Research News*. 29(5), pp. 273.
- [34] M. Reardon, "Juniper goes Shopping," [Online]. Available: <http://news.zdnet.co.uk/communications/0,1000000085,39196473,00.htm>. [Accessed: November 4, 2008].
- [35] M. Reardon, "Cisco announces Start-Up Acquisition," [Online]. Available: <http://news.zdnet.co.uk/itmanagement/0,1000000308,39159105,00.htm>. [Accessed: November 4, 2008].
- [36] J. Ricker, D. Munro, D. Hopeman, "XML & EDI – Peaceful Co-Existence. XML Solutions Corp.," [Online]. Available <http://www.tdan.com/view-articles/4838/>. [Accessed: March 17, 2008].
- [37] M. T. Schmidt, B. Hutchinson, P. Lambros, R. Phippen, "The Enterprise Service Bus: Making service oriented architecture real," *IBM Systems Journal*. 44(4), pp. 781-797, 2005.

- [38] W. Shen, Q. Hao, S. Wang, Y. Li, H. Ghenniwa, "An agent-based service-oriented integration architecture for collaborative intelligent manufacturing," *Robotics and Computer-Integrated Manufacturing*, 23, pp. 315-325, 2007.
- [39] R. A. Sommer, T. R. Gullledge, D. Bailey, "The n-Tier Hub Technology," *SIGMOD Record*. 31(1), pp. 18-23, 2001.
- [40] M. Sumner, "Risk Factors in Enterprise-Wide/ERP Projects," *Journal of Information Technology*, 15(4), pp. 317-327, 2007.
- [41] Taylor Corporation, "Taylor Corporation Acquires PhotoCraft," [Online]. Available: <http://members.whattheythink.com/news/newslink.cfm?id=29023>. [Accessed: November 4, 2008].
- [42] E. Tilevich, Y. Smaragdakis, "NRMI: Natural and Efficient Middleware," *IEEE Transactions on Parallel and Distributed Systems*, 19(2), pp. 174-187, 2008.
- [43] C. Wang, L. Xu, X. Liu, X. Qin, "ERP research, development and implementation in China: an overview," *International Journal of Production Research*, 43 (18), pp. 3915-3932, 2005.
- [44] Weick, K. (2001). *Making Sense of the Organization*. Hoboken, NJ: Blackwell Publishing Ltd.
- [45] D. Weyns, K. Schelfhout, T. Holvoet, International Conference on Software Engineering: Proceedings of the 2005 workshop on Design and evolution of autonomic application software, 2005.
- [46] Wikipedia, "Electronic Data Interchange," [Online]. Available: http://en.wikipedia.org/wiki/Electronic_Data_Interchange. [Accessed: February 15, 2008].

- [47] Wikipedia, "Enterprise Application Integration," [Online]. Available:
http://en.wikipedia.org/wiki/Enterprise_application_integration. [Accessed:
October 01, 2008].
- [48] Wikipedia, "List of ERP Software Packages," [Online]. Available:
http://en.wikipedia.org/wiki/List_of_ERP_software_packages. [Accessed:
October 21, 2008].
- [49] Wikipedia, "Remote Procedure Call," [Online]. Available:
http://en.wikipedia.org/wiki/Remote_procedure_call. [Accessed: October 27,
2008].
- [50] Wikipedia, "Service Oriented Architecture," [Online]. Available:
http://en.wikipedia.org/wiki/Service-oriented_architecture. [Accessed: November
8, 2008].
- [51] J. Wetherill, "Messaging Systems and the Java Message Service (JMS)," [Online].
Available:
<http://java.sun.com/developer/technicalArticles/Networking/messaging/>.
[Accessed: October 30, 2008].
- [52] I. Zahir, M. Themistocleous, P. E. D. Love, "The impact of enterprise application
integration on information system lifecycles," *Information & Management*, 2002.

APPENDIX A

GLOSSARY OF ACRONYMS

CORBA – Common Object Request Broker Architecture

CSV – Comma Separated Values

DCOM – Distributed Component Object Model

EAI – Enterprise Application Integration

EDI – Electronic Data Interchange

ERP – Enterprise Resource Planning

ESB – Enterprise Service Bus

MAS – Multi Agent Systems

OLAP – Online Analytical Processing

P2P – Point To Point

SOA – Service Oriented Architecture

SOAP – Simple Object Access Protocol

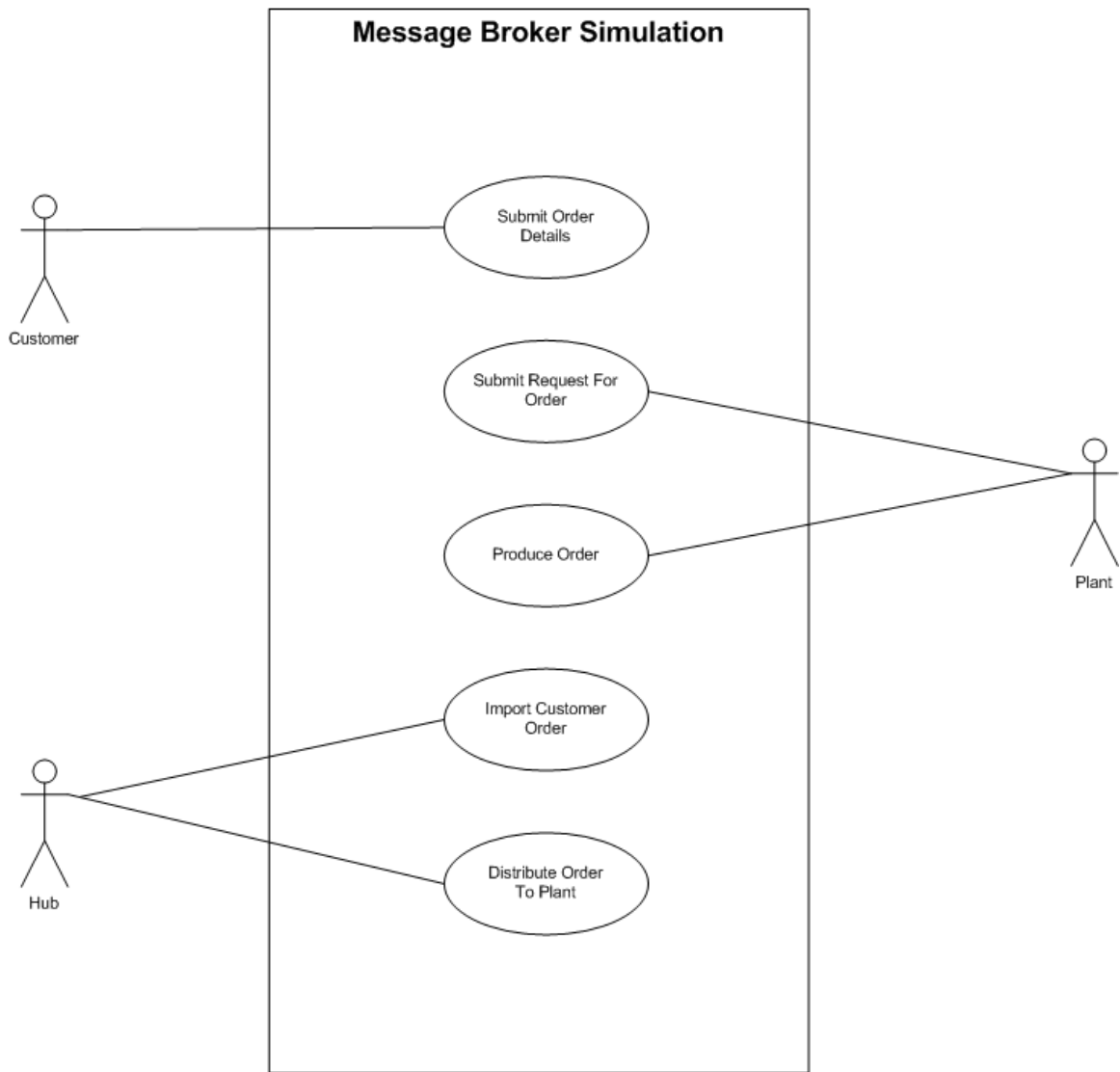
VAN – Value Added Network

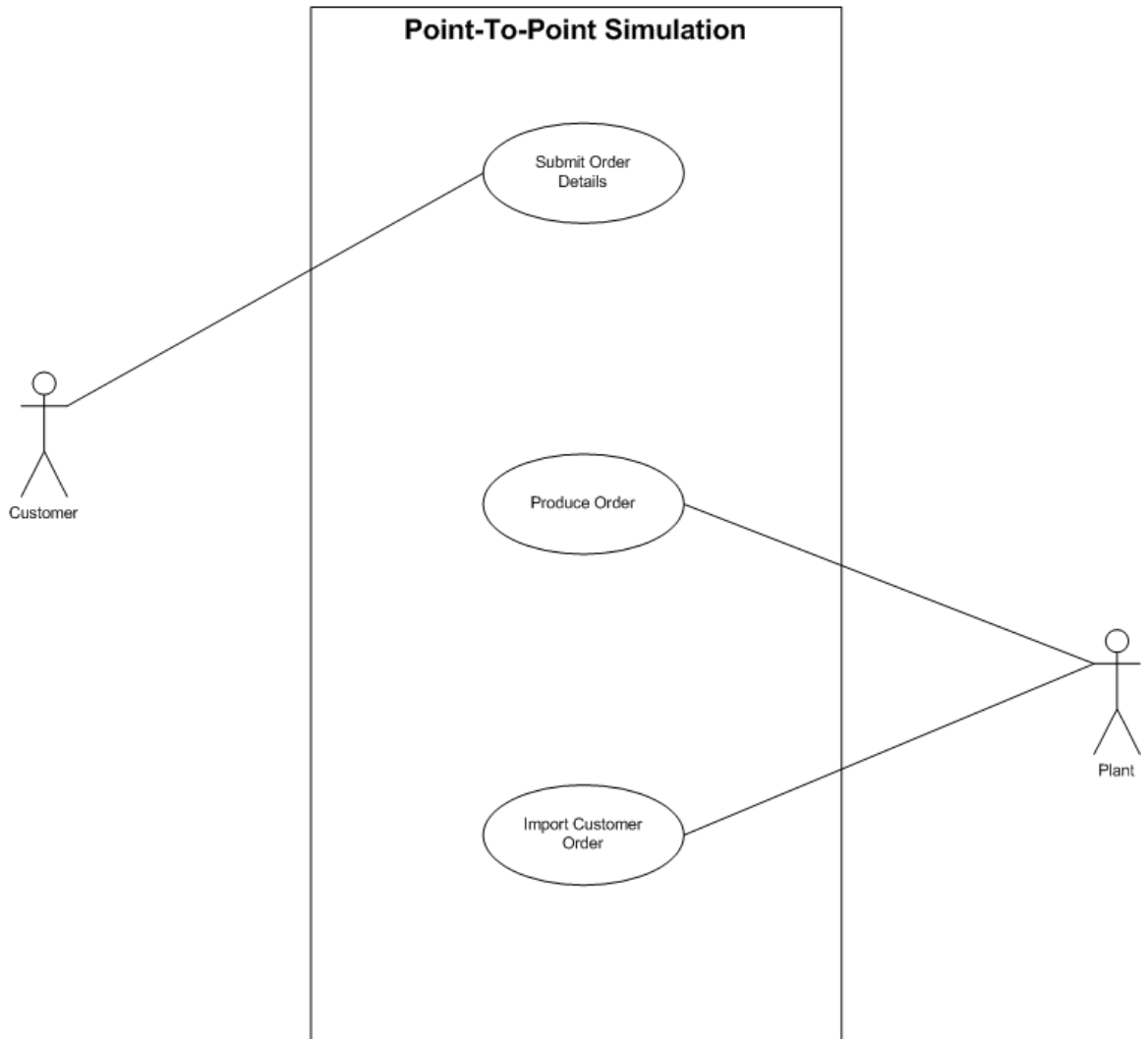
XSL or XSLT – eXtensible Style Sheet (Transformations)

XML – eXtensible Markup Language

APPENDIX B

USE CASE MODELS





APPENDIX C

SIMULATION CONFIGURATIONS

Hub Simulation – Normal Production

Plant,H,Oregon,111,A|B|C,5,50,%Base Directory%\Production\Hub\NormalRoute\,%Base Directory%\RequestsForWork
 Plant,H,Minnesota,222,A|B|C,5,50,%Base Directory%\Production\Hub\NormalRoute\,%Base Directory%\RequestsForWork
 Plant,H,Florida,333,A|B|C,5,50,%Base Directory%\Production\Hub\NormalRoute\,%Base Directory%\RequestsForWork
 Plant,H,Texas,444,A|B|C,5,50,%Base Directory%\Production\Hub\NormalRoute\,%Base Directory%\RequestsForWork
 Hub,1,,,,,,
 Customer,Customer1,123,A|B|C,10,1000,%Base Directory%\CustomerOrders,,
 Customer,Customer2,456,A|B|C,10,1000,%Base Directory%\CustomerOrders,,
 Customer,Customer3,789,A|B|C,10,1000,%Base Directory%\CustomerOrders,,
 Customer,Customer4,101,A|B|C,10,1000,%Base Directory%\CustomerOrders,,

Hub Simulation – Busy Production

Plant,H,Oregon,111,A|B|C,5,50,%Base Directory%\Production\Hub\BusyRoute\,%Base Directory%\RequestsForWork
 Plant,H,Minnesota,222,A|B|C,5,50,%Base Directory%\Production\Hub\BusyRoute\,%Base Directory%\RequestsForWork
 Plant,H,Florida,333,A|B|C,5,50,%Base Directory%\Production\Hub\BusyRoute\,%Base Directory%\RequestsForWork
 Plant,H,Texas,444,A|B|C,5,50,%Base Directory%\Production\Hub\BusyRoute\,%Base Directory%\RequestsForWork
 Hub,1,,,,,,
 Customer,Customer1,123,A|B|C,2,1250,%Base Directory%\CustomerOrders,,
 Customer,Customer2,456,A|B|C,2,1250,%Base Directory%\CustomerOrders,,
 Customer,Customer3,789,A|B|C,2,1250,%Base Directory%\CustomerOrders,,
 Customer,Customer4,101,A|B|C,2,1250,%Base Directory%\CustomerOrders,,

Hub Simulation – Limited Production

Plant,H,Oregon,111,A|B|C,1,50,% Base Directory%\Production\Hub\LimitedRoute\,% Base Directory%\RequestsForWork
 Plant,H,Minnesota,222,A|B|C,5,50,% Base Directory%\Production\Hub\LimitedRoute\,% Base Directory%\RequestsForWork
 Plant,H,Florida,333,A|B|C,5,50,% Base Directory%\Production\Hub\LimitedRoute\,% Base Directory%\RequestsForWork
 Plant,H,Texas,444,A|B|C,5,50,% Base Directory%\Production\Hub\LimitedRoute\,% Base Directory%\RequestsForWork
 Hub,1,,,,,,
 Customer,Customer1,123,A|B|C,10,1000,% Base Directory%\CustomerOrders,,
 Customer,Customer2,456,A|B|C,10,1000,% Base Directory%\CustomerOrders,,
 Customer,Customer3,789,A|B|C,10,1000,% Base Directory%\CustomerOrders,,
 Customer,Customer4,101,A|B|C,10,1000,% Base Directory%\CustomerOrders,,

Traditional Simulation – Normal Production

Plant,T,Oregon,111,A|B|C,5,50,% Base Directory%\Production\Traditional\Normal,unused
 Plant,T,Minnesota,222,A|B|C,5,50,% Base Directory%\Production\Traditional\Normal,unused
 Plant,T,Florida,333,A|B|C,5,50,% Base Directory%\Production\Traditional\Normal,unused
 Plant,T,Texas,444,A|B|C,5,50,% Base Directory%\Production\Traditional\Normal,unused
 Customer,Customer1,123,A|B|C,10,1000,% Base Directory%\Production\Traditional\Normal\Oregon\In,,
 Customer,Customer2,456,A|B|C,10,1000,% Base Directory%\Production\Traditional\Normal\Minnesota\In,,
 Customer,Customer3,789,A|B|C,10,1000,% Base Directory%\Production\Traditional\Normal\Florida\In,,
 Customer,Customer4,101,A|B|C,10,1000,% Base Directory%\Production\Traditional\Normal\Texas\In,,

Traditional Simulation – Busy Production

Plant,T,Oregon,111,A|B|C,5,50,% Base Directory%\Production\Traditional\Busy,unused
 Plant,T,Minnesota,222,A|B|C,5,50,% Base Directory%\Production\Traditional\Busy,unused
 Plant,T,Florida,333,A|B|C,5,50,% Base Directory%\Production\Traditional\Busy,unused
 Plant,T,Texas,444,A|B|C,5,50,% Base Directory%\Production\Traditional\Busy,unused
 Customer,Customer1,123,A|B|C,2,1250,% Base Directory%\Production\Traditional\Busy\Oregon\In,,
 Customer,Customer2,456,A|B|C,2,1250,% Base Directory%\Production\Traditional\Busy\Minnesota\In,,

Customer, Customer3, 789, A|B|C, 2, 1250, % Base Directory% \Production\Traditional\Busy\Florida\In,,
 Customer, Customer4, 101, A|B|C, 2, 1250, % Base Directory% \Production\Traditional\Busy\Texas\In,,

Traditional Simulation – Limited Production

Plant, T, Oregon, 111, A|B|C, 1, 50, % Base Directory% \Production\Traditional\Limited\, unused
 Plant, T, Minnesota, 222, A|B|C, 5, 50, % Base Directory% \Production\Traditional\Limited\, unused
 Plant, T, Florida, 333, A|B|C, 5, 50, % Base Directory% \Production\Traditional\Limited\, unused
 Plant, T, Texas, 444, A|B|C, 5, 50, % Base Directory% \Production\Traditional\Limited\, unused
 Customer, Customer1, 123, A|B|C, 10, 1000, % Base Directory% \Production\Traditional\Limited\Oregon\In,,
 Customer, Customer2, 456, A|B|C, 10, 1000, % Base Directory% \Production\Traditional\Limited\Minnesota\In,,
 Customer, Customer3, 789, A|B|C, 10, 1000, % Base Directory% \Production\Traditional\Limited\Florida\In,,
 Customer, Customer4, 101, A|B|C, 10, 1000, % Base Directory% \Production\Traditional\Limited\Texas\In,,

APPENDIX D

CODE LISTING

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;

namespace MNSU.Mankato.CS.APP.BPMarch.Customer
{
    /// <summary>
    /// Class which implements a customer for an EAI simulation
    /// </summary>
    public class Customer
    {
        private string name;
        private int customerId;
        private string items;
        private int frequency;
        private int orderCount;
        private int currentOrderId;
        private string outputPath;
        private int timeModifier;
        private bool displayOutput;
        private Random randomNum;

        /// <summary>
        /// Initializes a new instance of the <see cref="Customer"/> class.
        /// </summary>
        /// <param name="name">The name.</param>
        /// <param name="customerId">The customer id.</param>
        /// <param name="items">The items.</param>
        /// <param name="frequency">The frequency.</param>
        /// <param name="orderCount">The order count.</param>
        /// <param name="outputPath">The output path.</param>
        /// <param name="timeModifier">The time modifier.</param>
        /// <param name="displayOutput">if set to <c>true</c> [display output].</param>
        public Customer(string name, int customerId, string items,
            int frequency, int orderCount, string outputPath, int timeModifier,
            bool displayOutput)
        {
            this.name = name;
            this.customerId = customerId;
            this.items = items;
            this.frequency = frequency;
            this.orderCount = orderCount;
            this.outputPath = outputPath;
            this.timeModifier = timeModifier;
            this.displayOutput = displayOutput;
            currentOrderId = 0;
            randomNum = new Random();
        }

        /// <summary>
        /// The entry point for running running a customer.
        /// </summary>
        /// <param name="args">The args.</param>
        static void Main(string[] args)
        {
            string name = args[0];
            int customerId = Convert.ToInt32(args[1]);
            string items = args[2];
            int frequency = Convert.ToInt32(args[3]);
            int orderCount = Convert.ToInt32(args[4]);
            string outputPath = args[5];
            int timeModifier = Convert.ToInt32(args[6]);
            bool displayOutput = Convert.ToBoolean(args[7]);

            Customer customer = new Customer(
                name, customerId, items, frequency, orderCount,

```

```

        outputPath, timeModifier, displayOutput);

    customer.Run();
}

/// <summary>
/// Runs this instance.
/// </summary>
public void Run()
{
    while (currentOrderId < orderCount)
    {
        currentOrderId = currentOrderId + 1;
        Thread.Sleep(frequency * timeModifier);
        CreateOrder();
        LogProgress();
    }
}

/// <summary>
/// Writes progress to the screen for the customer.
/// </summary>
private void LogProgress()
{
    StringBuilder sb = new StringBuilder();
    sb.Append(name);
    sb.Append(" | ");
    sb.Append(customerId);
    sb.Append(" | ");
    sb.Append("Order " + currentOrderId + " Created");
    if(displayOutput)
        Console.WriteLine(sb.ToString());
}

/// <summary>
/// Creates an order.
/// </summary>
private void CreateOrder()
{
    CreateFolder();
    StreamWriter writer = File.CreateText(OutputFileName());
    string order = PrepareOrder();
    writer.Write(order);
    writer.Flush();
    writer.Close();
    writer.Dispose();
}

/// <summary>
/// Checks for existence of the output path folder and creates it if it
/// does not exist.
/// </summary>
private void CreateFolder()
{
    DirectoryInfo dir = new DirectoryInfo(outputPath);
    if (!dir.Exists)
    {
        dir.Create();
    }
}

/// <summary>
/// Creates a unique file name for an order
/// </summary>
/// <returns></returns>
private string OutputFileName()
{
    return Path.Combine(outputPath, System.Guid.NewGuid().ToString() + ".txt");
}

/// <summary>
/// Prepares order information for an order.
/// </summary>
/// <returns></returns>
private string PrepareOrder()
{
    //CustomerId,OrderId,Item,DateTimeCreated
    //Example: 12,1,WIDGET,10/24/2008
}

```

```

        StringBuilder sb = new StringBuilder();
        sb.Append(customerId);
        sb.Append(",");
        sb.Append(currentOrderId);
        sb.Append(",");
        sb.Append(GetItem());
        sb.Append(",");
        sb.Append(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss:fff"));
        return sb.ToString();
    }

    /// <summary>
    /// Gets a random item from the list of items which this customer may order.
    /// </summary>
    /// <returns>String representing an item id.</returns>
    private string GetItem()
    {
        string[] itemsArray = items.Split('|');
        int index = randomNum.Next(0, itemsArray.Length);
        return itemsArray[index];
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;

namespace MNSU.Mankato.CS.APP.BPMarch.Plant
{
    /// <summary>
    /// Class which implements plant functionality for simulating EAI performance.
    /// </summary>
    public class Plant
    {
        private string plantType;
        private string name;
        private int plantId;
        private string items;
        private int openRequests;
        private int capacity;
        private int workInProgress;
        private int speed;
        private string baseProductionPath;
        private string incomingOrderPath;
        private string outgoingOrderPath;
        private string requestWorkPath;
        private int timeModifier;
        private bool displayOutput;

        /// <summary>
        /// Initializes a new instance of the <see cref="Plant"/> class.
        /// </summary>
        /// <param name="plantType">Type of the plant.</param>
        /// <param name="name">The name.</param>
        /// <param name="plantId">The plant id.</param>
        /// <param name="items">The items.</param>
        /// <param name="capacity">The capacity.</param>
        /// <param name="speed">The speed.</param>
        /// <param name="baseProductionPath">The base production path.</param>
        /// <param name="requestWorkPath">The request work path.</param>
        /// <param name="timeModifier">The time modifier.</param>
        /// <param name="displayOutput">if set to <c>true</c> [display output].</param>
        public Plant(string plantType, string name, int plantId, string items,
            int capacity, int speed, string baseProductionPath, string requestWorkPath,
            int timeModifier, bool displayOutput)
        {
            this.plantType = plantType;
            this.name = name;
            this.plantId = plantId;
            this.items = items;
            this.capacity = capacity;
            this.speed = speed;
            this.baseProductionPath = baseProductionPath;
            this.incomingOrderPath = baseProductionPath + "\\\" + name + "\\In";
            this.outgoingOrderPath = baseProductionPath + "\\\" + name + "\\Out"; ;
        }
    }
}

```

```

        this.requestWorkPath = requestWorkPath;
        this.timeModifier = timeModifier;
        this.displayOutput = displayOutput;
        this.workInProgress = 0;
        this.openRequests = 0;

        if (!Directory.Exists(outgoingOrderPath))
        {
            Directory.CreateDirectory(outgoingOrderPath);
        }

        if (!Directory.Exists(incomingOrderPath))
        {
            Directory.CreateDirectory(incomingOrderPath);
        }
    }

    /// <summary>
    /// Creates and starts a thread to simulate production.
    /// </summary>
    public void Run()
    {
        ThreadStart setupProdStart = new ThreadStart(ManagePlantProduction);
        Thread setupProd = new Thread(setupProdStart);
        setupProd.Start();
    }

    /// <summary>
    /// Creates a work request to match the capacity of the plant.
    /// </summary>
    private void PrimeProductionFloor()
    {
        for (int i = 0; i < capacity; i++)
        {
            RequestWork();
        }
    }

    /// <summary>
    /// Creates a file to request work from the hub.
    /// </summary>
    private void RequestWork()
    {
        openRequests = openRequests + 1;
        StreamWriter writer = File.CreateText(CreateFileName());
        writer.Write(items + "," + incomingOrderPath);
        writer.Flush();
        writer.Close();
        writer.Dispose();
        if(displayOutput)
            Console.WriteLine(name + " Requested Work.");
    }

    /// <summary>
    /// Creates a random file name.
    /// </summary>
    /// <returns></returns>
    private string CreateFileName()
    {
        return Path.Combine(requestWorkPath,
            System.Guid.NewGuid().ToString() + ".txt");
    }

    /// <summary>
    /// Creates worker threads any time work is not at full capacity if there
    /// is work to be done.
    /// </summary>
    private void ManagePlantProduction()
    {
        if (this.plantType.Equals("H"))
        {
            PrimeProductionFloor();
        }
        //Permanently loop to try to
        while (true)
        {
            while (workInProgress <= capacity)
            {

```

```

        ThreadStart workerStart = new ThreadStart(ProduceOrder);
        Thread worker = new Thread(workerStart);
        worker.Start();
        //Short delay between starting production
        Thread.Sleep(100);
    }
    //Continually check current work in progress after a delay
    Thread.Sleep(50 * timeModifier);
}
}

/// <summary>
/// Simulates the work being done in a production line.
/// </summary>
private void ProduceOrder()
{
    if (workInProgress < capacity)
    {
        //Get the list of files ready to be worked on
        string[] workInQueue = Directory.GetFiles(incomingOrderPath, "*.txt");

        // Now read the creation time for each file
        DateTime[] creationTimes = new DateTime[workInQueue.Length];
        for (int i = 0; i < workInQueue.Length; i++)
            creationTimes[i] = new FileInfo(workInQueue[i]).CreationTime;

        // sort it
        Array.Sort(creationTimes, workInQueue);

        //If there is work to be done
        if (workInQueue != null && workInQueue.Length > 0)
        {
            //Get the first file to work on
            string order = workInQueue[0];
            string workingFile = Path.ChangeExtension(order, ".wrk");
            try
            {
                File.Move(order, workingFile);
            }
            catch { return; }

            //An open request has transitioned to work in progress
            openRequests = openRequests - 1;
            workInProgress = workInProgress + 1;

            if (this.plantType.Equals("H"))
            {
                //Request another piece of work so it can be
                //delivered while the current piece of work is in progress.
                ThreadStart requestWork = new ThreadStart(RequestWork);
                Thread worker = new Thread(requestWork);
                worker.Start();
            }

            if (displayOutput)
                Console.WriteLine(name + " Work Starting Capacity: " +
                    capacity + " WIP: " + workInProgress);

            //Simulate delay required during production
            Thread.Sleep(speed * timeModifier);

            //Assign a new path for the order now that it is complete
            string completedOrder = Path.Combine(outgoingOrderPath,
                Path.ChangeExtension(Path.GetFileName(order), ".txt"));

            if (File.Exists(workingFile))
            {
                //Work has completed, move the file to the output folder.
                File.Move(workingFile, completedOrder);
            }

            //Decrease the work in progress
            workInProgress = workInProgress - 1;

            //Update the order to reflect it has been completed
            SetOrderCompletedTimestamp(completedOrder, DateTime.Now);
        }
    }
    else

```

```

        {
            if (displayOutput)
                Console.WriteLine("No work in queue.");
        }
    }

    /// <summary>
    /// Appends a timestamp to an order file to identify when work has completed
    /// for an order
    /// </summary>
    /// <param name="filename">The filename.</param>
    /// <param name="timestamp">The timestamp.</param>
    private void SetOrderCompletedTimestamp(string filename, DateTime timestamp)
    {
        try
        {
            FileStream fs = File.OpenWrite(filename);
            StreamWriter writer = new StreamWriter(fs);
            writer.BaseStream.Seek(0, SeekOrigin.End);
            writer.Write(", " + timestamp.ToString("yyyy-MM-dd HH:mm:ss:fff"));
            writer.Flush();
            writer.Close();
            writer.Dispose();
        }
        catch
        {
            Thread.Sleep(2000);
            SetOrderCompletedTimestamp(filename, timestamp);
        }
    }

    /// <summary>
    /// The main entry point to start a plant's production simulation
    /// </summary>
    /// <param name="args">The args.</param>
    static void Main(string[] args)
    {
        string plantType = args[0];
        string name = args[1];
        int plantId = Convert.ToInt32(args[2]);
        string items = args[3];
        int capacity = Convert.ToInt32(args[4]);
        int speed = Convert.ToInt32(args[5]);
        string baseProdPath = args[6];
        string requestWorkPath = args[7];
        int timeModifier = Convert.ToInt32(args[8]);
        bool displayOutput = Convert.ToBoolean(args[9]);

        Plant plant = new Plant(plantType, name, plantId, items, capacity, speed,
            baseProdPath, requestWorkPath, timeModifier, displayOutput);

        plant.Run();
    }
}

using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;

using MNSU.Mankato.CS.APP.BPMarch.BO;

namespace MNSU.Mankato.CS.APP.BPMarch.Hub
{
    /// <summary>
    /// Class which implements hub functionality in a Message Broker topology.
    /// </summary>
    public class Hub : IDisposable
    {
        private int pollingFrequency;
        private int delayMultiplier;
        private Thread importThread;
    }
}

```

```

private Thread distributeThread;
private bool displayOutput;
private Random randNumber;

/// <summary>
/// Initializes a new instance of the <see cref="Hub"/> class.
/// </summary>
/// <param name="frequency">The frequency.</param>
/// <param name="delayMultiplier">The delay multiplier.</param>
/// <param name="displayOutput">if set to <c>true</c> [display output].</param>
public Hub(int frequency, int delayMultiplier, bool displayOutput)
{
    this.delayMultiplier = delayMultiplier;
    this.pollingFrequency = frequency;
    this.displayOutput = displayOutput;
    randNumber = new Random();
}

/// <summary>
/// Creates and starts threads for importing orders and distributing work
/// </summary>
public void Run()
{
    ThreadStart import = new ThreadStart(ImportOrders);
    importThread = new Thread(import);
    importThread.Start();

    ThreadStart distribute = new ThreadStart(DistributeWork);
    distributeThread = new Thread(distribute);
    distributeThread.Start();

    if (displayOutput)
        Console.WriteLine("Hub started. Press enter to end.");
    Console.ReadLine();
    this.Stop();
}

/// <summary>
/// Stops the import and distribute work threads.
/// </summary>
public void Stop()
{
    importThread.Abort();
    distributeThread.Abort();
}

/// <summary>
/// Imports the orders from the hubs inbox.
/// </summary>
private void ImportOrders()
{
    int infiniteLoopCounter = 0;
    string ordersInQueuePath = @"F:\temp\school\EAI\OrdersInQueue";

    while (true)
    {
        string[] orders = Directory.GetFiles(@"F:\temp\school\EAI\CustomerOrders");
        if (orders.Length == 0)
        {
            infiniteLoopCounter = infiniteLoopCounter + 1;
            Console.WriteLine("Infinite loop counter: " + infiniteLoopCounter);
            Thread.Sleep(1000);
        }
        if (infiniteLoopCounter >= 60)
        {
            break;
        }
        for (int i = 0; i < orders.Length; i++)
        {
            infiniteLoopCounter = 0;
            string order = orders[i];
            string finalOrderPath = ordersInQueuePath;
            bool tryToMove = true;

            try
            {
                string itemType = GetOrderInfo(order, Order.ITEM_TYPE_INDEX);

```

```

        if (itemType != null)
        {
            finalOrderPath = Path.Combine(ordersInQueuePath, itemType);
            if (!Directory.Exists(finalOrderPath))
            {
                Directory.CreateDirectory(finalOrderPath);
            }
        }
    }
    catch
    {
        tryToMove = false;
    }

    if (tryToMove)
    {
        try
        {
            //The file may be in use so
            //take no action if it is still being written.
            File.Move(order,
                Path.Combine(finalOrderPath, Path.GetFileName(order)));
        }
        catch
        { }
    }
}
Thread.Sleep(pollingFrequency & delayMultiplier);
}
}

/// <summary>
/// Gets information from a csv file based on the supplied index.
/// </summary>
/// <param name="order">The file to receive information from.</param>
/// <param name="index">The index of information being sought.</param>
/// <returns>string of information at the given index</returns>
private string GetOrderInfo(string file, int index)
{
    string fileText = File.ReadAllText(file);
    string[] textArray = fileText.Split(',');
    return textArray[index];
}

/// <summary>
/// Distributes work to plants.
/// </summary>
private void DistributeWork()
{
    string requestPath = @"F:\temp\school\EAI\RequestsForWork";

    while (true)
    {
        if (Directory.Exists(requestPath))
        {
            string[] requests = Directory.GetFiles(requestPath);

            // Now read the creation time for each file
            DateTime[] creationTimes = new DateTime[requests.Length];
            for (int i = 0; i < requests.Length; i++)
                creationTimes[i] = new FileInfo(requests[i]).CreationTime;

            // sort it
            Array.Sort(creationTimes, requests);

            //Loop over all current requests for work
            for (int i = 0; i < requests.Length; i++)
            {
                //The current request for work
                string requestFile = requests[i];
                //Work that can be produced by the requestor

                try
                {
                    string itemTypesRequested = GetOrderInfo(requestFile,
                        RequestForWork.ITEM_TYPE_INDEX);
                }
            }
        }
    }
}

```

```

        if (itemTypesRequested != null)
        {
            //Retrieve an order that can be produced by requestor
            string distributionOrder =
RetrieveSuitableWorkOrder(itemTypesRequested);
            //If an order could be found that matches the plants abilities
            if (distributionOrder != null)
            {
                //Get the directory to submit the order to
                string submitToDirectory = GetOrderInfo(requestFile,
RequestForWork.SUBMIT_TO_DIRECTORY_INDEX);

                if (submitToDirectory != null)
                {
                    //Submission file path for current order
                    string fullSubmitToPath =
Path.Combine(submitToDirectory,
Path.GetFileName(distributionOrder));

                    try
                    {
                        if (!Directory.Exists(submitToDirectory))
                        {
                            Directory.CreateDirectory(submitToDirectory);
                        }
                        File.Move(distributionOrder, fullSubmitToPath);
                        File.Delete(requestFile);
                    }
                    catch
                    {
                    }
                }
            }
        }
    }
}

Thread.Sleep(pollingFrequency * delayMultiplier);
}
}

/// <summary>
/// Retrieves a suitable work order based on the supplied item type.
/// </summary>
/// <param name="itemTypeString">The item type string.</param>
/// <returns>Path to an order file.</returns>
private string RetrieveSuitableWorkOrder(string itemTypeString)
{
    string openOrderPath = @"F:\temp\school\EAI\OrdersInQueue";
    string[] itemTypes = itemTypeString.Split('|');
    List<string> itemTypeList = new List<string>();
    for (int i = 0; i < itemTypes.Length; i++)
    {
        itemTypeList.Add(itemTypes[i]);
    }

    while (itemTypeList.Count > 0)
    {
        int index = randomNumber.Next(0, itemTypeList.Count);
        //randomly check for an order that matches the type that can be
        //produced by the plant and return the FileSystemInfo for that order
        string currentPath = Path.Combine(openOrderPath, itemTypeList[index]);

        if (Directory.Exists(currentPath))
        {
            string[] fileNames = Directory.GetFiles(currentPath);

            // Now read the creation time for each file
            DateTime[] creationTimes = new DateTime[fileNames.Length];
            for (int i = 0; i < fileNames.Length; i++)
                creationTimes[i] = new FileInfo(fileNames[i]).CreationTime;

            // sort it
            Array.Sort(creationTimes, fileNames);

            if (fileNames != null && fileNames.Length > 0)
            {

```

```

        return fileNames[0];
    }
    itemTypeList.RemoveAt(index);
}
return null;
}

/// <summary>
/// Execution entry point for the hub application.
/// </summary>
/// <param name="args">The args.</param>
static void Main(string[] args)
{
    int pollingInterval = Convert.ToInt32(args[0]);
    int timeModifier = Convert.ToInt32(args[1]);
    bool displayOutput = Convert.ToBoolean(args[2]);
    Hub hub = new Hub(pollingInterval, timeModifier, displayOutput);
    hub.Run();
}

/// <summary>
/// Performs application-defined tasks associated with
/// freeing, releasing, or resetting unmanaged resources.
/// </summary>
public void Dispose()
{
    try
    {
        this.Stop();
    }
    catch { }
}
}

namespace MNSU.Mankato.CS.APP.BPMarch.BO
{
    /// <summary>
    /// A business object which corresponds to information in an order.
    /// </summary>
    public class Order
    {
        public static int CUST_NUM_INDEX = 0;
        public static int ORDER_ID_INDEX = 1;
        public static int ITEM_TYPE_INDEX = 2;
        public static int DATE_CREATED_INDEX = 3;
        public static int DATE_PRODUCED_INDEX = 4;
    }
}

namespace MNSU.Mankato.CS.APP.BPMarch.BO
{
    /// <summary>
    /// A business object which corresponds to information in a request for work
    /// </summary>
    public class RequestForWork
    {
        public static int ITEM_TYPE_INDEX = 0;
        public static int SUBMIT_TO_DIRECTORY_INDEX = 1;
    }
}

using System;
using System.Collections.Generic;
using System.Configuration;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace MNSU.Mankato.CS.APP.BPMarch.Simulator
{

```

```

public partial class SimulatorForm : Form
{
    private int speedModifier = 100;

    /// <summary>
    /// Initializes a new instance of the <see cref="SimulatorForm"/> class.
    /// </summary>
    public SimulatorForm()
    {
        InitializeComponent();
        PopulateSimulationConfigurations();
    }

    /// <summary>
    /// Handles the Click event of the btnTestCustomer control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="System.EventArgs"/>
    /// instance containing the event data.</param>
    private void btnStartSimulation_Click(object sender, EventArgs e)
    {
        txtOutput.Text = "Simulation starting...";
        SetSpeedModifier();
        ClearOpenWorkRequests();

        FileStream fs = new FileStream(
            comboBox1.SelectedValue.ToString(),
            FileMode.Open, FileAccess.Read);
        string line;
        StreamReader reader = new StreamReader(fs);
        while((line = reader.ReadLine()) != null)
        {
            string[] parameters = line.Split(',');
            if (parameters[0].ToUpper().Equals("HUB"))
            {
                Process.Start(
                    ConfigurationSettings.AppSettings["HUB.EXE"],
                    parameters[1] + " " + //Polling Frequency
                    speedModifier + " " +
                    chkDisplayOutput.Checked); //time modifier in milliseconds

                txtOutput.Text = txtOutput.Text +
                    Environment.NewLine + "Hub Created.";
            }
            else if(parameters[0].ToUpper().Equals("CUSTOMER"))
            {
                Process.Start(
                    ConfigurationSettings.AppSettings["CUSTOMER.EXE"],
                    parameters[1] + " " + //Customer Name
                    parameters[2] + " " + //Customer Id
                    parameters[3] + " " + //Customer Products
                    parameters[4] + " " + //Customer Order Frequency
                    parameters[5] + " " + //Number of customer orders
                    parameters[6] + " " + //Path of customer's orders
                    speedModifier + " " +
                    chkDisplayOutput.Checked);

                txtOutput.Text = txtOutput.Text + Environment.NewLine
                    + "Customer Created: " + parameters[1] + ".";
            }
            else if(parameters[0].ToUpper().Equals("PLANT"))
            {
                Process.Start(
                    ConfigurationSettings.AppSettings["PLANT.EXE"],
                    parameters[1] + " " + //Plant Type (Hub or Traditional)
                    parameters[2] + " " + //Plant Name
                    parameters[3] + " " + //Plant Id
                    parameters[4] + " " + //Items
                    parameters[5] + " " + //Capacity
                    parameters[6] + " " + //Speed
                    parameters[7] + " " + //Production base path
                    parameters[8] + " " + //Requests for open work
                    speedModifier + " " +
                    chkDisplayOutput.Checked);

                txtOutput.Text = txtOutput.Text + Environment.NewLine
                    + "Plant Created: " + parameters[2] + ".";
            }
        }
    }
}

```

```

    }
    reader.Close();
    reader.Dispose();
    fs.Close();
    fs.Dispose();
}

/// <summary>
/// Populates the simulation configurations.
/// </summary>
private void PopulateSimulationConfigurations()
{
    DirectoryInfo dir = new DirectoryInfo(
        ConfigurationSettings.AppSettings["CONFIGURATIONS"]);

    FileInfo[] configurations = dir.GetFiles();
    comboBox1.DataSource = configurations;
}

/// <summary>
/// Sets the speed modifier.
/// </summary>
private void SetSpeedModifier()
{
    try
    {
        speedModifier = Convert.ToInt32(txtSpeedModifier.Text);
    }
    catch { }
}

/// <summary>
/// Clears the open work requests.
/// </summary>
private static void ClearOpenWorkRequests()
{
    string dirPath = ConfigurationSettings.AppSettings["REQUESTS_FOR_WORK"];
    try
    {
        if (Directory.Exists(dirPath))
        {
            Directory.Delete(dirPath);
        }
        Directory.CreateDirectory(dirPath);
    }
    catch { }
}

/// <summary>
/// Handles the Click event of the btnReset control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="e">The <see cref="System.EventArgs"/>
/// instance containing the event data.</param>
private void btnReset_Click(object sender, EventArgs e)
{
    string reqForWorkDir = ConfigurationSettings.AppSettings["REQUESTS_FOR_WORK"];
    string productionDir = ConfigurationSettings.AppSettings["PRODUCTION"];
    string ordersInQDir = ConfigurationSettings.AppSettings["ORDERS_IN_QUEUE"];
    string custOrdersDir = ConfigurationSettings.AppSettings["CUSTOMER_ORDERS"];

    DeleteDirectory(reqForWorkDir);
    DeleteDirectory(productionDir);
    DeleteDirectory(ordersInQDir);
    DeleteDirectory(custOrdersDir);

    Directory.CreateDirectory(productionDir);
    Directory.CreateDirectory(reqForWorkDir);
    Directory.CreateDirectory(ordersInQDir);
    Directory.CreateDirectory(custOrdersDir);

    txtOutput.Text = "Simulation Reset.";
}

/// <summary>
/// Deletes the directory.
/// </summary>
/// <param name="path">The path.</param>

```

```

private void DeleteDirectory(string path)
{
    if (Directory.Exists(path))
    {
        try
        {
            Directory.Delete(path, true);
        }
        catch { }
    }
}

/// <summary>
/// Prepares the header message.
/// </summary>
/// <returns></returns>
private string PrepareHeaderMessage()
{
    return "Order Count,"
        + "Avg Duration in milliseconds,"
        + "Total Duration,"
        + "First Order Created,"
        + "Last Order Produced, Longest Duration" + Environment.NewLine;
}

/// <summary>
/// Prepares the result message.
/// </summary>
/// <param name="orderCount">The order count.</param>
/// <param name="duration">The duration.</param>
/// <param name="startTime">The start time.</param>
/// <param name="stopTime">The stop time.</param>
/// <param name="longestDuration">Duration of the longest.</param>
/// <returns></returns>
private string PrepareResultMessage(
    int orderCount, double duration, DateTime startTime,
    DateTime stopTime, double longestDuration)
{
    return orderCount + ","
        + (duration / orderCount) + ","
        + duration + ","
        + startTime.ToString("yyyy-MM-dd HH:mm:ss:fff") + ","
        + stopTime.ToString("yyyy-MM-dd HH:mm:ss:fff") + ","
        + longestDuration;
}

/// <summary>
/// Gets the production times.
/// </summary>
/// <param name="orders">The orders.</param>
/// <param name="duration">The duration.</param>
/// <param name="startTime">The start time.</param>
/// <param name="stopTime">The stop time.</param>
/// <param name="longestDuration">Duration of the longest.</param>
private void GetProductionTimes(FileInfo[] orders, out double duration,
    out DateTime startTime, out DateTime stopTime, out double longestDuration)
{
    duration = 0;
    startTime = DateTime.MaxValue;
    stopTime = DateTime.MinValue;
    longestDuration = 0;
    for (int i = 0; i < orders.Length; i++)
    {
        FileInfo order = orders[i];
        string orderText = File.ReadAllText(order.FullName);
        DateTime start = GetStartTime(orderText);
        DateTime stop = GetStopTime(orderText);
        if (start < startTime)
        {
            startTime = start;
        }
        if (stop > stopTime)
        {
            stopTime = stop;
        }
        TimeSpan ts = stop.Subtract(start);
        duration = duration + ts.TotalMilliseconds;
        if (ts.TotalMilliseconds > longestDuration)
    }
}

```

```

        {
            longestDuration = ts.TotalMilliseconds;
        }
    }
}

/// <summary>
/// Gets the start time.
/// </summary>
/// <param name="orderText">The order text.</param>
/// <returns></returns>
private DateTime GetStartTime(string orderText)
{
    string[] orderInfo = orderText.Split(',');
    string datetime = orderInfo[3];
    return ParseDateTime(datetime);
}

/// <summary>
/// Gets the stop time.
/// </summary>
/// <param name="orderText">The order text.</param>
/// <returns></returns>
private DateTime GetStopTime(string orderText)
{
    string[] orderInfo = orderText.Split(',');
    string datetime = orderInfo[4];
    return ParseDateTime(datetime);
}

/// <summary>
/// Parses the date time.
/// </summary>
/// <param name="datetime">The datetime.</param>
/// <returns></returns>
private DateTime ParseDateTime(string datetime)
{
    int year = Convert.ToInt16(datetime.Substring(0, 4));
    int month = Convert.ToInt16(datetime.Substring(5, 2));
    int day = Convert.ToInt16(datetime.Substring(8, 2));
    int hour = Convert.ToInt16(datetime.Substring(11, 2));
    int minute = Convert.ToInt16(datetime.Substring(14, 2));
    int sec = Convert.ToInt16(datetime.Substring(17, 2));
    int millisec = Convert.ToInt16(datetime.Substring(20, 3));
    return new DateTime(year, month, day, hour, minute, sec, millisec);
}

/// <summary>
/// Handles the Click event of the btnClearOutput control.
/// </summary>
/// <param name="sender">The source of the event.</param>
/// <param name="e">The <see cref="System.EventArgs"/>
/// instance containing the event data.</param>
private void btnClearOutput_Click(object sender, EventArgs e)
{
    txtOutput.Text = String.Empty;
}

/// <summary>
/// Tallies the results.
/// </summary>
/// <param name="grandTotalFile">The grand total file.</param>
/// <param name="resultPath">The result path.</param>
private void TallyResults(string grandTotalFile, string resultPath)
{
    StreamWriter sw = File.CreateText(grandTotalFile);
    DirectoryInfo resultDir = new DirectoryInfo(resultPath);
    FileInfo[] results = resultDir.GetFiles("*.csv");
    for (int i = 0; i < results.Length; i++)
    {
        FileInfo result = results[i];
        string[] lines = File.ReadAllLines(result.FullName);
        string resultText = lines[1];
        sw.Write(Path.GetFileNameWithoutExtension(result.Name) + ",");
        sw.WriteLine(resultText);
    }
    sw.Flush();
    sw.Close();
}

```

```

        sw.Dispose();
    }

    /// <summary>
    /// Handles the MouseClick event of the btnAnalyzeResults control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="System.EventArgs"/>
    /// instance containing the event data.</param>
    private void btnAnalyzeResults_MouseClick(object sender, EventArgs e)
    {
        DirectoryInfo prod = new
            DirectoryInfo(ConfigurationSettings.AppSettings["PRODUCTION"]);

        DirectoryInfo[] outputDirs =
            prod.GetDirectories("Out", SearchOption.AllDirectories);

        for (int i = 0; i < outputDirs.Length; i++)
        {
            double longestDuration = 0;
            DirectoryInfo dir = outputDirs[i];
            DirectoryInfo plant = dir.Parent;
            DirectoryInfo simulationName = plant.Parent;
            string resultFile = simulationName.Name + "_" + plant.Name + ".csv";

            string resultPath = Path.Combine(
                ConfigurationSettings.AppSettings["RESULTS"], resultFile);

            FileInfo[] orders = dir.GetFiles();
            double duration = 0;
            DateTime startTime = DateTime.Now;
            DateTime stopTime = DateTime.Now;

            GetProductionTimes(orders, out duration, out startTime,
                out stopTime, out longestDuration);

            StreamWriter writer = File.CreateText(resultPath);
            writer.Write(PrepareHeaderMessage());

            writer.Write(PrepareResultMessage(orders.Length, duration,
                startTime, stopTime, longestDuration));

            writer.Flush();
            writer.Close();
            writer.Dispose();
        }

        txtOutput.Text = "Analysis complete.";
    }

    /// <summary>
    /// Handles the Click event of the button1 control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="System.EventArgs"/>
    /// instance containing the event data.</param>
    private void GrandTotals_Click(object sender, EventArgs e)
    {
        TallyResults(@"F:\temp\school\EAI\Results\Hub_GrandTotal.csv",
            @"F:\temp\school\EAI\Results\Hub");

        TallyResults(@"F:\temp\school\EAI\Results\Traditional_GrandTotal.csv",
            @"F:\temp\school\EAI\Results\Traditional");

        txtOutput.Text = "Grand Totals Calculated.";
    }
}

//Application Configuration File
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >

```

```

        <section name="MNSU.Mankato.CS.APP.BPMarch.Simulator.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" allowExeDefinition="MachineToLocalUser" requirePermission="false" />
    </sectionGroup>
</configSections>
<appSettings>
    <add key="HUB.EXE"
value="C:\projects\school\EAI\Hub\bin\Debug\MNSU.Mankato.CS.APP.BPMarch.Hub.exe"/>
    <add key="PLANT.EXE"
value="C:\projects\school\EAI\Plant\bin\Debug\MNSU.Mankato.CS.APP.BPMarch.Plant.exe"/>
    <add key="CUSTOMER.EXE"
value="C:\projects\school\EAI\Customer\bin\Debug\MNSU.Mankato.CS.APP.BPMarch.Customer.exe"/>
    <add key="CONFIGURATIONS" value="F:\temp\school\EAI\Configurations"/>
    <add key="REQUESTS_FOR_WORK" value="F:\temp\school\EAI\RequestsForWork"/>
    <add key="PRODUCTION" value="F:\temp\school\EAI\Production"/>
    <add key="ORDERS_IN_QUEUE" value="F:\temp\school\EAI\OrdersInQueue"/>
    <add key="CUSTOMER_ORDERS" value="F:\temp\school\EAI\CustomerOrders"/>
    <add key="RESULTS" value="F:\temp\school\EAI\Results\"/>
</appSettings>
</configuration>

```