Database and Message Interoperability Using the eXtensible Markup Language (XML)

December, 2000

Paul C. Barr
<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>DEC 2000</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>00-12-2000 to 00-12-2000</td>
</tr>
<tr>
<td>4. TITLE AND SUBTITLE</td>
<td>Database and Message Interoperability Using the eXtensible Markup Language (XML)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5a. CONTRACT NUMBER</td>
<td></td>
<td>5b. GRANT NUMBER</td>
<td></td>
</tr>
<tr>
<td>5c. PROGRAM ELEMENT NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5d. PROJECT NUMBER</td>
<td></td>
<td>5e. TASK NUMBER</td>
<td></td>
</tr>
<tr>
<td>5f. WORK UNIT NUMBER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. AUTHOR(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MITRE Corporation, 7515 Colshire Drive, McLean, VA, 22102-7539</td>
<td>8. PERFORMING ORGANIZATION REPORT NUMBER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</td>
<td>10. SPONSOR/MONITOR’S ACRONYM(S)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. DISTRIBUTION/AVAILABILITY STATEMENT</td>
<td>Approved for public release; distribution unlimited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. SUPPLEMENTARY NOTES</td>
<td>The original document contains color images.</td>
<td>14. ABSTRACT</td>
<td></td>
</tr>
<tr>
<td>15. SUBJECT TERMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. SECURITY CLASSIFICATION OF:</td>
<td></td>
<td>17. LIMITATION OF ABSTRACT</td>
<td>18. NUMBER OF PAGES</td>
</tr>
<tr>
<td>a. REPORT</td>
<td>unclassified</td>
<td>c. THIS PAGE</td>
<td>unclassified</td>
</tr>
<tr>
<td>b. ABSTRACT</td>
<td>unclassified</td>
<td>19a. NAME OF RESPONSIBLE PERSON</td>
<td></td>
</tr>
</tbody>
</table>

The original document contains color images.
MITRE Department
and Project Approval:

Beth T. Meinert
Department Manager
C2 Systems Architecture & Integration
Abstract

The Department of Defense (DoD) is facing significant interoperability issues within the Services as they seek to architect solutions for distributed systems composed of clients and servers of heterogeneous hosts to enable joint service operations. The eXtensible Markup Language (XML) and related technologies offer promise for applying data management technology to documents and, also, for providing a neutral syntax for interoperability among disparate systems. This paper provides the description of the work performed for the Army Systems Engineering Office (ASEO) and the results of applying XML and other leading edge software technologies to enable interoperability among dissimilar databases and message formats.

The solution approach taken investigated the utility of the XML to write translator(s) between message formats and databases to reduce operator burden and to increase interoperability. Further, we investigated other leading-edge software technologies to address the “information portability” issues associated with distributed systems; i.e., JAVA. The demonstration implemented a United States Marine Corps (USMC) message scenario of battlefield engagement to interoperate with the United States Army (USA) via automated message translation and database interoperability.

The experimental distributed architecture is provided, as well as the findings of the proof of concept demonstration.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Background</td>
<td>1-1</td>
</tr>
<tr>
<td>2. <strong>Proof of Concept Architecture Implementation</strong></td>
<td>2-1</td>
</tr>
<tr>
<td>2.1 Overarching Issue and Objective</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2 Problem Statement Summarized</td>
<td>2-1</td>
</tr>
<tr>
<td>2.3 Solution Approach</td>
<td>2-2</td>
</tr>
<tr>
<td>2.4 Logical View of Tasks</td>
<td>2-2</td>
</tr>
<tr>
<td>2.4.1 Database-to-Database Interoperability</td>
<td>2-2</td>
</tr>
<tr>
<td>2.4.2 Message-to-Message Interoperability</td>
<td>2-3</td>
</tr>
<tr>
<td>2.5 Physical Hardware View</td>
<td>2-5</td>
</tr>
<tr>
<td>2.6 Computers and Software—Physical Representation</td>
<td>2-5</td>
</tr>
<tr>
<td>2.7 System Architecture</td>
<td>2-8</td>
</tr>
<tr>
<td>2.7.1 Use Cases</td>
<td>2-8</td>
</tr>
<tr>
<td>2.7.2 Database-to-Database Interaction Software</td>
<td>2-9</td>
</tr>
<tr>
<td>2.7.3 Messaging Interaction Software</td>
<td>2-11</td>
</tr>
<tr>
<td>2.7.4 Systems Architecture (Battlefield Engagement Scenario)</td>
<td>2-17</td>
</tr>
<tr>
<td>3. <strong>Summary Methodology/Architectural Solution</strong></td>
<td>3-1</td>
</tr>
<tr>
<td>3.1 Future Directions Using XML</td>
<td>3-1</td>
</tr>
<tr>
<td><strong>Appendix. University of Connecticut Project Report</strong></td>
<td>A-1</td>
</tr>
<tr>
<td><strong>Glossary</strong></td>
<td>GL-1</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1. Database-to-Database Interoperability Using XML</td>
<td>2-3</td>
</tr>
<tr>
<td>Figure 2-2. USMTF/JVMF Message Translation Interoperability Using XML</td>
<td>2-4</td>
</tr>
<tr>
<td>Figure 2-3. Solution Approach</td>
<td>2-4</td>
</tr>
<tr>
<td>Figure 2-4. Physical Hardware View</td>
<td>2-5</td>
</tr>
<tr>
<td>Figure 2-5. 1A</td>
<td>2-6</td>
</tr>
<tr>
<td>Figure 2-6. 1B</td>
<td>2-7</td>
</tr>
<tr>
<td>Figure 2-7. 1C</td>
<td>2-7</td>
</tr>
<tr>
<td>Figure 2-8. 1D</td>
<td>2-8</td>
</tr>
<tr>
<td>Figure 2-9. Use Diagram</td>
<td>2-9</td>
</tr>
<tr>
<td>Figure 2-10. UML Sequence Diagram</td>
<td>2-10</td>
</tr>
<tr>
<td>Figure 2-11. UML Sequence Diagram Showing Field Frame</td>
<td>2-11</td>
</tr>
<tr>
<td>Figure 2-12. Message Graphical User Interface</td>
<td>2-12</td>
</tr>
<tr>
<td>Figure 2-13. Message Interaction Sequence Diagram</td>
<td>2-13</td>
</tr>
<tr>
<td>Figure 2-14. Message Call Interaction Sequence Diagram (1)</td>
<td>2-13</td>
</tr>
<tr>
<td>Figure 2-15. Message Call Interaction Sequence Diagram (2)</td>
<td>2-14</td>
</tr>
<tr>
<td>Figure 2-16. Message Call Sequence (1)</td>
<td>2-15</td>
</tr>
</tbody>
</table>
Figure 2-17. Message Call Sequence (2) 2-16
Figure 2-18. Initialization Process 2-16
Figure 2-19. “Proof of Concept” System’s Architecture 2-17
Section 1

Background

MITRE and the Army Systems Engineering Office (ASEO) representatives participated in a United States Army (USA) and United States Marine Corps (USMC) interoperability meeting on 4 April 2000. The USMC, as well as the other services, are facing significant interoperability problems as they seek to architect solutions for distributed systems composed of clients and servers of all types of operating system/hardware combinations. Previous solutions involving Defense Information Infrastructure (DII) Common Operating Environment (COE) has proved to be inadequate, for a number of reasons, including:

a. The loss of synchronization due to intermittent communications,
b. The inability to promote database synchronization across multiple platforms, and
c. The need to retransmit entire databases wasting valuable bandwidth in the event of failure.

From a practical basis, they are facing concrete problems such as:

a. Low and limited bandwidth between the nodes that comprise the distributed system,
b. Inflexibility of client/server solutions when dynamic changes occur (i.e., to allow a client to switch servers during the course of a battle), and
c. The ability to handle mobile or hand-held autonomous computing units.

As a result of these and other difficulties, there has been the realization that the USMC has a rather exhaustive list of requirements that must be met in order to successfully architect a distributed system with inter-operating components:

1. Allow clients to dynamically determine their “best” server, which may change over time.
2. Allow clients to perform database processing while “disconnected” from the server and upon reconnection, to permit any updates/changes to be reconciled.
3. Provide a software architectural solution that is dynamically configurable or tailor-able at all levels of the command hierarchy, based on changing system needs/requirements.
4. Address issues related to minimizing data flow over low bandwidth connections, and offer models that support different computation models (push, pull, publish/subscribe, etc.)
Overall, the resulting distributed architecture must be able to support large numbers of clients and be able to track and reconcile database information stored in different semantic formats in different locations across a myriad of operating systems/hardware platforms.

The interoperability problems faced by USMC are typical of those being addressed by government, industry, and academia nationwide. The problems and issues are in the areas of software architecture and database interoperability for distributed systems. There has been extensive work in architectural description languages (ADL) and modeling languages (UML) as a means to specify distributed applications. Systems realized via an ADL can then attempt to leverage middleware (Common Object Request Broker [CORBA], Java Internetworking Initiative [JINI], eXtensible Markup Language [XML], etc.) for implementation purposes. Likewise, database interoperability techniques have become very standardized in the past five years, exploiting middleware and open standards; i.e., Open DataBase Connectivity (ODBC), as a means to allow different databases to interact. However, what has been lacking is the ability to merge software architectural approaches and database interoperability techniques into a synergistic approach that offers a unified meta-model to specify distributed applications. Such a meta-model would be all encompassing, to allow software architectural, hardware, and database requirements to be specified and, once specified, to facilitate the transition to a working distributed system.

In the long-term, the task is to develop an integrated, unified meta-model that is able to support the software architectural specification and database interoperability requirements of a distributed system. In the short term, as reflected in the task for fiscal year 2000, a “proof of concept” demonstration was developed to explore issues related to database interoperability, as well as message translation. In support of this effort of exploring the translation between message formats; i.e., United States Message Text Format (USMTF) and Joint Variable Message Format (JVMF), a client using one format (USMTF) seeks to interact with a server or database using another format (JVMF). A “proof of concept” architecture was designed to demonstrate database-to-database transfer and message interactions. The work was conducted on a series of three workstations (two WindowsNT hosts and one Linux host) with the ability to plug-in a laptop to simulate a client that arrives/departs. This short-term effort, in fact, is the basis for the longer-term goal of a meta-model that supports the software architectural specification and database interoperability requirements of a distributed system.
Section 2

Proof of Concept Architecture Implementation

The description of the work performed for the Army Systems Engineering Office (ASEO) for the “proof of concept” architecture implementation is broken into seven sections:

a. Overarching Issue and Objective
b. Problem Statement Summarized
c. Solution Approach
d. Logical View of Tasks
e. Physical Hardware View
f. Computers and Software - Physical Representation
g. System Architecture

A description of the work performed for each of the sections will be provided.

2.1 Overarching Issue and Objective

The United States Services are facing significant interoperability problems as they seek to develop solutions for distributed systems composed of clients and servers of all types of operating systems, software, and hardware combinations.

The high level objective of the task was to investigate the utility of XML and other leading edge software technologies to facilitate the seamless flow of information and interoperability between the Services.

2.2 Problem Statement Summarized

On 4 April 2000, MITRE representatives and ASEO representatives participated in an USA and USMC interoperability issues meeting. From that meeting, the following issue was revealed: The USMC routinely inter-operates with both the U.S. Navy (USN) and the USA. The Army uses the Joint Common Data Base (JCDB), which is implemented using the Commercial-off-the-Shelf (COTS) product from Informix while the Marines use the USN’s Tactical Database Manager (TDBM). Hence, there are database-to-database interoperability issues.
The USMC communicates with the USA via JVMF messages and interacts with the USA’s JCDB. The USMC communicates with the USN via USMTF messages and interacts with the USN’s TDBM.

This requires the USMC operators to translate between dissimilar message formats and databases, which increases the operator burden.

2.3 Solution Approach

The solution approach taken investigated the utility of the XML to write translator(s) between message formats and databases to reduce operator burden and increase interoperability. Further, we investigated other leading-edge software technologies to address the “information portability” issues associated with distributed systems; i.e., JAVA.

Our approach, also, designed and implemented an USMC message based battlefield engagement scenario to illustrate the utility of XML and Java to promote automated message translation and database interoperability between the Services.

2.4 Logical View of Tasks

The logical view of tasks is divided into two related views:

a. Database-to-Database Interoperability (Army's JCDB and USN’s TDBM )


2.4.1 Database-to-Database Interoperability

As depicted in Figure 2-1, a Java Object Translator extracts (by SQL statements - reads) from an Oracle database pertinent data and translates the data into XML. As shown, XML is input to a second Java Object Translator (via the dotted line is on the same LAN), which translates the XML to inject into the Access database (by SQL statements - writes). The functionality is SQL to XML extract, XML to SQL inject using Java Objects Translators (Remote Method Invocation [RMI] Registry operation).
2.4.2 Message-to-Message Interoperability

Referring to Figure 2-2, the XML from the prior database-to-database translation (Figure 2-1) becomes the input to a third Java Object translator and, depending on the destination, inter-operating with the Army field units, or the USMC field units, a USMTF or JVMF message is generated. At the receiving end the message is translated to XML via another Java Object translator and serves as an input to the database-to-database interoperability square as depicted in Figure 2-3. Figure 2-3 represents a composite of the two previous diagrams to illustrate the complete solution approach.
Figure 2-2. USMTF/JVMF Message Translation Interoperability Using XML

Figure 2-3. Solution Approach
2.5 Physical Hardware View
As shown in Figure 2-4, the hardware that was used to provide the “proof of concept,” which investigated the utility of the XML to write translator(s) between message formats and databases to reduce operator burden and increase interoperability, involved two hosts running the Windows NT operating system and a third host machine running Linux operating system connected to a hub. The Linux machine was provided with a double local area network (LAN) card to provide connectivity in two LAN networks. The fourth machine, a laptop, was connected at a later time to demonstrate the database synchronization ability.

![Figure 2-4. Physical Hardware View](image)

2.6 Computers and Software—Physical Representation
This section will detail the mapping of the software onto the hardware. As depicted in Figures 2-5, 2-6, 2-7, and 2-8, each machine has a database. To demonstrate interoperability between dissimilar databases, the Windows NT hosts contained Microsoft Access databases and the Linux host used the Oracle database.

The systems architecture is that of client-server using the RMI Registry. Hence, each of the hosts contains software to implement the RMI Registry. As depicted in the figures, there are three databases: the Army database, the Marine database, and the Enemy database. As the
battlefield engagement scenario is executed, the unit changes position (longitude, latitude, direction, speed); and the event is recorded in the respective database and communicated to the other two databases. Hence, each host is also a client of the database server. The field clients only understand USMTF for the Army or JVMF for the Marines. The Linux host contains an additional software module called TimeServer. This module compresses the simulation time of the battlefield engagement scenario.

Figure 2-5. 1A
Figure 2-6. 1B

Figure 2-7. 1C
2.7 System Architecture

2.7.1 Use Cases

As depicted in the Figure 2-9, there are eight actors:

1. One prototype user actor to initialize the prototype,
2. One timer server actor to update the battlefield engagement simulation,
3. Four Database Actors (Map dB, Army dB, Marine dB, and Enemy dB), and
4. Two Field Actors (Marine Field and Army Field).

Each user sends data and/or receives data from the Databases and sends the data to the Field.
2.7.2 Database-to-Database Interaction Software

The database-to-database “proof of concept” software is broken into three parts. The three parts include a database access module, a message translation module, and a mapping client module. The database access module provides access to the Marine and Army databases by using Java DataBase Connectivity (JDBC) and SQL to get and put information into the databases. The information obtained from the databases is passed as a message object to the message translation module, which can take the message object and produce a representation of the message in XML, JVMF, or USMTF format. Finally, the mapping client tier provides a graphical interface for the user where the message information can be displayed as battlefield data. The data in the databases is inserted and updated by a set of data generating classes.
A database server object is created for the Marine, Army, and enemy databases. The database server gets information from the database by creating a remote object that reads information from the database every two seconds using SQL queries. The database server’s graphical interface then queries this remote object every two seconds for its latest messages, receives a list of message objects, and then displays the messages on the screen in either the USMTF or JVMF format.

A set of database clients uses the enemy database server’s remote object and the other friendly database server’s remote object to get the latest messages available every two seconds. Then they insert that information into their particular database. This part of the software would be the message translation tier that takes XML messages from the database server and changes them into the corresponding message format that its client needs, either USMTF or JVMF. Currently, it takes a message object from the database server and updates its database with the fields in this object. This message object can be formatted to an XML, USMTF, or JVMF message.

The Army and Marine threads are similar:

- The marine thread gathers data from the enemy and army databases, inserts data into its own database and then displays it for the user.
- The m_enemyDBServerRemote and m_otherDBServerRemote objects provide the DBClient with the latest messages from their database (army/marine).

**Figure 2-10. UML Sequence Diagram**
Like the database server, the field server gets information from the database by creating a remote object that reads information from the database every two seconds, using the same methods as the database server remote object. The field server’s graphical interface then queries this remote object every two seconds for its latest messages and then displays the messages on the screen. The field clients use the same field server’s remote object to get the latest messages available every two seconds from the database and display their information on the map.

![UML Sequence Diagram Showing Field Frame](image)

**Figure 2-11.** UML Sequence Diagram Showing Field Frame

### 2.7.3 Messaging Interaction Software

There are two programs for the message interaction part of the task. The marine program allows users to input the following types of JVMF messages: K01.01, K05.01, K05.13, K05.14, K05.15. Once the user has input a message, it can be converted to XML and sent to another host machine, or converted back to JVMF. The Army program provides the same capabilities for USMTF messages, A423 (Order) and C400 (Sitrep).

In the example below, Figure 2-12, the user selects the message type from a drop-down list. According to the message type, a form is shown with the appropriate fields. In this case, a free text message has been chosen. Once the user fills in the fields, he can hit the
convert>>XML button; and the resulting XML data will be displayed on the XML window on the top right. Figure 2-3 represents the process that takes place when the convert>>XML button is hit.

**Figure 2-12. Message Graphical User Interface**
Object1 initial JvmfPanel
if(currentMessage.isCompleted())
else
xmlPanel
XmlPanel currentMessagePanel
MessagePanel xmlArea
JTextArea currentMessage
Message static
XmlUtil static
JOptionPane

if(currentMessage.isCompleted())
else
setText(xml):void
showMessageDialog(this, "Sorry, some mandatory fields are not completed!", "Error", JOptionPane.ERROR_MESSAGE):void
setXmlString(XmlUtil.xmlToString(currentMessage.toXmlDocument()):void)
'txmlToString(XmlDocument):String'(currentMessage.toXmlDocument())
toXmlDocument():XmlDocument

User pressed convert>>XML Button

Figure 2-13. Message Interaction Sequence Diagram

Note that in the diagram, Figure 2-13, a call is made to message in order to convert the data in the message to an XML document. The process that takes place in this call is provided in the Figure 2-14.

Figure 2-14. Message Call Interaction Sequence Diagram (1)
Once the data appears as XML on the top right window, the user may click the convert>>JVMF button, which will display the data the user input or any data received in the JVMF format.

Figure 2-15. Message Call Interaction Sequence Diagram (2)

Note that this sequence, as shown in Figure 2-16, makes use of a static XmlUtil class to convert an XML message back to a message object. This is the sequence diagram for the XmlDocument To Message() call.
Finally, the user may send the message to another computer running the same program, which would result in the following sequence.
Figure 2-17. Message Call Sequence (2)

This diagram, Figure 2-17, shows that the program will take the IP address given and send the current message displayed to that machine on port 4444. If the same program is running on that machine, it can receive this message and display it.

The initialization Sequence diagram of Figure 2-18 is provided to describe the progression of the initialization process.

Figure 2-18. Initialization Process
2.7.4 Systems Architecture (Battlefield Engagement Scenario)

The battlefield engagement block diagram, Figure 2-19, provides the view of the software modules, objects, servers and clients, and their relationship with each other. In each case, there is initialization software that clears out the databases prior to the next simulation run, brings up the servers, and starts the clients.

As depicted in the above architecture, Figure 2-19, the engagement is initiated by one, two, or three insertions into the Army Database, the Marine Database, and/or the Enemy Database. To follow the sequence, the movement of enemy troops is inserted in the Enemy’s Database. The alert causes the Object translator to extract the information with an XML extraction event. In turn, the same object sends the XML to the Marine Client Object and to the Army Client Object, using the RMI Registry discovery mechanism. Both, in turn, inject this information into their respective Databases, using the XML transformation mechanism.
Information is extracted from the Databases using XML/SQL and sent to the Field Servers: who, in turn, push information to the Field; i.e., USMTF to the Army Field and JVMF to the Marine Field.
Section 3

Summary Methodology/Architectural Solution

This working note documents the methodology of the “proof of concept” software development effort, which provides a possible solution to the United States Inter-Services need for message and database interoperability to reduce operator burden.

The task investigated the utility of the XML to write translator(s) between message formats and databases to reduce operator burden and to increase interoperability. The results indicated that XML has great potential to improve message and database interoperability. Further, the investigation of other leading-edge software technologies to address the “information portability” issues associated with distributed systems (i.e., JAVA) gave positive results.

The battlefield engagement simulation described the overall software architectural structure and interactions, as well as detailing the steps for achieving database operability when a new database was added (Laptop Host). It provided the “proof of concept” of database interoperability, as well as message translation using XML.

3.1 Future Directions Using XML

The application of XML to the United States Services over the next few years has great potential to improve information exchange and interoperability. Major software companies – such as Microsoft, Sun, IBM, Sybase, Informix, and Oracle – are integrating XML within their products. XML is a powerful method of structuring data in a text or ASCII file. XML files can handle data/text information, as well as pointers to binary files. As such, XML shows great promise for facilitating information exchange in distributed systems.
# Table Of Contents

## Root Package

Class Diagrams

### Class Diagram <default>

#### Class Node Detail

- Class `army`
- Class `ArmyDBclient`
- Class `ArmyDBserver`
- Class `ArmyFieldServer`
- Class `DBClient`
- Class `DBServer`
- Class `DBServerRemoteImpl`
- Class `DE`
- Class `DefaultWindowHandler`
- Class `Distrib`
- Class `EnemyDBserver`
- Class `Erase`
- Class `FieldFrame`
- Class `FieldServer`
- Class `FieldServerRemoteImpl`
- Class `Force`
- Class `Inserter`
- Class `JVMF`
- Class `marine`
Class MarineDBclient ..............................................................................................................
Class MarineDBserver ...........................................................................................................
Class MarineFieldServer ......................................................................................................
Class Message ....................................................................................................................... 
Class MessageFrame ............................................................................................................... 
Class MessageTransformation ................................................................................................ 
Class mitre.JavaObject .......................................................................................................... 
Class mitre.MitreMain ...........................................................................................................
Class MovingLayer ............................................................................................................... 
Class State ...........................................................................................................................
Class TimeRemoteImpl .......................................................................................................... 
Class USMTF .......................................................................................................................... 
Class X .....................................................................................................................................
Class XML ..............................................................................................................................

Interface Node Detail ..............................................................................................................
Interface Const ....................................................................................................................... 
Interface DBServerRemote ....................................................................................................
Interface FieldServerRemote ............................................................................................... 
Interface Position ................................................................................................................ 
Interface TimeRemote ........................................................................................................... 

Class Detail ............................................................................................................................
Class army .............................................................................................................................

Operation Detail ...................................................................................................................
main ........................................................................................................................................ 

Class ArmyDBclient ............................................................................................................... 

Operation Detail ...................................................................................................................
main ........................................................................................................................................ 

Class ArmyDBserver .............................................................................................................

A-3
Operation Detail.................................................................................................................................
main......................................................................................................................................................

Class *ArmyFieldServer* ......................................................................................................................
Operation Detail.................................................................................................................................
main......................................................................................................................................................

Class *DBClient* ......................................................................................................................................
Attribute Detail......................................................................................................................................
m_connection....................................................................................................................................... m_container.................................................................................................................................
m_currentTimeLabel......................................................................................................................... m_enemyDBserverRemote........................................................................................................ m_gbConstraints .........................................................................................................................

m_gbLayout........................................................................................................................................ m_leftLowerTextArea...................................................................................................................... m_leftUpperTextArea ................................................................................................................ m_otherDBserverRemote........................................................................................................ m_rightLowerTextArea................................................................................................................ m_rightUpperTextArea .............................................................................................................. m_timeRemote........................................................................................................................ m_which......................................................................................................................................

TIME_LABEL.........................................................................................................................................

Constructor Detail...............................................................................................................................
DBClient................................................................................................................................................

Operation Detail.................................................................................................................................
addComponent...................................................................................................................................... makeWidgets.........................................................................................................................

Class *DBServer* ....................................................................................................................................
Attribute Detail.................................................................................................................................
m_container....................................................................................................................................... m_currentTimeLabel..............................................................................................................
m_dbServerRemoteImpl........................................................................................................................
m_gbConstraints...........................................................................................................................................
m_gbLayout..................................................................................................................................................
m_leftTextArea...........................................................................................................................................
m_rightTextArea........................................................................................................................................
m_which......................................................................................................................................................
TIME_LABEL..................................................................................................................................................

Constructor Detail..........................................................................................................................................
DBServer.....................................................................................................................................................

Operation Detail.......................................................................................................................................... 
addComponent............................................................................................................................................
makeWidgets ................................................................................................................................................
updateText ...................................................................................................................................................

Class DBServerRemoteImpl..........................................................................................................................

Attribute Detail...............................................................................................................................................
for_sync .....................................................................................................................................................
m_connection............................................................................................................................................
m_recentMessage.....................................................................................................................................
m_time....................................................................................................................................................
m_timeRemote.......................................................................................................................................... 
m_which ...................................................................................................................................................

Constructor Detail..........................................................................................................................................
DBServerRemoteImpl.................................................................................................................................

Operation Detail.......................................................................................................................................... 
getRecentMessage.....................................................................................................................................
getTime.....................................................................................................................................................

Class DE.....................................................................................................................................................

Attribute Detail...............................................................................................................................................
dstConnection .............................................................................................................................................
srcConnection.............................................................................................................................................
Constructor Detail

DE

Operation Detail

main

transfer_data

Class DefaultWindowHandler

Operation Detail

windowClosing

Class Distrib

Attribute Detail

dstConnection

srcConnection

Constructor Detail

Distrib

Operation Detail

main

transfer_data

Class EnemyDBserver

Operation Detail

main

Class Erase

Constructor Detail

Erase

Operation Detail

main

Class FieldFrame

Attribute Detail

aMapLegend

forces

informationArea

latitudeLabel
longitudeLabel
mapViewer
messageArea
messageFrame
movingLayer
m_container
m_currentTime
m_currentTimeLabel
m_fieldServerRemote
m_gbConstraints
m_gbLayout
m_isRunning
m_messages
m_timeRemote
m_which
sxMapViewer
ZOOM_FACTOR

Constructor Detail
FieldFrame

Operation Detail
addComponent
addComponent
addMovingLayer
initMapViewer
makeBottom
makeLeftWidgets
makeText
makeWidgets
updateForces

Class FieldServer

Attribute Detail

m_container
m_currentTimeLabel
m_fieldServerRemoteImpl
m_gbConstraints
m_gbLayout
m_textArea
m_which
TIME_LABEL

Constructor Detail
FieldServer

Operation Detail
addComponent
makeWidgets
updateText

Class FieldServerRemoteImpl

Attribute Detail

for_sync
m_aliveId
m_connection
m_deadId
m_recentMessage
m_time
m_timeRemote

Constructor Detail
FieldServerRemoteImpl

Operation Detail
getAliveId
getDeadId

getRecentMessage

g getTime

Class `Force`

Attribute Detail

alive

appear_time

battalion

BN_AR_ENEMY

BN_AR_FRIEND

BN_IN_ENEMY

BN_IN_FRIEND

BN_MECH_ENEMY

BN_MECH_FRIEND

brigade

category

direction

disappear_time

id

IMAGE_DIRECTORY

in_battle

latitude

LIMBO_TIME

longitude

m_image

m_limboTime

old_battalion

old_brigade
position
side
speed
time

Constructor Detail
Force
Force

Operation Detail
decreaseLimboTime
getImage
getLimboTime
getPosition
identityChanged
setPosition
updateInformation

Class Inserter

Attribute Detail
connection
mapViewer
sxMapViewer
THREE
TOTAL_ID_NUM
TOTAL_TIMES

Constructor Detail
Inserter

Operation Detail
find_direction
find_speed
getState
Class JVMF

Attribute Detail
- alive
- appear_time
- battalion
- brigade
- category
- direction
- disappear_time
- id
- in_battle
- latitude
- longitude
- side
- speed
- time

Constructor Detail
- JVMF
- JVMF
- JVMF

Operation Detail
- toMessage
- toString
- toXML

Class marine