SOA-based Maintenance Analysis of BIT System (Built-in-Test)

Li Yang
Abstract

SOA-based Maintenance Analysis of BIT System (Built-in-Test)

Li Yang

Build-in test (BIT) is a vital part of modern complex military equipment and systems. This diploma work will explore how the diagnosis and information from the BIT can be used in Lift- the Swedish Armed Forces’ Logistics Management Systems for technical service. BIT can be used both for the analysis of errors and for the assessment when appropriate maintenance should be done.

Lift-“Lednings-och Informationssystem för Förmögenhetsförörjning och Teknisk tjänster” (“The management and information systems for consumables supply and Technical Services”), is a logistics system that handles the bulk of defense equipment. The technical service includes support for maintenance of technically complex equipment. Lift consists of a number of large collaborative system and available locally, centrally and internationally. Lift is built to high safety requirements to cope with both secret and open information. This thesis works only with open information.

The purpose of the thesis was to investigate and understand the Enterprise Data Integration, Message transformation and interface with military system so that it is possible to capture Error report from military equipments to central system automatically.

Thesis work consists of two main parts, first an investigation and then a prototype. The investigation should be for a number of representative material systems (such as ships and artillery) examine whether and how the BIT error information can be extracted and transferred to the maintenance system. Along with activities to assess the type of BIT information as is relevant to both error repairing and preventive maintenance. Existing standards in the field need to be identified and compared to needs. Possibly the thesis provide a basis for standardization in this field (PLCS).

An equipments system is chosen to make a prototype. The prototype will build an SOA based collection of information from the BIT equipments system to the central IT system - Lift. It covers the chain from the interface of BIT system, collection of autonomous local Lift-computer, transformation of BIT information to common database format adapted for the maintenance analysis, replication to the central Lift-computer and access to the analytics.

Technology includes SOA, XML, Java, JMS, 4GL language, SonicESB communication bus and the Progress relational database OpenEdge. Investigation and prototyping is done under the Scrum agile development process.
Contents

**SOA-BASED MAINTENANCE ANALYSIS OF BIT SYSTEM (BUILT-IN-TEST)** .............................................. I

**ABSTRACT** .............................................................................................................................................. II

**LIST OF TABLES** ....................................................................................................................................... VI

**ACKNOWLEDGMENT** .............................................................................................................................. VII

**PREFACE** .................................................................................................................................................. VIII

1. **INTRODUCTION** .................................................................................................................................. 1

   1.1 **BACKGROUND** ............................................................................................................................ 1

   1.2 **PROBLEM FORMULATION** .......................................................................................................... 1

   1.3 **TOOLS** ............................................................................................................................................. 2

      1.3.1 Lift System ................................................................................................................................. 2

      1.3.2 Sonic ESB .................................................................................................................................. 2

      1.3.3 DataXtend™ Semantic Integrator (SI) ....................................................................................... 3

      1.3.4 Progress OpenEdge .................................................................................................................. 5

2. **PROJECT STRATEGY** .............................................................................................................................. 6

   2.1 JMS: PTP COMMUNICATION ............................................................................................................... 6

      2.1.1 Introduction ............................................................................................................................... 6

      2.1.2 JMS Architecture ....................................................................................................................... 6

      2.1.3 JMS Messaging Models ........................................................................................................... 7

      2.1.4 Core JMS API ............................................................................................................................ 8

   2.2 DXSI: DATA TRANSFORMATION .................................................................................................... 9

   2.3 OPTIMIZE: EMAIL ADAPTOR ......................................................................................................... 12

   2.4 SONIC ESB .......................................................................................................................................... 13

      2.4.1 CBR .......................................................................................................................................... 13

      2.4.2 XSLT .......................................................................................................................................... 14

      2.4.3 File Handle ................................................................................................................................ 14

   2.5 PROGRESS OPENEDGE ................................................................................................................... 14
2.6 Using DATAXTEND and SONIC Together ................................................................. 15
  2.6.1 Integration overview ....................................................................................... 15
  2.6.2 Architectural ..................................................................................................... 16
  2.6.3 Integration Steps ............................................................................................. 16
    2.6.3.1 Overview ........................................................................................................ 16
    2.6.3.2 Generating and Building Sonic Command Files ........................................ 17

2.7 Integrating SONIC Workbench with OpenEdge Architect ............................... 17

3. Specification ........................................................................................................ 18
  3.1 Overview ............................................................................................................... 18
    3.1.1 Problem formulation ..................................................................................... 18
    3.1.2 Solution .......................................................................................................... 19
  3.2 Architectural Design Specification ..................................................................... 20
  3.3 Design Components Master List ......................................................................... 21
  3.4 Design Details ....................................................................................................... 21
    3.4.1 COM1: Email Adaptor ..................................................................................... 21
    3.4.2 COM2: JMS Test Client ................................................................................. 22
    3.4.3 COM3: Routing Rule ....................................................................................... 23
    3.4.4 COM4: DXSI .................................................................................................... 24
    3.4.4.1 <dvrnr>Hbg090127</dvrnr> automatically ................................................ 25
    3.4.4.2 <ID> ............................................................................................................. 25
    3.4.5 COM5: XSLT .................................................................................................. 29
    3.4.6 COM6: File Drop ........................................................................................... 30
    3.4.7 COM7: OpenEdge ......................................................................................... 31
    3.4.8 COM8: LiftL System ...................................................................................... 32

4 Result .................................................................................................................... 33
  4.1 Running Result .................................................................................................... 33
  4.2 Original Visby Flat File ....................................................................................... 33

5. Conclusion ............................................................................................................ 37
5.1 SUMMARY OF THESIS WORK ..................................................................................................................37
5.2 LESSONS LEARNT WHEN CONDUCTING WORK .................................................................................37
5.3 FUTURE RESEARCH .............................................................................................................................37

6. GLOSSARY ..................................................................................................................................................39

7. REFERENCES .............................................................................................................................................41

8. APPENDIX ..................................................................................................................................................42
LIST OF TABLES

Figure 1: Sonic ESB work flow ................................................................................................................ 3
Figure 2: Sonic ESB Lift Cycle .................................................................................................................. 3
Figure 3: DataXtend Work Lift Cycle ...................................................................................................... 4
Figure 4: JMS PTP ................................................................................................................................... 7
Figure 5: JMS Pub-Sub ............................................................................................................................. 7
Figure 7: JMS State Transition Diagram .................................................................................................. 9
Figure 8: The Growing Semantic Challenge in SOA .............................................................................. 10
Figure 9: Common Model Architecture solution ..................................................................................... 11
Figure 10: : DataXtend I as a Semantic Service ..................................................................................... 12
Figure 11: OpenEdge Desktop .................................................................................................................. 15
Figure 12: BIT working flow ................................................................................................................... 19
Figure 13: Architectural Flow Diagram .................................................................................................. 20
Figure 14: Email Adapter Configuration ................................................................................................ 22
Figure 15: JMS Test Client Configuration ............................................................................................... 23
Figure 16: Routing rules configuration ................................................................................................... 24
Figure 17: DXSI Exchange Model Define ............................................................................................... 24
Figure 18: DXSI Attribute Transformation ............................................................................................. 25
Figure 19: DXSI Enumeration Mapping configuration ........................................................................... 26
Figure 20: Mapping List between Equipments and Lift ......................................................................... 27
Figure 21: Class Mapping between equipment and Common Model ....................................................... 28
Figure 22: DXSI final overview ................................................................................................................ 28
Figure 23: DataXtend Sequence ............................................................................................................... 29
Figure 24: XSLT configuration ................................................................................................................. 30
Figure 25: File Drop Service configuration ............................................................................................. 31
Figure 26: OpenEdge configuration .......................................................................................................... 31
Figure 27: LiftL system ............................................................................................................................ 32
Figure 28: Sonic Process tracking ............................................................................................................ 33
ACKNOWLEDGMENT

I would like to acknowledge Lars Pontén (FMV) and Professor. Ivan Christoff (Uppsala University) for their valuable comments and useful recommendations in shaping the structure of this study. Thank you very much!

Further gratitude is extended to Leif Dahlgren (FMV), Tomas Sträng (FMV), Ingvar Åström (Prosilia) Maria Elfvik (Prosilia), Urban Kvarby (Prosilia), Erik C Johansson (Baesystem), and Peter Hellekant (Baesystem) for helping me complete the Master Thesis work for FMV. Special mention is accorded to Tobias Talltorp (Lemontree), Bjørn Kroghrud (Kroghrud), Erik Steinholtz (Progress) for their encouragement and efforts made research work possible for me.

Endless gratitude is accorded to my ever-supportive father Shaoqing Yang, my loving mother Guihua Ding, my gentle brother and sister and my loving boyfriend - for their love and encouragement all throughout the struggles I went through to complete the MSc program in Computer Science and this thesis study, most specially.

Finally, I am very grateful to all my classmates and friends in the MSc course who generously shared their knowledge and experience with me in my study and stay in Uppsala.

Li Yang

16-Aug-2009
Preface

This thesis is submitted in partial fulfillment of the requirements for a Master Degree in Computer Science. It contains work done from February to July 2009. My supervisors on the project have been Leif Dahlgren (FMV), Lars Pontén (FMV) and Professor Ivan Christoff (Uppsala University). The thesis has been made solely by the author; most of the text, however, is based on the research of others, and we have done our best to provide references to these sources.

Writing this thesis has been hard but in the process of writing I feel I have learned a lot and offer me great opportunity to experience industry-based work! I have dealt with a lot of subjects, in an attempt to give this thesis a broad perspective on enterprise-based data integration, thus combining many aspects of theory and pretty practical conduction interaction.

This thesis is divided into two main parts. Part one consists of a series of enterprise data integration theory and conception which used for investigation and analyzation. Part two consists of five chapters that deal with how this thesis is conducted and implemented.

Due to the special references resource of this thesis (i.e. most of the references are from the Progress Software company which FMV has supplied software tools to FMV), the author has to try her best to give the more information of the theory. Therefore, the contents of this thesis are relatively substantial.

Glossary presents some important phases which used for this thesis work.
1. Introduction

1.1 Background

Försvarets materielverk (FMV) supply technology for the Swedish security. Here interacts technology and professionalism to procure, operate and decommission systems and services to Defense and other authorities within the civil security sector. FMV's main task is to provide the defense with cost-effective and innovative technical solutions and materials. Defense is a major customer, but the FMV is also working on behalf of other authorities and implementing export on behalf of the Government (FMV).

The new SOA-based defense supposes to lead the Swedish military using network-based technology to handle the error and maintenance instead of handling manually. This working flow makes the military system automatically handle error and maintenance information based on the BIT system with communication with the central Lift system.

1.2 Problem Formulation

Nowadays, the military equipments (e.g. ships and cannon) just collect necessary information but lacking of communication with Lift system and all of the operations has to be handled manually. Additionally, information format and type cannot match with the needs in Lift system. Therefore, it becomes a bit tricky to diagnose and analyze in Lift side and it's time consuming as well. Therefore, figure out a way to handle error and maintenance becomes a vital part for FMV.

The purpose with the thesis is to examine whether and how the BIT error information can be extracted and transferred to the maintenance system. Along with activities to assess the type of BIT information as is relevant to both error repairing and preventive maintenance. Furthermore, information through the work flow used by FMV need to be standardized in order to be organized formally within the military system (PLCS).

Through extensive study of literature and cooperation with people who are working with Enterprise Data Integration field, I investigated how requirements and criteria for the new information presentation could be designed.
1.3 Tools

This SOA-based BIT system based on the following technologies: SOA, XML, JMS, Data Integration, Relational Databases. Therefore, we need to describe the import enterprise tools for this project.

1.3.1 Lift System

LIFT stands for Lednings-och Informationssystem för Förnödenhetsförsjning och Teknisk tjänst (FMV)(Management and Information Systems for consumables supply and Technical Service) in the Armed Forces, consumables supply and technical service. Lift provides support for planning, monitoring, implementation and monitoring of equipment maintenance.

The system is currently composed of a number of local installations (Lift-L) and an overall central system (Lift-C). Both systems handle open and secret information, and Lift-C also qualified confidential information (H / TS) because of information summary of the whole country. Lift-C is built with graphical interface, while the lift-L currently has a character-based interface (FMV).

In additional to systems Lift and Lift-L-C, in the LIFT family there is also system Lift-G (Basic Data), which is linked to Lift Output System DU-web that presents various reports on the various different ways.

1.3.2 Sonic ESB

Progress® Sonic ESB® is a messaging-based enterprise service bus that simplifies the integration and flexible re-use of business applications within a service-oriented architecture (SOA) (ESBAp). Sonic ESB eliminates the rigidity and fragility of point-to-point integration with a robust, event-driven architecture that can evolve, scale and extend throughout the enterprise. Through patent-pending Continuous Availability Architecture (CAA), Sonic can guarantee timely and continuous delivery of mission-critical business events. As show in Figure 1
Progress® Sonic Workbench™ is an Eclipse-based SOA toolset to model, configure, test and deploy processes and services using products in the Sonic ESB Product Family. Its graphical tools make it easy to model business processes and subsequently specify configuration and deployment details, supporting an intuitive "top-down" approach to the SOA development lifecycle.

Sonic Workbench provides an integrated SOA toolset to streamline the lifecycle of an ESB project: Process Modeling, Configuration, Testing & Debugging, Deployment. As shown in Figure 2.

The Sonic ESB Product Family also includes development and runtime integration with Progress® DataXtend™ Semantic Integrator (SI), which dramatically simplifies the problem of common data model lifecycle management, transformation and validation.

### 1.3.3 DataXtend™ Semantic Integrator (SI)

DataXtendSI is used to achieve data interoperability with common-model-based data services inside SOA (Service-oriented Architecture). It provides an overview of product features and benefits with the goal of gaining insight into where the product fits into the development cycle as well as different architectural
patterns for deploying the services it generates. DataXtend SI enables business analysts, architects and
developers to create, maintain and govern common-model-based data services inside SOA, providing data
interoperability that fulfills the promise of SOA (DXSI).

DataXtend SI improves both development and runtime, with common eclipse tooling and the ability to
deploy semantic services in ESB containers automatically. As Sonic ESB helps organizations eliminate
rigid architectures of point-to-point connections, DataXtend SI solves the problem of point-to-point
transformations, making it much easier to integrate data and evolve an SOA with diverse connected
systems. (DXSI)

In summary, DataXtend SI supports you from design to runtime and reduces the effort required to modify
and upgrade the system over its lifetime, as shown in the following illustration: Figure 3

At the center of DataXtend SI are two components, the Designer and the Engine (DXSI):

- **DataXtend® SI Designer™** is a complete graphical design environment for creating and
  managing Exchange Models (mediations between applications and services with different
  structures and semantics), rules, and data services. DataXtend SI Designer imports existing
  schemas for data sources, data services, and the common information model, where they can be
  enriched with transformation and validation rules using DataXtend SI's comprehensive set of
  visual tools.

- **DataXtend® SI Engine™** is the runtime component of the DataXtend SI architecture. At runtime,
  data is dynamically converted and validated before it is submitted to backend systems. All data
  services use standard interfaces for incorporation into any business process or target application.
The output of DataXtend SI Engine is Java, running as stateless services with optimal performance.

1.3.4 Progress OpenEdge

Progress® OpenEdge® is an integrated platform for the rapid development, deployment and management of standards-based and service-oriented business applications (OE). It aims to simply the job of creating and operating the business applications. A software architecture that represents the business in a high productive way OpenEdge's unified environment is comprised of development tools, application servers (in this thesis work, use to explore connection to Lift server), application management tools, a relational database, and the capability to easily connect and integrate with other applications and data sources (in this thesis work, use to connection with LiftL) In a nutshell, OpenEdge simply the job of creating business applications.

The advantages of using OpenEdge is that Integration is inherent; a platform that allows you to use whatever UI technology you want; a database that is high performance, highly available and reliable, and highly scalable and it focus is 100% on the RDBMS and development tools. (OE)
2. Project Strategy

Tasks for implementation divided into three main parts:
1st extracting and transfer information from military equipments to SonicESB queue by using JMS communication mechanism.
2nd Transferring and transforming data within SonicESB and DataXtendSI.
3rd building communication interface between BIT system and Lift System.

2.1 JMS: PTP communication

2.1.1 Introduction

JMS or Java Messaging System is an asynchronous communication API for Java applications. Java Message Service was developed by Sun Microsystems to provide a means for Java programs to access enterprise messaging systems or message oriented middleware (MOM) (JMS). They provide a mechanism for integrating applications in a loosely coupled way. They provide asynchronous delivery of data between applications on a store and forward basis; i.e., the applications do not communicate directly with each other, but instead communicate with the MOM, which acts as an intermediary, thus handling all network communication details for you. If e.g. the network connection is not available, the MOM will store the message until the connection becomes available, and then forward it to the destination. Thus, the JMS API is characterised as asynchronous and reliable (it can ensure that a message is delivered once and only once).

2.1.2 JMS Architecture

The main players of the JMS architecture are the following:

- JMS Provider: A messaging system that implements the JMS specification. A JMS Provider, also known as a JMS Server, routes messages between clients.
- Clients: Java applications that send or receive JMS messages. A message sender is called the Producer, and the recipient is called a Consumer.
- Messages: Messages contain data or events exchanged between Producers and Consumers.
- Destinations: A Producer sends a message to a JMS Destination (either a Queue or a Topic), and the Consumer(s) listening on the JMS Destination receives the message.
2.1.3 JMS Messaging Models

JMS has two messaging models (JMS): Point-to-Point (P2P) and Publish-Subscribe (Pub-Sub). P2P (As shown in Figure 4) is a traditional one-to-one queuing mechanism, i.e. although multiple Consumers can listen on a queue, only one Consumer receives a particular message. Producers send messages to a queue, and the JMS Provider (aka JMS Server) delivers each message sequentially to a Consumer listening on the queue. The following figure shows the relationships between Point-to-Point Producers and Consumers.

![Figure 4: JMS PTP](image)

Publish-Subscribe (As shown in Figure 5) is a one-to-many broadcast model, similar to a newsgroup or a bulletin board or an RSS newsfeed. Producers publish messages to a topic, and the JMS Server delivers messages sequentially to those Consumers subscribed to that topic. The following figure shows the relationships between Pub-Sub Producers and Consumers.

![Figure 5: JMS Pub-Sub](image)
The biggest difference between P2P and Pub-Sub is that in the Publish-Subscribe Model, all Consumers that subscribe to a Topic can receive all messages published to that topic, while with Point-to-Point; only one Consumer on a queue receives a particular message.

Even though JMS is inherently asynchronous, the JMS specification allows for messages to be consumed in either of two ways (JMS):

- Synchronously: A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
- Asynchronously: A client can register a message listener with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage() method, which acts on the contents of the message.

A JMS message consists of a header, properties and a body. The **header** is a standard set of fields that are used by both clients and providers to identify and route messages; it contains a message id, destination, timestamp, priority etc. The **properties** provide a facility for adding optional header fields to a message e.g. for categorization or classification, to provide compatibility with other messaging systems, or to use them to create message selectors. JMS defines a standard set of properties that are optional for providers to supply. The **body** contains the content to be delivered to a receiving application. The body or content of the JMS message can be many things, e.g. XML, a JavaBeans (i.e. an object) etc. JMS API provides five message body formats: text, map, bytes, stream, and object. To send an object as a JMS Message, it must obey the following rules:

- An object must implement java.io.Serializable.
- Each data member must be serializable. By default, String, the Java primitives (int, float, and so on) and the Java primitive wrappers (Integer, Float, and so on) are all serializable.

### 2.1.4 Core JMS API

The JMS API resides in the javax.jms package. These are the most important classes and interfaces for our purposes (JMS):
- **Message**: Holds business data and routing information. Although you'll find several types of Messages, most of the time you'll use either a TextMessage (that contains textual data in its body) or an ObjectMessage (that holds serializable objects in its body).

- **Destination**: Holds messages sent by the Producer to be received by the Consumer(s). A Destination is either a Queue or a Topic that the JMS Server manages on behalf of its clients.

- **Connection**: Enables a JMS client to send or receive Messages. Use a Connection to create one or more Sessions.

- **ConnectionFactory**: A ConnectionFactory is either a QueueConnectionFactory or a TopicConnectionFactory, depending on the messaging model, and it exists to create Connections.

- **Session**: A Session creates Producers, Consumers, and Messages. A Publish-Subscribe application uses a TopicSession, and a Point-to-Point application uses a QueueSession. Sessions are single-threaded.

![Figure 6: JMS State Transition Diagram](image)

### 2.2 DXSI: Data transformation

One of the most expensive and complex challenges of integrating business applications is ensuring the validity of data exchanged between systems. Over 40% of the cost of business integration today is spent on manually reconciling and validating inconsistent data exchanged between enterprise systems (DXSI). As shown in Figure 7

Data has different meanings (semantics) in different systems. In addition, many data transformations between systems are complex. The challenge of identifying and maintaining semantic mappings between systems can require tremendous effort. According to the official investigation with revealing data integration efforts are increasing across most major industries as a result of the demand for real-time,
globally accessible data, over 80% of organizations amplify this effort by reconciling physical differences using hand coded point-to-point mappings (DXSI).

Even with SOA, mapping and validating data between systems is largely still done manually using XSLT and Java coding, resulting in tightly coupled, PTP data integration even when business logic is loosely coupled. If changing the data representation in one system, you have to manually adjust mappings to other systems, up to \( n^2 \) changes, where \( n \) is the number of integrated systems. Without careful attention to data architecture, integration remains difficult.

![Accidental Architecture of Data](image)

Figure 7: The Growing Semantic Challenge in SOA

DataXtend SI uses common model integration (CMI: as shown in Figure 8) to relate systems that interact with each other. With a CMI approach to data integration, all systems map to a single data model instead of directly to each other. It addresses semantic data integration, letting you specify rules based on the meaning of data and your business constraints. DataXtend SI lets us design and test data service in a graphical, interactive development tool, and then deploy them in any Java environment. It provides semantic integration for implementing common model architecture for SOA and other enterprise architectures, capturing complex transformations between data formats without writing code, etc. A common model is an information model through which all data from services participating in the integration is transformed.
One challenge of using common model integration is ensuring the common model is adequate for the required integrations. A common model should be comprehensive, abstract, and extensible. Industry-specific common models meet these requirements, but these qualities can also make them complex for simple integrations and difficult to maintain.

A business component in a system invokes a data service when it requires some operation performed on data. It sends a request to the data service, passing it input data as part of the request. The data service performs its operation and returns a response to the calling process. While performing its operations, a data service might interact with data sources, which are external systems such as databases, Web services, or legacy systems. It interacts with these by transforming its input into the common model format, then transforming the common model format into the format required by the data source interface.

In DataXtend SI, a map is a construct that defines how a data service validates, transforms, and aggregates data at runtime from one representation to another. The schemas which import into data service and data source models define the format expected by their interfaces, mapping these formats to and from the common model. Maps are hierarchical, the levels include (DXSI): Model map, Schema map, Class map and Transformation between attributes. Transformations are maps between attributes, at runtime, they move the actual data values between models, or between combinations of attributes that have the same meaning.
The flexibility of DataXtend SI allows you to transform messages from one format to another, aggregate data from multiple data sources, and/or transform messages into a canonical format. As mentioned previously, you can use data services in a message-bus architecture, as Web Services, in a supported EJB server, or in any Java-based integration framework. As shown in Figure 9

![Figure 9: DataXtend SI as a Semantic Service](image)

DataXtend SI includes a Sonic service that simplifies use of data services in an ESB. The Sonic service allows you to invoke a data service operation from the context of a Sonic process. The itinerary of a Sonic process resembles a flow chart that describes the steps performed by the process. You can seamlessly drag and drop the DataXtend-generated exchange model files into the Sonic Workbench itinerary to create the DataXtend SI service.

At runtime, the SI Sonic service type receives the message when it is the next step of the itinerary, runs the operation, and then passes the result on to the next step of the itinerary. Sonic uses strings to pass around XML messages, so the SI Sonic service marshals from the string into Dataxtend entities and back.

### 2.3 Optimize: Email Adaptor

In order to extract information from military equipments, we need to get the information automatically when the equipments side sending report via email, so here come Email Adaptor, which help us to get the email information automatically and put it directly to Sonic ESB queue which we will analyze.
2.4 Sonic ESB

2.4.1 CBR

After Email Adapter (we can browse queue: qMail and see message we have got), which is fine for Archer part, while for Visby part, need to drop to disk, how should we do? How can we distinguish what kinds of message we get and how to handle it? The answer is CBR (Content-Based Routing).

Sonic ESB provides the Content-Based Routing (CBR) service type that allows you to incorporate message routing decisions into your applications apart from the business logic (SonicESB). For all of the routing scenarios, CBR services use an embedded rules engine to determine routing destinations. I will explain it as follows:

A Content-Based Router pattern routes a single incoming message to exactly one destination (out of many) based on the content of the message.

1st. An XQMessage arrives at the Entry endpoint for a CBR service. The Dispatcher (an ESB framework component running in the Container) pre-addresses the message with the Exit endpoints for the service and places the message in the Inbox. Then the Dispatcher hands off the service context to the CBR service.

2nd. The CBR service uses the service context to get the incoming message and determine the rules in effect. A standalone CBR service uses rules specified in its service configuration. When the service is used in an ESB process, optional runtime parameters on the CBR service step override default rules.

3rd. The rules and the message are passed off to a rule engine that evaluates them and returns one or more ESB addresses.

4th. The CBR service readdresses the message according to the results of rule evaluation and places it in the Outbox. The Dispatcher takes the message out of the Outbox and sends it to the ESB addresses.

Particularly, there is a CBR called XCBR (XPath routing rules). Each XCBR routing rule consists of a rule condition and one or more rule addresses. The rule condition is a Boolean-valued XPath expression applied to targeted content in a message. The rule addresses are ESB processes, services, or endpoints, or step names. These rules are flexible because they can return: 1st One or multiple addresses; 2nd A default destination if no rules evaluate to true.
2.4.2 XSLT

An XML Transformation service applies an XSLT stylesheet to XML in the body of a message, changing the content from one form to another (SonicESB). The ability to transform message contents is particularly useful when different recipients of a message expect it in different XML formats. Sonic extensions to XSLT make it possible to create stylesheets that access message headers or generate new messages.

An XSLT stylesheet contains elements for transforming source XML documents into result XML documents. When a stylesheet is applied to a message, the template is instantiated to create each part of the result document. A pattern associated with the template is matched against the source document, binding matching document elements to the template on each evaluation.

In addition to transforming the body of a message, in many cases you want to transform message headers as well for example to copy selected headers from an incoming message to an outgoing message, or to set values in the body of the message into the headers like the reply to URL (SonicESB). Sonic header extensions give XSLT stylesheets the ability to read and write header properties on a message, including JMS header values and user-defined properties. Use header extensions to get property and header values from a source document and set property and header values in a result document.

Another use of an XML Transformation service is to create new output messages from an input message. We use Sonic message extensions to XSLT in the stylesheet to implement this scenario. This is a more flexible means of implementing a Splitter pattern than the Splitter service and has a wider range of application.

2.4.3 File Handle

After transform to destination format with DXSI (using the common model as intermediate model), we need to put the message on a file using File Drop service.

2.5 Progress OpenEdge

Progress® OpenEdge platform is standards-based, integrated, and capable of building, running, and managing service oriented business applications (SOA). OpenEdge relational database is based on 4GL language which is a high-level programming language. We can use 4GL to write (4GL):

- Client-side code for Graphical User Interface (GUI) clients, Character clients, WebSpeed
clients, and Web Clients.  
- Business logic that can reside on the server.  
- Interface logic to move data between client and server.  
- Database logic in the Data Dictionary.

For this work I use the following tools within OpenEdge: Figure 10

\[\text{Used to write and test 4GL procedures.}\]
\[\text{Set of visual programming tools used to create and assemble application components, includes the Section Editor, used to create and edit application files.}\]
\[\text{Used to copy or create a database, to define the database structure, including tables, fields, and indexes, and to generate reports on this information.}\]
\[\text{Used to step through a program to locate errors.}\]
\[\text{Set of utility programs related to the process of developing and running applications.}\]

Figure 10: OpenEdge Desktop

Use the 4GL primarily to write business logic and applications, how to develop the business logic in the 4GL that can be centrally located on the server, and accessed by many client types. And then client consume message with JMS or other mechanism.

2.6 Using DataXtend and Sonic together

2.6.1 Integration overview

While Sonic and DataXtend SI both simplify development of service-oriented architectures, each has a different focus. DataXtend's common model architecture excels at handling not only data transformation, but semantic mediation, where a data service accesses multiple sources to satisfy a request. Sonic provides a robust platform for messaging, process orchestration, and security. An architecture that leverages the strengths of both products provides significant benefits.

The Sonic workbench is an Eclipse plug-in that provides a design and testing environment for Sonic ESBs. The DataXtend SI installer has an option to link the SI workbench into an existing Sonic workbench
(DXSI). With the two workbenches using the same Eclipse, you can drag and drop data services from DataXtend projects onto a Sonic project. The installation also configures a DataXtend test container and service in the Sonic workbench.

2.6.2 Architectural

The two approaches require different treatment in both the Sonic and DataXtend projects (DXSI):

- As a transformation service, a data service uses Sonic data sources and you configure it to return the data source message(s). In the Sonic project, you need to then route those messages to the process step(s) that access the data source(s) and handle the response.
- As a semantic mediation service, a data service accesses data sources directly. The result from the data source(s) goes back through DataXtend, allowing it to process the response. To accomplish this, you need to expose required data sources in the Sonic project as simple, atomic ESB services. The ESB service acts as a technology adapter, presenting the data source interface in an XML format that you can import into the DataXtend project.

2.6.3 Integration Steps

2.6.3.1 Overview

A Sonic ESB process itinerary defines the sequence of process steps through which an ESB message will pass. DataXtend data services can be used as process steps to provide transformation and semantic mediation for message data. Each DataXtend process step has configuration parameters that are stored in a command file (with an .esbdx extension). After a DataXtend project has been completed and tested, you can enable Sonic command file generation for some or all of the data services in a DataXtend project. Separate command files support document and RPC style messages.

You add data services as process steps by dragging the appropriate command file to the ESB Process editor in the Sonic Design perspective. Once in the ESB process itinerary, you can rearrange the order of the process steps and configure them. You will need to map each process step's request and response parameters. After you have used the DX Design perspective, you will be able to switch back and forth between the Sonic Design perspectives.
2.6.3.2 Generating and Building Sonic Command Files

After have defined and tested the DataXtend project, use the DX Design perspective to configure the data services. For each data service that you want to use as an ESB process step, from the Exchange Model editor, select a data service for which you want to generate Sonic command files. After have configured the appropriate data services, save and build the project. Afterward, we can drag and drop the generated command files into a Sonic process, and add Semantic Integrator data service steps to a Sonic ESB process.

2.7 Integrating Sonic Workbench with OpenEdge Architect

The compatibility between Progress® / OpenEdge® and Sonic product releases, documenting what platforms the Progress and OpenEdge Adapter for SonicMQ and OpenEdge Adapter for Sonic ESB can be installed and run on.

The OpenEdge Adapter for SonicMQ and the OpenEdge Adapter for Sonic ESB may be installed and used on any platform that is supported by both Progress/OpenEdge and by SonicMQ/ESB, as indicated in the tables supplied. There are three adapters: Progress/OpenEdge Clients and the Progress/OpenEdge Adapters for Sonic (MQ and ESB), SonicMQ Clients and the SonicMQ Brokers and Progress/OpenEdge Adapters for Sonic (MQ and ESB) and the SonicMQ Clients. For this thesis work, we use the last one to integrate Sonic with OpenEdge (OE).
3. Specification

3.1 Overview

3.1.1 Problem formulation

The following figure shows our problem situation, it shows two main independent sides, one side is the military equipments such as ships, cannons and airplanes. The other side is LiftL system, which goes further distributing to DUweb, economical-system, engineering systems, etc. From this figure we can clearly see that many equipments systems go through many IT- systems. The problems come up are how to build a common SOA-based Maintenance Analysis system to set up communication between the two sides? How to make a common format for the different military equipments and retrieve information from equipments to Lift-system?

![Figure 11: Problem Formulation](image)

And this also brings up our integration problems: What kinds of information are we interested and want to retrieve: Operating values/environments, machine records and configuration, etc; what kinds of technique should we use to set up communication between two sides: USB, cable, GSM/3G, Satellite etc; what kinds of Transfer Protocol should we use: file, email and message-bus (garnered transfer).
3.1.2 Solution

This document provides a high-level implementation description of the BIT (Build-in Test) to LIFT (Lednings- och Informationssystem för Förnödenhetsförsörjning och Teknisk tjänst). The main working flow is shown in Figure 12.

![Figure 12: BIT working flow](image)

From this figure, we can see the left side is military equipments and the right side is Lift system as we have shown the previous figure. I worked for the middle part (yellow part) which start from SonicESB to XMLinläsning and also make connection between two sides.

For the left side, information from guns is collected and formulated by Befors, they make communication between guns and their own system-BAE Systems Infrastructure and store within database through Mobil Lane, so Befors is responsible for this part; while for ships, they use USB to get information from ships and send the reports via email to my part.

For the right side, we have other team (Prosilia) which works for the right side-LiftL system.

For my diploma work (the middle part), I’ll give details as follows.
3.2 Architectural Design Specification

Here is the main working flow for this thesis work. And it contains enterprise data integration and JMS messaging passing mechanism, so I will explore it into several steps and describe how we implement it. This flow diagram is based on Sonic-Workbench and focused on Sonic ESB and DataXtendSI, I will also explore other important parts which cannot include in this flow diagram. As shown in Figure 13

![Diagram](image)

**Figure 13: Architectural Flow Diagram**

When the equipments side sends report, we use the Email Adapter to get it and put it to SonicESB. Since we have different files format between Visby and Archer which we need to handle respectively, so we must decide and recognize what kinds of information we got from email, so here we use Routing Rule as a
service within ESB process. If it’s from Visby then we need DXSI to transform data and handle data integration, and then we use XSLT to extract elements information from XML file within ESB process, then using File Drop service to disk and read the dropped file to LiftL by using OpenEdge. While it’s from Archer, then we just put it directly to ESB process and File Drop to disk and using OpenEdge read the dropped file to LiftL. Scanning Module LXMLin retrieve messages from the bus. Load collected data from the Lift database and stores in the database.

### 3.3 Design Components Master List

This section contains the listing of all requirements for the BIT system. The list contains unique requirements numbers and names with a short description of each requirement. The following section describes these requirements in full detail.

- COM1: Email Adaptor
- COM2: JMS Test Client
- COM3: Routing Rule
- COM4: DXSI (DataXtend SI)
- COM5: XSLT
- COM6: File Drop
- COM7: OpenEdge
- COM8: LiftL System

### 3.4 Design Details

#### 3.4.1 COM1: Email Adaptor

Configure Email Adaptor so that we can get information automatically. Importing .xar file that contains the ESB artifacts needed for getting started with the mail service into the DomainManager. Use Start- >Progress->Sonic 7.6->Tools->ESB Deployment Import Tool.
Open the ESB Deployment Import Tool and connect to the Domain, import the MailService.xar file and press the 'Import the Archive' button. Open the Sonic Management Console (SMC), configure containers and components which are compulsory for using Email Adaptor. In the Configure tab specify 'ctMail' as Name and set username and password and add Component to ESB container 'esbMail', expand Brokers and MgmtBroker specify 'qMail' as Name. Expand SMC and select Activation Daemon specify adMail as Name and then look up the new container to add Component.

After configuration, we get the following diagram (as shown in Figure 14), and right now we can wait for new email comes and put it on the Sonic Queue qMail and Browse it.

![Figure 14: Email Adapter Configuration](image)

Additionally, I also develop a monitoring mechanism for the coming message, which ensures that the measured message are put on Sonic ESB process and how much information has put into, we can also browse the history information we have got before.

### 3.4.2 COM2: JMS Test Client

The JMS Test Client is useful for testing the client applications that you write. You can use the test client to consume messages produced by your application, or to produce messages for your application to consume, to create JMS messaging components and exchange messages. As shown in Figure 15
1st step need to connect to tcp://localhost:2510 to make sure the SonicESB works, where create queues for sending and receiving by using JMS.

2nd step need to connect to tcp://localhost:2506 to make sure EmailAdaptor works, where create queue for attracting contents from coming email.

![Figure 15: JMS Test Client Configuration](image)

### 3.4.3 COM3: Routing Rule

Define the Routing rule to decide what kinds of file we got from Email, either from Archer or Visby. Since they have different contents and need to be handled differently. If it’s Visby, then need to convert the flat file to xml format and then go to DXSI first. While if it’s Archer, then put on the SonicQueue Directly. Consider this, we decided to use the XPath routing rules to make a decision whether it’s Visby or Archer, since each XCBR routing rule consists of a rule condition and one or more rule addresses. The rule condition is a Boolean-valued XPath expression applied to targeted content in a message. As shown in Figure 16.
Notice: When you use a CBR service in an ESB process, either to create branching or to forward in-process messages, optional runtime parameters on the CBR service step override default rules.

### 3.4.4 COM4: DXSI

After DXSI, we can add DXSI as a transformation service into ESB process as we described before.

First we need to build communication models based on xml schema (Common model, Data Service, Data Source) in DXSI. As shown in Figure 17

![Figure 17: DXSI Exchange Model Define](image)

After some necessary preparation and we have to start the most important part, data transformation with DXSI. There are some tricky parts we need to take care; we need map data between different formats; we need create expressions for complex transformations.
Content-Based Routing in this thesis work, defined this routing using a precondition and a computed attribute. When you map a class in the data source to the same class in the common model to which you mapped a data service class, DataXtend SI will execute that map at runtime. The map causes a message to go out to the data source. A precondition is a boolean rule or method that must return true at runtime for the class map to be executed at all.

3.4.4.1 <dvrnr>Hbg090127</dvrnr> automatically

How to extract only the first one as fileName, not all of them? And as for the filename, it contains two elements from the file, what should we do?

Solution: Using XPath XSLT to extract information what we want

3.4.4.2 <ID>

We need map the <ID> element of Data Service to <mvtyp> element in Common Model. First we try to give the values which we do not need in Lift as default -1, but it cannot recognized at Lift side, so at last we decide to create a list for mapping the necessary elements and ignore the rest. Here, we apply the advanced technique within DXSI, such as Validation rules, Conditional mapping and Enrichment. So we build an Enumeration Map "idusIDToMvtyp" (as shown in Figure 19). Fist need to do precondition for the transformations under class map. As shown in Figure 18

![Figure 18: DXSI Attribute Transformation](image-url)
Because in equipments side, they have the unique ID number with two digits, while in Lift side using another unique id number with three digits, in order to analyze the error report in Lift side from military equipments, we need to do this mapping. The id number with the equipments and Lift sides are mapped as follows (Figure 20).
Other values, which are not existed in short xml file which converted from Visby flat file, we need to put it in the new xml file (we can call it Visby.xml file, of course for Archer as well), but we need to put it in Lift.xml file and let communication with LiftL system. For example, we have following elements: <hmv>, <confid>, <fandnr>, <andkl>, <radar>. We need to new transformation under the class map between the root elements.

Then we get the class-mapping under the connection between Data Service and Common Model (Figure 21)
The final transformation is like this (Figure 22), and we can also see how the logic executed step-by-step from DXSI Sequence (Figure 23).
After DataXtend transformation part is done, we need to export DXSI project and add into SonicESB as one of the service within ESB process. For this we need to generate boot file within DXSI, and then drag the generated file into ESB process. Here we must restart the container since we must reload the container whenever you deploy or modify a service or process or change the container.

3.4.5 COM5: XSLT

We need give the filename from elements within the file, and here we use XPath to extract information and format the new xml file name by using xslt, and here we need Accessing JMS message headers, so we also need to add header extension function names and saxon script element are recognized by the transformation engine.

Since we need to extract information from the elements in XML file, here we use XPath to make a rule and extract information. As shown in Figure 24
3.4.6 COM6: File Drop

After transformation of DataXtend SI, we need to drag file to local disk and then read it to LiftL system through Progress OpenEdge program.

We need to configure the SMC and add Entry Point and End Point for ESB process. In this thesis work, the message goes to the File Drop service after transformation. This service uses an incoming header property from the XSLT service to generate the FileName. The file drop service performs the drop by writing the file to the specified drop directory and sends status messages to an external endpoint to report on the status of file drops. The main code of File Drop service is as follows (Figure 25):

```xml
1 <xsl:stylesheet version="1.0" encoding="UTF-8">
2 </xsl:stylesheet>
```

![Figure 24: XSLT configuration](image)

Figure 24: XSLT configuration
3.4.7 COM7: OpenEdge

After the last service (File Drop) of ESB process is done, we need to read the file into LiftL system to analyze. For this, Progress OpenEdge helps us make connection and communication with LiftL system with 4GL language. Then we need to run the Progress OpenEdge explore to connect to LiftL database and make sure the server is running (Figure 26).

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<fileDropRequest>
    <statusDestinations>
        <destinationAddress>
            <name>dev.Drop.Entry</name>
            <type>endpoint</type>
        </destinationAddress>
        <destinationAddress>
            <name>dev.exit</name>
            <type>exit</type>
        </destinationAddress>
    </statusDestinations>
    <filename type="FROM_HEADER">FileName</filename>
    <dropDirectory>C:LXML</dropDirectory>
</components>
</fileDropRequest>
```

Figure 25: File Drop Service configuration

Figure 26: OpenEdge configuration
3.4.8 COM8: LiftL System

Run “Progress Explorer Tool” and make sure to connect to LiftL Database. And then run code “ReadXML” which will put the dropped file into Lift Database, when it’s done, we can check if it’s already in Lift and analyze the Error report. As shown in Figure 27

![LiftL System](image-url)

Figure 27: LiftL system
4 Result

4.1 Running Result

When everything is done, the only work we need to do is press run (scenario) button, wait and view the results. Before we do this, I recommend adding a Process Tracking service into ESB process, which can help us track each service. Additionally it’s useful for us to know which message we have got from email, either Visby or Archer after XCBR service, for instance, we can notice the content is from Visby from the following diagram (Figure 28).

![Figure 28: Sonic Process tracking](image)

4.2 Original Visby flat file

We get the original Visby flat file from the military equipments via email, here using Email Adaptor to extract information automatically and put it on the Sonic ESB with JMS messaging mechanism. We can browse queue (qMail) with JMS Test Client and view the whole message (subject, header and content) got from email.

Now I will show how the working flow works with the results.

First comer Visby flat file through email, and now we will see how the flat file will be transferred automatically and transformed into the xml file matching LiftL system.

Visby flat file: PMSData - 2009-03-18.txt

- BOGP-Drifttid;57;19,40;2009-03-18 23:55:00
- Förlig kapstan-Drifttid;40;0,0;2009-03-18 23:55:00
- DBRO-Boosterpump1-Drifttid;58;35,60;2009-03-18 23:55:00
- DBRO-Boosterpump2-Drifttid;59;1,30;2009-03-18 23:55:00
• DBRO-Total volym;42;53,92;2009-03-18 23:55:00
• DV-Dricksvattengenerator-Drifttid;11;739,54;2009-03-18 23:55:00
• GEN1-Drifttid;8;2021,76;2009-03-18 23:55:00
• GEN2-Drifttid;9;2110,92;2009-03-18 23:55:00
• GEN3-Drifttid;10;1982,56;2009-03-18 23:55:00
• HFM BBI-No Of Starts;28;232;2009-03-18 23:55:00
• HFM BBI-Drifttid;4;331,90;2009-03-18 23:55:00
• HFM BBY-No Of Starts;26;225;2009-03-18 23:55:00
• HFM BBY-Drifttid;5;318,90;2009-03-18 23:55:00
• HFM SBI-No Of Starts;25;76;2009-03-18 23:55:00
• HFM SBI-Drifttid;2;210,90;2009-03-18 23:55:00
• HFM SBY-No Of Starts;27;316;2009-03-18 23:55:00
• HFM SBY-Drifttid;3;226,90;2009-03-18 23:55:00
• LFM BB-Drifttid;7;834,0;2009-03-18 23:55:00
• LFM SB-Drifttid;6;826,0;2009-03-18 23:55:00
• Drivlinor-Drifttid;29;1244,92;2009-03-18 23:55:00
• MRGC BB-Drifttid;22;1237,57;2009-03-18 23:55:00
• MRGC SB-Drifttid;23;1227,57;2009-03-18 23:55:00
• SKUM-Tanknivå;50;1953,33;2009-03-18 23:55:00
• SMO-Mineralolja-Nivå;44;451,8;2009-03-18 23:55:00
• SMO-Syntetolja-Nivå;46;260,29;2009-03-18 23:55:00
• TRYCKLUFT-Kompressor 1-Drifttid;60;981,64;2009-03-18 23:55:00
• TRYCKLUFT-Kompressor 2-Drifttid;30;974,4;2009-03-18 23:55:00
• VSD-Hydraulolja-Nivå;48;339,80;2009-03-18 23:55:00

Once we get the message and put it on Sonic ESB, we use FileDrop service to drag file to local disk, and convert Visby flat file into XML format with Java code, then we get the following Visby-xml file, in order to simplify the show, so I just show one element from the Visby flat file.

Visby-XML: visby.xml

• <?xml version="1.0" encoding="utf-8"?>
• <operationsreport>
• <ship> Helsingborg </ship>
• <reading>
Afterwards, using JMS messaging mechanism put the converted Visby-xml file on Sonic ESB again and transformed within Sonic process.

After a series of operation with the different tools, finally we get the xml file which can be recognized and analyzed in Lift System, in order to simplify and clearly so I just show one element which visby. xml file has shown (as file name has changed to what Lift side wants).

Lift XML: Visby-hbg-090318.xml

- <ttUtgodfimvh>
- <myndkodm>0</myndkodm>
- <ordernr />
- <avisyste />
- <iutyp />
- <indnr />
- <artid />
- <bgtar>2009</bgtar>
- <mvdat>2009-03-18</mvdat>
- <sekvnr>1</sekvnr>
- <atgkodg/>
- <atgvers/>
- <trkod>adf</trkod>
- <mvfunk/>
- <mv>53.92</mv>
- <dv>53.92</dv>
- <mvslut>0.0</mvslut>
- <statdat/>
<mvtyp>510</mvtyp>
<mvsort />
<mvp />
<mvdec />
<hmv>false</hmv>
<anddat />
<cofind>false</cofind>
<cofdat />
<dvbgartarst />
<indnrh>hbg</indnrh>
<artidh>M9999-904031</artidh>
<mvkl>235500</mvkl>
<mvbet />
<dvrnr>hbg090318</dvrnr>
<fpantnr>0</fpantnr>
<mvmax />
<andanv />
<andkl>adf</andkl>
<radnr>0</radnr>
</ttUtgodfimvh>
5. Conclusion

5.1 Summary of thesis work

Build-in test (BIT) plays a pretty important role in modern complex military equipment and systems. This diploma work has implemented the prototype about how the diagnosis and information from the BIT can be used in Lift- försvarsmaktens ledningssystem for technical service. Additionally, BIT can also be used for both the analysis of errors and the assessment when appropriate maintenance should be done.

Based on the prototype of this thesis work flow, defense system can extract and transfer BIT error information to central Lift system, analyze information from BIT system in order to repair errors and maintain equipments. The prototype works based on the military equipments systems Visby and Archer.

In a nutshell, this diploma work builds a SOA based collection of information from the BIT equipments system to the central IT system - Lift. It covers the chain from the interface of BIT system, collection of autonomous local Lift-computer, transformation of BIT information to common database format adapted for the maintenance analysis, replication to the central Lift-computer and access to the analytics.

5.2 Lessons learnt when conducting work

During the implementation of the thesis work, I have learnt a lot enterprise business technology and put the theory into practice, which offer a great opportunity for me to learn how enterprise data integration works and how to choose and use right tools to work in a more efficient way, especially for large enterprise system project.

What’s more, I have also learnt the new technology which I have not touched before – 4GL language, which is quite different with the normal SQL language and used in progress OpenEdge relational database system.

It teaches me how to think in an industry based way for a project other than only stay on just finish a project.

5.3 Future research

This chapter describes possible issues emerged from this research which deserves further investigation to productify BIT system.
* Realize proposed platform – Productify prototype and architectural description into a concrete business solution. This can be followed by an analysis on additional usage metrics.

* Handle zip-files – In reality, the military equipments side maybe send the zip-file format information via email, so we encounter the problem about how we can handle it based on prototype, that is, how to configure Email Adapter and Sonic ESB to meet this need. Because right now the prototype only works for simple format files, e.g. txt or xml file. Furthermore, configure Email Adapter to handle multiple emails; since every time the new coming email will overwrite the previous one on the Sonic ESB process, that is, we can always see the latest information. Just in case if we want to keep track of previous email report on Sonic ESB process, we need a concrete solution followed by an analysis on additional requirements.

* Completely automatic – implement a industry based totally automatically BIT system, since one part of this thesis work leads the project not “completely” automatic, that is, when extract information from email based on Email Adapter, we need to File Drop to hard disk and run Java code to convert the file, it would be best if all of the operations are handled.
6. Glossary

FMV – Försvarets materielverk (Swedish Defence)

LIFT – Lednings- och Informationssystem för Förnödenhetsförsörjning och Teknisk tjänst

DXSI – Progress DataXtend Semantic Integration tool, use for Data Integration

ESB – Enterprise Service Bus. A software architecture construct, implemented by technologies found in a category of middleware infrastructure products. ESBs are usually based on standards and provide foundational services for more complex architectures via an event-driven and standards-based messaging engine (the bus). A standards-based integration platform that combines messaging, web services, data transformation and intelligent routing in an event-driven service-oriented architecture (SOA). An ESB connects and mediates all communications and interactions between services.

Data integration – The process of combining data residing at different sources and providing the user with a unified view of these data.

Content-based routing (CBR) – service that can be configured to listen on one address and using internal logic, routes messages to one or more addresses (ESB processes, services, or endpoints). Typically, a CBR service routes messages according to complex rules. The rule specification is either a routing rules file (.cbr), an XPath routing rules file (.xcbr), or a JavaScript rules file (.js). A CBR service typically does not modify message content, but can do so, if required.

ESB process – Lists the sequence of services, nested ESB processes, and endpoints that a message passes through. (This sequence is often called an itinerary.) Messages entering a Sonic ESB process are evaluated and given a process context which typically contains routing information. ESB processes can span multiple containers and multiple hosts, and provide QoS, error handling, and message tracking across all contained services and nested ESB processes.

Sonic Management Console (SMC) – A graphical management tool to configure, monitor and manage SonicMQ and Sonic ESB deployments. Administrators typically use the Sonic Management Console to configure containers, services, endpoints and connections, as well as runtime management. The Sonic Management Console hosts the ESB Configured Objects section, used for Sonic ESB administration, mainly configuring the Directory Service to recognize new types of services, endpoints and connections.
Transformation service – A direct interface for an ESB container to access the resources necessary to transform a message to potentially many predefined formats. The transformation service applies an XSLT stylesheet to the XML in the body of a message, transforming the content of the message from one format to another.
7. References


FMV. Lift och Du-Web. FMV. [Online] [Cited: 06 01, 2009.]

Help Content of DataXtend SI. s.l. : Progress Software.

JMS Messaging with SonicMQ. Progress Web-based Training. [Online] [Cited: 03 28, 2009.]

OpenEdge Architect. Progress Web-based Training. [Online] [Cited: 03 22, 2009.]

Service Oriented Integration with Sonic ESB. Progress Web-based Training. [Online] [Cited: 03 07, 2009.]
8. Appendix

FileConvert.java

MultiPartSend.java

MultiPartReceive.java

Monitor.java

Visby.xsl

Archer.xsl

FileDrop.drop

LiftLRSF.xsd