Implementing RosettaNet
Business-to-Business Integration
Using J2EE and Web Services

Juho Tikkala
November 29 2004
Part 1 — The Thesis
Background

• E-business, connecting information systems of partnering organizations using Internet-based technologies, has become an essential part of today’s business environment (Rodgers et al. 2002)
  – Significant advantages can be obtained by automating inter-organizational business processes, because of increased speed, efficiency, and reliability (Shim et al. 2000, Rodgers et al. 2002)
• To achieve this, *business-to-business integration* must be performed (Medjahed et al. 2003)
• *Business-to-business integration frameworks* are generic solutions that enable performing such integration. XML-based business-to-business integration frameworks (*e-business frameworks*) employ XML technologies and the Internet to provide this functionality. (Shim et al. 2000)
Background

- RosettaNet is an industry consortium that aims to create e-business process standards for high-tech industries (RosettaNet 2004)
  - Information technology, electronic components, semiconductor manufacturing, telecommunications, and logistics
  - These standards form the RosettaNet e-business framework
  - Dictionaries define common sets of XML elements
  - Partner Interface Processes (PIPs) define business processes as XML documents and related messaging choreography
  - RosettaNet Implementation Framework (RNIF) is the messaging portion of the RosettaNet e-business framework
Background

• There seems to be few scientifically documented experience reports on implementing e-business frameworks with current tools described from a technical point of view (Nurmilaakso 2003)
• The purpose of this work is to produce such experience on implementing RNIF by constructing a prototype system
• The prototype has two high-level goals
  – Create a learning platform for students of Information and Communication Technology enabled commerce at Helsinki University of Technology Software Business and Engineering Institute (SoberIT)
  – Create an RNIF implementation as a generic software component, providing services that are assumed useful in a business-to-business integration scenario, to gather and report experience on implementing such systems
• The work was conducted at SoberIT between December 2003 and May 2004
• The problem definition is based on previous work conducted at SoberIT in context of the NetData project (2002–2004)
• This work was conducted in partnership with IBM
Research Problem and Objectives

- The research problem can be formulated with following questions
  1. How can RNIF be implemented in enterprise software systems?
  2. What are the main issues in implementing RNIF with software development tools commonly available in early 2004? How suitable are the tools, what could be improved in them? How much effort is required?
  3. What level of performance can be obtained, in terms of message throughput and maximum message size?
  4. Are there any interoperability problems with RNIF implementations?

- Additional goals set for the prototype system are, in order of decreasing priority:
  - Compliance with the RNIF 2.0 specification
  - Ease of use for developer of applications employing the prototype
  - Portability, the prototype should be usable in as wide variety of hardware/software environments as allowed by the selected implementation tools, with minimal configuration required
  - Scalability, level of performance that can be gained, should be maximized to degree possible within limits of available implementation resources
Research Method

• The research is largely constructive (Kasanen et al. 1993) in nature, centered on building a prototype system
• A general understanding of problem is obtained by first performing a brief literature study
  – Business-to-business integration
  – XML technologies
  – Business-to-business integration frameworks
  – Messaging in an XML-based business-to-business frameworks
  – Java 2 platform Enterprise Edition (J2EE)
  – Implementing business-to-business integration
• The problem is analyzed further and a set of functional and technical requirements for the prototype implementation is derived
• A prototype is constructed by applying typical software design practices
  – IBM WebSphere tools used
• Demonstrating that the solution works is done by applying testing to the prototype
Prototype Overview

Trading Partner A

service application

Trading Partner B

IBM WebSphere

Prototype provides:
- SOAP/RNIF message conversion
- Message validation
- Message routing
- Encryption/decryption
- Digital signatures

Proprietary SOAP 1.1 w/attachments interface, PIP XML documents and routing information

RNIF 2.0 interface
Testing

• RNIF-compliance
  – Basic RNIF-compliance was tested with the RosettaNet Ready™ Self-Test Kit
  – Test scripts for all basic types of PIPs
  – The prototype system is able to communicate with other RNIF-compliant applications to some degree with all PIP types

• Practical usability
  – Practical usability of the prototype system was tested by a student group in the spring 2004 T-86.301 (Project Course on ICT Enabled Commerce) course at HUT
  – The group explored transforming XML-based SAP IDOC documents to RNIF business messages and vice versa. The prototype system provided the RNIF messaging portion of the developed system.
  – The group used the prototype system in “black-box” manner
  – It seems the prototype is reasonably easy to learn, its interface is usable, and the general approach is applicable to at least this type of integration scenario
Testing

• Interoperability
  – Test RNIF interoperability between the prototype system and a commercial product, Microsoft BizTalk Accelerator for RosettaNet version 2.0
  – The prototype and Microsoft BizTalk exchanged messages and it was verified that all validation checks are passed and acknowledgments received
  – Established commercial RNIF implementations might not always follow the specifications to the letter, and there are areas where correct behavior is not clear
  – The prototype managed to communicate with BizTalk after minor modifications

• Maximum message size
  – The prototype application consumes in order of 6–14 times as much system memory as the size of the business message that is being processed

• Message throughput
  – The prototype system can attain consistent message throughput of about 2–4 business messages per second
  – Especially encrypting and digitally signing a message takes comparatively lots of time
Conclusions

• Research questions were answered
  – One possible method of implementing RNIF was presented using regular enterprise software development tools. Practical suitability of the system architecture approach taken is a question that cannot be answered solely based on this work.
  – The tools were in general suitable to the task, various detailed issues from individual tools were found. Total implementation effort in order of a bit less than three person-months.
  – Level of performance was measured during performance testing
  – It is clear that some interoperability problems are to be expected, even when interacting established commercial RNIF implementations

• Main contribution of this work
  – Providing insight on what problems need to be addressed when constructing a software RNIF implementation and showing one possible way to do it
  – Experience of applying related software development tools, J2EE, and especially web services, in context of RNIF implementation was gained
References

Part 2 — The Prototype System
Prototype Overview

Trading Partner A
- service application
- SOAP
- Proprietary SOAP 1.1 w/attachments interface, PIP XML documents and routing information

Trading Partner B
- SOAP
- RNIF 2.0 interface

IBM WebSphere
- prototype
- RNIF
- Prototype provides:
  - SOAP/RNIF message conversion
  - Message validation
  - Message routing
  - Encryption/decryption
  - Digital signatures
Prototype Interface

- A RosettaNet business message consists of:
  - An XML business document
  - Related metadata (message sender, recipient, PIP code, etc.)
  - Optional binary attachments
- Operations provided by the prototype are:
  - Send a business message asynchronously
  - Send a business message synchronously
  - Send a failure notification business message (PIP 0A1)
- The interface is symmetric for bidirectional communication. From the point of view of a service application, it must implement these operations:
  - Receive a business message asynchronously
  - Receive a business message synchronously
  - Receive a failure notification business message (PIP 0A1)
- For each operation, the XML business document, related metadata, and possibly attachments are specified
Prototype Interface

interface
RosettaNetPortType
+submitSynchronous(inout header : BusinessMessageHeader, inout body : xsd:string, inout attachments : xsd:string) : Status
+failureNotification(in header : BusinessMessageHeader, in failureNotification : FailureNotification) : Status

type
BusinessMessageHeader
+sender[1] : EndPointDescriptor
+receiver[1] : EndPointDescriptor
+initiator[0..1] : Initiator
+fromRole[1] : rnif:fromRole
+toRole[1] : rnif:toRole
+toService[1] : rnif:toService
+isSecureTransportRequired[0..1] : xsd:boolean
+messageTrackingID[1] : rnif:messageTrackingID
+BusinessActivityIdentifier[0..1] : rnif:BusinessActivityIdentifier

enum
Status
+OK = OK
+ACK = ACK
+RE = RE

type
FailureNotification
+fromRole[1] : bd:fromRole
+reason[1] : Reason
+toRole[1] : bd:toRole

enum
Reason

type
EndPointDescriptor
+GlobalBusinessIdentifier[0..1] : rnif:GlobalBusinessIdentifier
+UniformResourceLocator[0..1] : UniformResourceLocator

type
Initiator

type
UniformResourceLocator
Prototype Interface

• Using these operations, it is possible to send and receive all valid RosettaNet business messages

• You still have to do something yourself, though
  – The prototype only addresses RNIF communication and metadata: you have to handle XML business documents yourself, e.g. using XSLT
  – No process definition support, not even timeout and retry logic: you have to implement messaging choreography yourself, e.g. using Java code
Prototype Usage

• The prototype system exposes a WSDL descriptor accessible using HTTP
• This URL is entered to a tool, e.g. WebSphere web service creation wizard that creates local RPC stubs used to access the web service according to JAX-RPC specification
  – JAX-RPC is the Java API for XML-based RPC
  – Roughly, a type definition in a web service XML message corresponds to a Java class
  – The web service can be accessed just like regular Java classes
private void send3A4(final GlobalBusinessIdentifier to, final Source xml) throws RemoteException, MalformedFault, ServerFault {
    Role role;
    Service service;

    BusinessMessageHeader header = new BusinessMessageHeader();
    header.setSender(new EndpointDescriptor());
    EndpointDescriptor receiver = new EndpointDescriptor();
    receiver.setGlobalBusinessIdentifier(to);
    header.setReceiver(receiver);
    role = new Role();
    role.setGlobalPartnerRoleClassificationCode(new GlobalPartnerRoleClassificationCode("Buyer"));
    header.setFromRole(role);
    role = new Role();
    role.setGlobalPartnerRoleClassificationCode(new GlobalPartnerRoleClassificationCode("Seller"));
    header.setToRole(role);
    service = new Service();
    service.setGlobalBusinessServiceCode(new GlobalBusinessServiceCode("Buyer Service"));
    header.setFromService(service);
    service = new Service();
    header.setToService(service);
    MessageTrackingID messageTrackingID = new MessageTrackingID();
    messageTrackingID.getInstanceIdentifier(new org.rosettanet.www.InstanceIdentifier("123"));
    header.setMessageTrackingID(messageTrackingID);
    header.setBusinessActivityIdentifier(new BusinessActivityIdentifier("Request Purchase Order"));
    ActionIdentity actionIdentity = new ActionIdentity();
    header.setActionIdentity(actionIdentity);
    ProcessIdentity processIdentity = new ProcessIdentity();
    processIdentity.setVersionIdentifier(new VersionIdentifier("02.02.00"));
    processIdentity.getInstanceIdentifier(new InstanceIdentifier("456"));
    header.setProcessIdentity(processIdentity);
    new RosettaNetServiceLocator().getRosettaNetPort().submitAsynchronous(header, xml, null);
}
Prototype Usage

• With WebSphere, the development procedure is e.g. as follows:
  1. Enter prototype WSDL URL to WebSphere Studio Application Developer (WSAD) to generate web service stub classes
  2. Using WSAD, code your own service logic using Java, XSLT, and other J2EE tools
  3. Deploy your generated service to WebSphere Application Server
  4. Test your service — the prototype has a web-based interface that can be used to e.g. see the messages that are passed