T-86.301 Project Course on ICT Enabled Commerce &
T-86.161 Special Topics in Information Technology for Production II

SAP R/3 integration to RosettaNet processes using Web Service interfaces

Patrik Ajalin
Unai Briongos
Ari Kesäniemi
Outi Tarvainen

Abstract

This study concentrates on how well RosettaNet can be used to integrate a non-SAP partner and a SAP partner. Background information of SAP and RosettaNet is given, and possible mapping is discussed. An implementation prototype is also constructed, to test how the situation could be implemented in practice. It proves that the basic concept of the implementation is correct, although it lacks many of the qualities commonly associated with business integration systems.

The message mapping and prototype implementation between the frameworks are restricted to purchase order messages. The PIP used is PIP 3A4 Request Purchase Order for both purchase order request and confirmation. The IDoc message types used are ORDERS and ORDRSP, which correspond to the aforementioned PIP business messages. We try to generalize the results of the study also to other messages.

The message types of both standards are well documented and understandable. There were no major differences at least in the case of purchase order. However, an IDoc contains much in-house data that a PIP cannot deliver, which the SAP system has to generate according to some mapping tables when receiving an IDoc transformed from a PIP.

Table of Contents

1. Introduction

This document is the final report for courses “T-86.301 Project Course on ICT Enabled Commerce” and “T-86.161 Special Topics in Information Technology for Production II” at Helsinki University of Technology (HUT).

1.1. Background of the Study

SAP R/3 is the most widely used ERP system in the world [Bernroider et. al, 2000]. It is the world's most-used standard business software for client/server computing [SAPfans]. But what if we have a supply chain partner that has not implemented SAP, but still would like to have an e-business solution with his partner who has implemented SAP. How to integrate non-SAP partners to SAP?

RosettaNet is an open e-business standard or as the organization describes itself, the “lingua franca for e-business”. RosettaNet is a non-profit organization that provides a standard e-business language for supply chain partners in an industry-wide environment. Many of the biggest companies in the world have joined the RosettaNet community in order to make RosettaNet a real powerful e-business standard [RN 1].

This study concentrates on how well RosettaNet can be used to integrate the non-SAP partner and the SAP partner. It studies how well do the SAP IDoc message and the RosettaNet business messages match and how easily can the transformation between them be done. An implementation prototype is also constructed, to test how the situation could be implemented in practice. The study basically tells us how well RosettaNet could function as the message deliverer between a SAP partner and a non-SAP partner.

1.2. Research Problem

The research problem as given by the course was to prototype SAP R/3 integration to RosettaNet processes using Web Service interfaces. The research problem was further clarified by the company representative from ICL Invia, Vesa Niininen, and the course staff. The main research problem can be expressed as one major question:

- How can purchase order messaging between SAP IDocs and RosettaNet PIPs be implemented between trading partners?

The main question can be narrowed down into sub-questions:

- How do SAP IDoc ORDERS and RosettaNet PIPs 3A4 map together and what is the estimated effort in doing so?
- What does the technical solution for the integration look like and how demanding is it to implement?

1.3. Methodology of the study

The study is very much of a constructive research nature. A constructive research can be defined to have the following phases: finding a suitable problem of both practical and research interest, obtaining a general understanding of the problem, innovatively constructing a solution, demonstrating that it works, showing the theoretical connections and research contribution of the solution, and examining the applicability of the solution [Kasanen et. al, 1993]. Our methodology follows this definition. First the underlying technologies and standards need to be studied and then the research issues are to be evaluated by implementing a prototype and evaluating it.

1.4. Objectives of the Study

In order to provide answers to the research problems we will have to understand the standards, the contents of the messages and the technologies to be used. Therefore the first step will be to study them thoroughly. When understanding them well enough the implementation of the prototype is planned and executed. Finally an evaluation of all the involved steps is made, based on which an estimate on how easily such an implementation is made and the how to do it. Objectives can be broken down into the following issues:

- Gain understanding of the standards and technologies used.
- Gain understanding of the practical implementations involved.
- Integrating a prototype of the purchase order exchange.
- Generalize empirical results from purchase order processing to general business integration between SAP and RosettaNet.

1.5. Scope of the Study

The message mapping and prototype implementation between the frameworks are restricted to purchase order messages. The PIP 3A4 is used for both purchase order request and purchase order confirmation. The corresponding IDoc message types are ORDERS and ORDRSP. All other messaging related to business processes between companies are outside of the scope of this study. The prototype is not an industrial implementation and is not meant to work or be used industrially anywhere; it is merely a test prototype to evaluate the solution created for the research problem. The information flow with SAP IDocs and RosettaNet PIP business messages is depicted in figure 1.

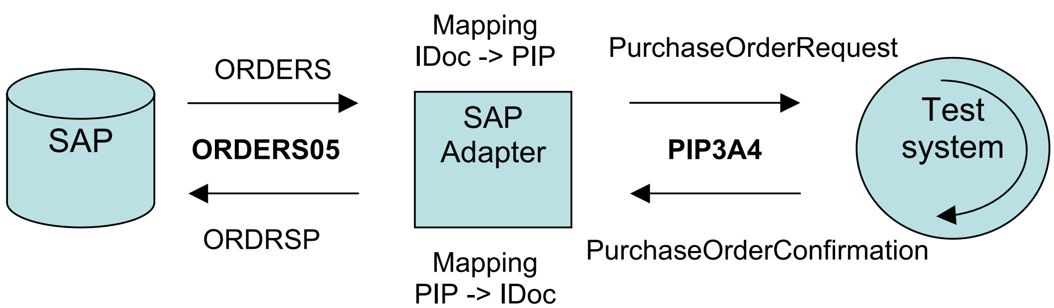

Figure 1 Prototype construction.

2. SAP R/3 IDocs and RosettaNet PIPs

2.1. SAP

Founded in 1972, SAP is the recognized leader in providing collaborative business solutions for all types of industries and for every major market. SAP is the world's largest inter-enterprise software company and the world's third-largest independent software supplier overall. SAP employs over 29,600 people in more than 50 countries. [SAP 1]

2.2. SAP R/3

SAP R/3 is a completely integrated, enterprise wide information system that replaces previous separate information handling systems with a series of software modules that communicate with each other seamlessly. I.e. SAP software is a total information system that weaves together all the data from manufacturing with all the data from inventory with all the data from purchasing with all the data from accounting with all the data from sales and so on. [SAP 4]

R/3 uses the client/server model and provides the ability to store, retrieve, analyze, and process in many ways corporate data for financial analysis, production operation, human resource management, and most other business processes. A recent release of R/3 makes it possible to get to the R/3 database and applications through Internet access and Web browsers. A sales representative can initiate the workflow for a sales order by filling out an electronic form on a laptop that will be "translated" into input for the R/3 system. [SAP 2]

2.2.1. IDocs

Intermediate Document (IDoc) is a standard data structure for electronic data electronic information exchange between any two SAP systems or between a SAP application and an external program. IDocs serve as the vehicle for data transfer in SAP's Application Link Enabling (ALE) system. ALE uses IDocs to exchange data between logical systems. IDocs are used for asynchronous transactions: each IDoc generated exists as a self-contained text file that can then be transmitted to the requesting workstation without connecting to the central database. Another SAP mechanism, the Business Application Programming Interface (BAPI) is used for synchronous transactions. Below in figure 2 is a description of the ALE architecture in SAP.

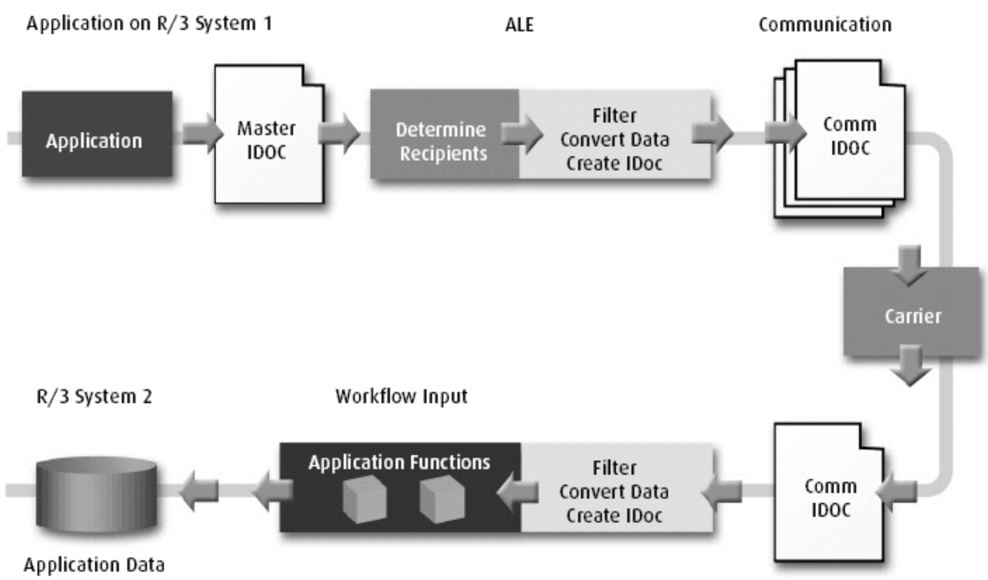

Figure 2 SAP ALE Architecture [Hummingbird, 2003]

A networked computing environment for a large enterprise is likely to connect many geographically distributed computers to a main database, which contains all the relevant information used in different activities within the enterprise. These computers are likely to use different hardware and/or operating system platforms. An IDoc encapsulates data so that it can be exchanged between different systems without conversion from one format to another. [SAP 3]

The IDoc type indicates the SAP format that is to be used to transfer the data for a business transaction between SAP Systems and partner systems. An IDoc is the instance of an IDOC type, supporting a real business process in the form of an IDoc type. The IDoc types define different categories of data, such as e.g. purchase orders or invoices. An IDoc type can transfer several message types: these are the "logical messages" that correspond to different business processes. An IDoc type is described using the following components:

- A control record. The format of the control record is identical for all IDoc types.
- One or more data records. A data record consists of a fixed administration part and a data part (segment). The number and format of the segments can vary for each IDoc type.
- Status records. The status records describe the processing stages, which an IDoc can pass through and have an identical format for each IDoc type. [SAP 5]

An IDoc can be generated at any point in a transaction process. For example, during a shipping transaction process, an IDoc that includes the data fields required to print a shipping manifest may be generated. After a user performs an SAP transaction, one or more IDocs are generated in the sending database and passed to the ALE communication layer. The communication layer performs a Remote Function Call (RFC), using the port definition and RFC destination specified by the customer model. The IDoc is transmitted to the receiver, which may be an R/3, R/2, or some external system. [SAP 3]

2.2.2. IDoc ORDERS

There are five IDoc types called ORDERS01 to ORDERS05 that are used to transfer logical message types ORDERS (purchase order) and ORDRSP (order confirmation). The message type ORDERS is used for both purchase orders and sales orders. Purchasing and sales share naturally the same IDoc type, because what is a purchase order on sender side will become a sales order on the receiver side and vice versa. [SAP 1]

In our mapping we use the IDoc type ORDERS05. It was chosen because it is the most comprehensive one of the five available. The other ORDERS IDoc types are subsets of this type. It has to be noted that only message types from ORDERS02 to ORDERS05 support global identifiers that are recommended by RosettaNet.

The most commonly used segments in ORDERS05 are described in Table 1.

Table 1 The most used segments in ORDERS05 [SAP 6]

Segment name	Segment description
E1EDK01	IDoc: Document header general data
E1EDK14	IDoc: Document Header Organizational Data
E1EDK03	IDoc: Document header date segment
E1EDK04	IDoc: Document header taxes
E1EDK05	IDoc: Document header conditions
E1EDKA1	IDoc: Document Header Partner Information
E1EDKA3	IDoc: Document Header Partner Information Additional Data
E1EDK02	IDoc: Document header reference data
E1EDK17	IDoc: Document Header Terms of Delivery
E1EDK18	IDoc: Document Header Terms of Payment
E1EDK35	IDoc: Document Header Additional Data
E1EDK36	IDOC: Doc.header payment cards
E1EDKT1	IDoc: Document Header Text Identification
E1EDKT2	IDoc: Document Header Texts
E1EDP01	IDoc: Document Item General Data
E1EDP02	IDoc: Document Item Reference Data
E1EDP03	IDoc: Document Item Date Segment
E1EDP04	IDoc: Document Item Taxes
E1EDP05	IDoc: Document Item Conditions
E1EDP20	IDoc schedule lines
E1EDPA1	IDoc: Doc.item partner information
E1EDPA3	IDoc: Document Item Partner Information Additional Data
E1EDP19	IDoc: Document Item Object Identification
E1EDP17	IDoc: Document item terms of delivery
E1EDP18	IDoc: Document Item Terms of Payment
E1EDP35	IDoc: Document Item Additional Data
E1EDPT1	IDoc: Document Item Text Identification
E1EDPT2	IDoc: Document Item Texts
E1EDS01	IDoc: Summary segment general

2.3. RosettaNet

2.3.1. Introduction to RosettaNet

RosettaNet is a consortium for developing B2B specifications in four grand industries and two communities. Many of the world's largest Electronic Components (EC), Information Technology (IT), Semiconductor Manufacturing (SM) and Telecommunications (TC) companies work to create and implement open e-business process standards in the name of RosettaNet. RosettaNet also includes the Logistics (LG) and Solution Provider (SP) communities. The over 500 companies working on RosettaNet represent together over $1 trillion in revenue.

RosettaNet's Global Industry Councils consist of councils from five industries mentioned above. They are divided into EC Council, IT Council, SM Council, SP Council and TC Council. The councils consist of representatives from each industry and actively promote RosettaNet and work together to actively further develop RosettaNet. RosettaNet Partners are members of the trading networks, or nearly related companies benefiting from the development of the trading network. Supply chain companies, such as manufacturers and distributors, provide expertise and human resources for RosettaNet project teams and also implement RosettaNet themselves. [RN 1]

2.3.2. RosettaNet functionality

It is easy for humans to engage in a business process. We share a common language and understand each other. Servers have a harder time understanding each other. HTML/XML functions as the alphabet for the servers and e-commerce applications function as the instrument by which an e-business process is transmitted. The real dialog is still missing though. That is why RosettaNet defines standard dictionaries to be used as words, PIPs to be used as messages between partners and a RNIF (RosettaNet Implementation Framework), which is used as the envelope to deliver messages between partners. [Cover Pages] RosettaNet also uses a global company identifier called the DUNS (Data Universal Numbering System) and a global product identifier called the GTIN (Global Trade Item Number). [RN 1]

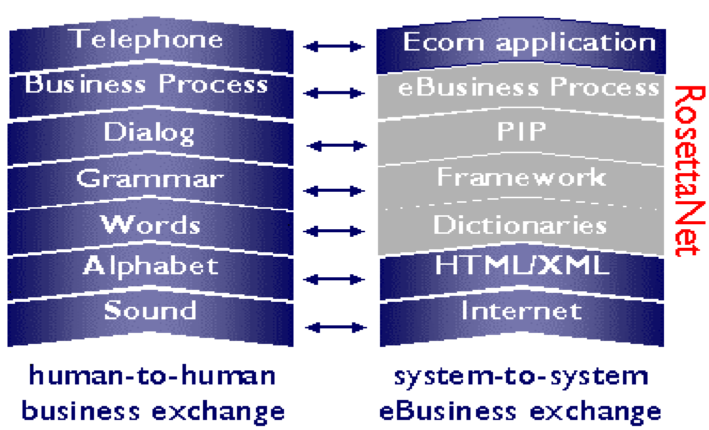

Figure 3 Comparison between human interaction and server interaction using RosettaNet [RN 1]

2.3.3. RosettaNet PIPs

RosettaNet PIPs are a very important part of the RosettaNet B2B process. They are specified XML-based messages, which define all the information needed in a business process between trading partners. The structures of the PIP messages are defined by DTDs (Document Type Definition). A DTD is a specification that accompanies a document, in this case the PIP, and identifies what the code (or markup) in the document means and how it is to be processed. RosettaNet has created Message Guidelines for each PIP to make the structure and contents of a PIP more readable to a human user. A Message Guideline describes exactly the same thing to a human as the DTD to the XML parser.

PIPs' functions:

- Encapsulate business processes
- Specifies activities, decisions and roles for the trading partners involved in the transaction
- Specify structure and format of the business documents
- Stand as discrete units of work that can be attached and built into other PIPs

There are PIPs created for most business processes and at the moment they amount to over 100 PIPs. The PIPs fit into seven clusters, representing the business logic, message flow and message content to enable conversation between partners. Each cluster is broken down into segments, which contain individual PIPs. All the clusters are listed below as well as all segments in cluster 3 and the PIPs in segment 3A. [RN 2]

- Cluster 1: Partner, Product and Service Review
- Cluster 2: Product Information
- Cluster 3: Order Management
    - Segment 3A: Quote and Order Entry
        - PIP3A1: Request Quote
        - PIP3A2: Query Price & Availability
        - PIP3A3: Transfer Shopping Cart
        - PIP3A4: Request Purchase Order
        - PIP3A5: Query Order Status
        - PIP3A6: Distribute Order Status
        - PIP3A7: Notify of Purchase Order Update
        - PIP3A8: Request Purchase Order Change
        - PIP3A9: Request Purchase Order Cancellation
        - PIP3A10: Notify of Quote Acknowledgement

        - PIP3A11: Notify of Authorization to Build
        - PIP3A12: Notify of Authorization to Ship
        - PIP3A13: Notify of Purchase Order Information
        - PIP3A14: Distribute Planned Order
    - Segment 3B: Transportation and Distribution
    - Segment 3C: Returns and Finance
    - Segment 3D: Product Configuration
- Cluster 4: Inventory Management
- Cluster 5: Marketing Information Management
- Cluster 6: Service and Support
- Cluster 7: Manufacturing

2.3.4. PIP 3A4

This study concentrates on PIP3A4: Request Purchase Order, Version:V02.00. The Request Purchase Order PIP enables a buyer to issue a purchase order and quickly obtain a response that acknowledges which of the product line items are accepted, rejected or pending. Figure 4 describes the business process model for the Request Purchase Order PIP. [RN 3]

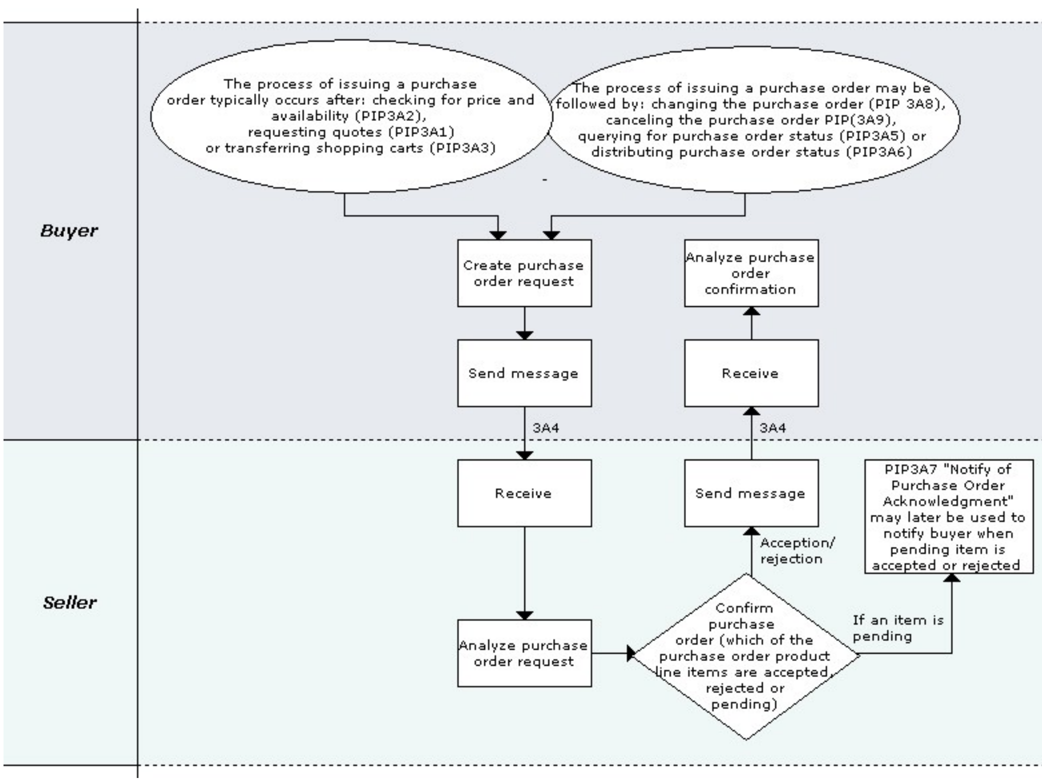

Figure 4 Business Process Model of PIP 3A4 [RN 3]

2.4. General comparison of the message formats

The differences between an IDoc and a RosettaNet business message are not huge. They both contain some kind of "envelope" for the message and then the actual business message separately from the envelope. The structural description of a RosettaNet business document can be viewed in figure 5 below.

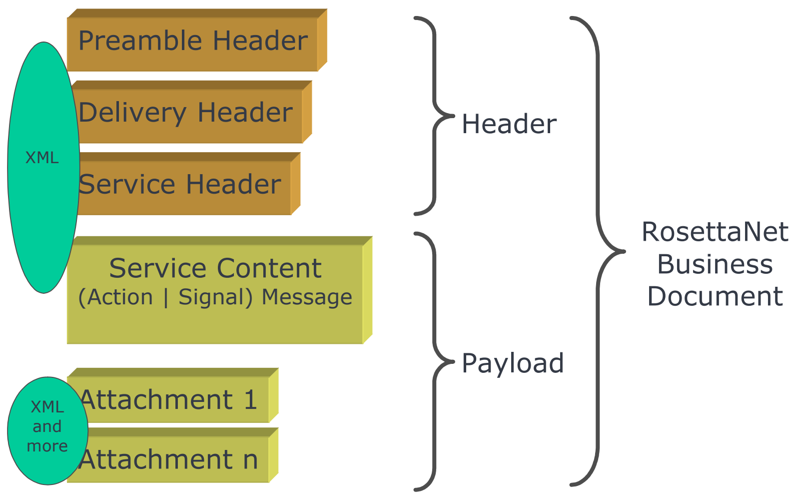

Figure 5 The structural description of a RN message [RN 5]

The Preamble Header contains e.g. information about what standard is used and which version of the standard is used. The Delivery Header contains information like destination information and message datetimestamp. The Service Header contains information about what kind of message it is and what PIP is used. The Service Content contains the actual business content, which is identified in the Service Header. The Service Content equals the PIP. When generating a message the basic need comes from a business process. This means that Service Content of the business message is the most important part; all the information in the Headers is not obligatory. Below in figure 6 is a general description of the actual service content (the PIP) in the message. [RN 5]

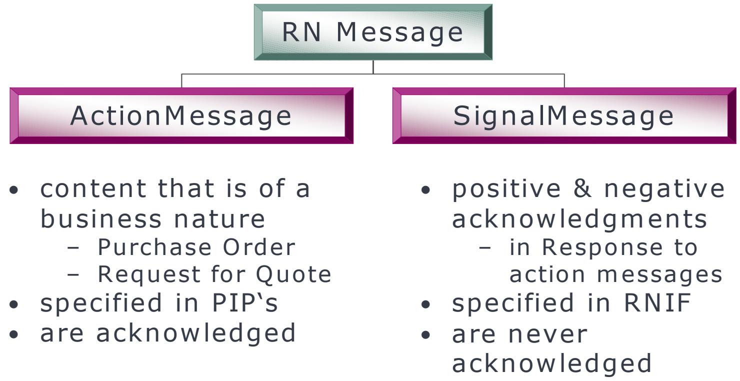

Figure 6 General service content description of a RN message [RN 5]

In a SAP IDoc the information is structured according to the interface described in figure 7 below. The IDoc contains administrative information, which can be thought of as the envelope for the message and therefore have basically the same function as the RNIF for the PIP. The application data is divided into segments and elements in a tree structure. The segments of IDoc ORDERS05 were presented in Table 1 in section 2.2.2. The IDoc also contains status information which is the same as processing status ("What has happened to the IDoc earlier?"). It is stored in the IDoc status information. The status information also contains details about errors and indicates the data in which the error occurred. This status information is not forwarded as part of the IDoc but separately via "status processing". [SAP ABAP]

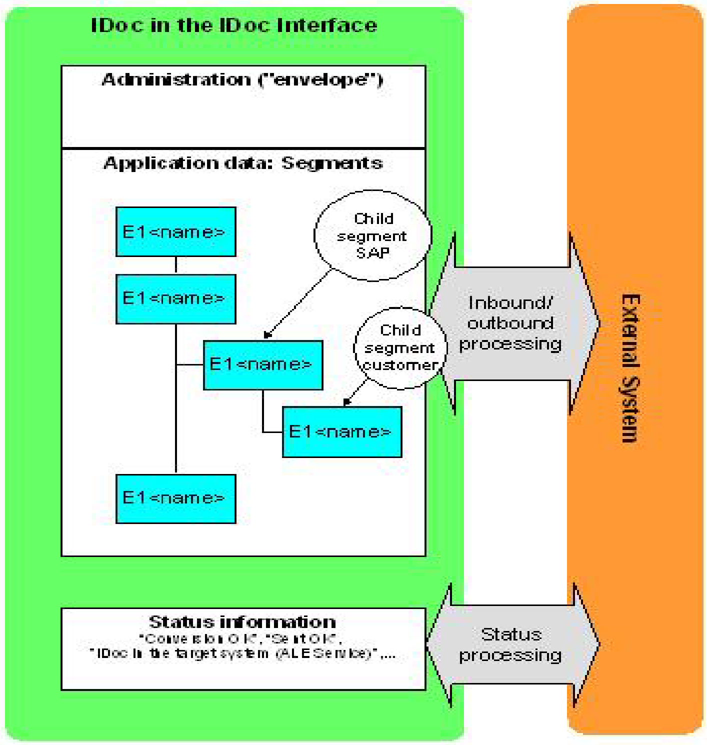

Figure 7 IDoc Structure [SAP ABAP]

No attachments can be sent with an IDoc, whereas a RosettaNet business message can contain attachments.

3. Web Services and IBM WebSphere

3.1. Web Services

According to W3C, the definition of Web Service is the following: "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [WS 2].

In other words, a Web Service is "a software application identified by a URI whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols" [WS 3].

Thus, underlying any Web Service there is a software application; the difference resides in the new approach of thinking about applications as services instead of the traditional stand-alone applications view. Typically in an enterprise environment, applications will be distributed across many machines, in order to improve performance, availability and scalability. Web Services provide a way to view such a system thinking of it as a collection of interacting services, each of which provides access to a well-defined collection of functionality. The fact that functionality is exposed as services is a key issue when it comes to flexibility, since it allows other pieces of functionality to make use of other services in a natural way regardless of their physical location, and the whole system is designed and implemented as a set of interactions among services, evolving through the addition of new services [Brown].

The described new approach is, of course, also valid when those services are provided by different companies, what makes it very interesting from the business point of view, enabling a higher degree of interaction and communication between companies.

Web Services provide the interoperability necessary to allow organizations to communicate or exchange data without any need to know each other's IT systems.

Web Services are a particular case of that overall service oriented architecture [IBM 4] using the Internet as the communication channel, requiring more attention because it is a public and insecure mechanism.

In a broad overview, we could say that Web Services involve the use of XML as a data container, SOAP as a means to transfer the data, WSDL as a way to describe the available services and Universal Description Discovery and Integration (UDDI) allows the publication and discovery of those available services. Thus, in summary, information about Web Services is published in UDDI registries, where we can find the WSDLs for them, which describes exactly how they work. Once we know all the details about a Web Service, using it is as simple as an SOAP message exchange, including XML based information.

Web Services are defined in independent layers (see figure 8), so that different standards and protocols can be used. A Web Service message can use XML, but it might transport binary data, generally using SOAP headers, but SOAP encoding is not required for the encoding of message bodies, and as for the transport, even though HTTP is the most common some others might be used (such as SMTP).

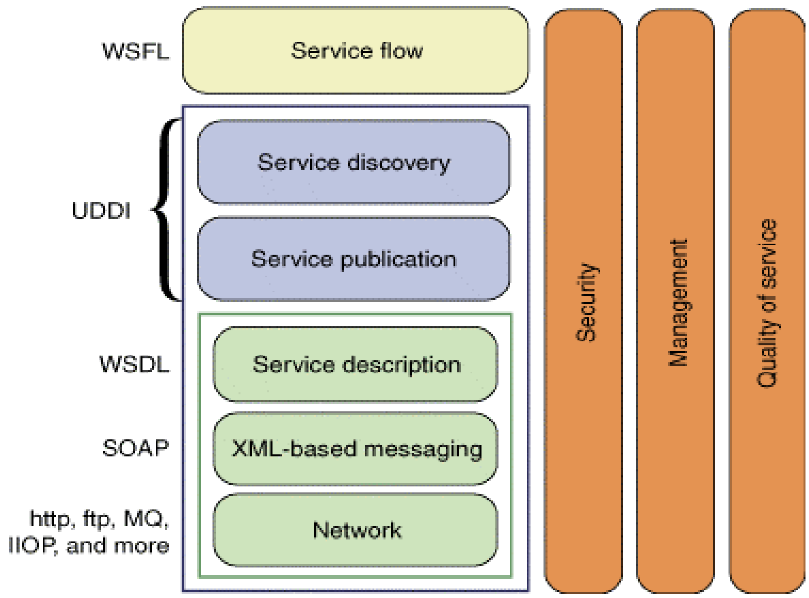

Figure 8 [IBM]

The layer-independent Web Service standards make formats and transport protocols flexible but at the same time entail some interoperability issues, which are tried to be solved by the Web Services Interoperability Group. [WS 4]

On the other hand, the service oriented architecture approach (See Figure 9) allows services implemented in different platforms and different programming languages to interact in a way, that a service requestor does not have to care any more about how the service offered by a service provider is implemented, because while he follows Web Service standards he will be able to interact with it. The service oriented architecture "hides" the actual implementation of the service from the user of the service, and that promotes interoperability.

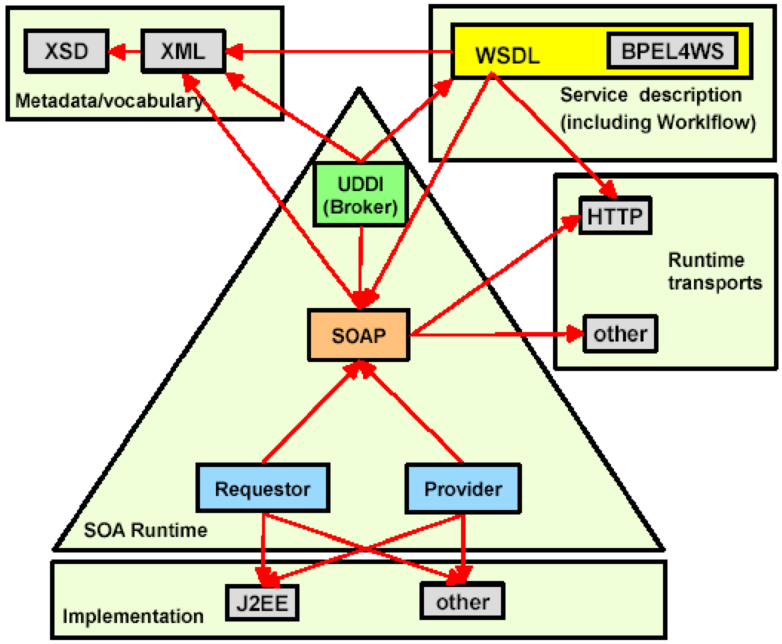

Figure 9 SOA = Service Oriented Architecture [IBM]

Web Services are intended to provide interoperability between systems, regardless of the architecture or implementation approach of each of the end-point systems. One significant programming model and architecture which fully enables the development of Web Services is the Java 2 Enterprise Edition (J2EE), which IBM has adopted in its major runtime environment, and particularly WebSphere Application Server. Some other companies, such as Sun or BEA have chosen the same major programming model. Together with J2EE, the other most significant programming model and architecture to develop Web Services is Microsoft's .NET platform, mainly conceived as an evolution of Visual Basic programming.

Each of the programming models has its own advantages and disadvantages, mostly linked to company specific issues, such as existing programming skills or vendor availability, which makes existing comparisons hardly reliable. Finally regardless of the chosen platform the developed Web Services should be able to interact with each other, and it seems that in the future both platforms are going to survive [Jones].

Within the scope of this report, the use of Web Services interfaces plays an important role in the flexibility of the designed system. As can be seen from the overall solution description, the RosettaNet framework is used as a communication way with third parties. At a very high level, we could divide the challenges faced when using RosettaNet in two levels: the messages themselves and their handling.

RosettaNet offers an agreed structure for the business messages which must be followed as defined in the PIPs, and it also provides guidelines on how to package, route and transport those PIP messages, through the RNIF specification. Thus, we face a situation where messages with many different contents and from different sources might be delivered and received but all of them need to be packaged and routed in a similar way. The adopted solution, where the RNIF related needs are provided by a Web Service, is a clear answer to that situation. This way, no matter how PIPs are created or what the system does with the received ones, they can take advantage of the functionality provided by the Web Service for their handling (See Figure 10). In order to support both the delivering and reception of PIP messages, the designed system must implement both the Web Service server and client.

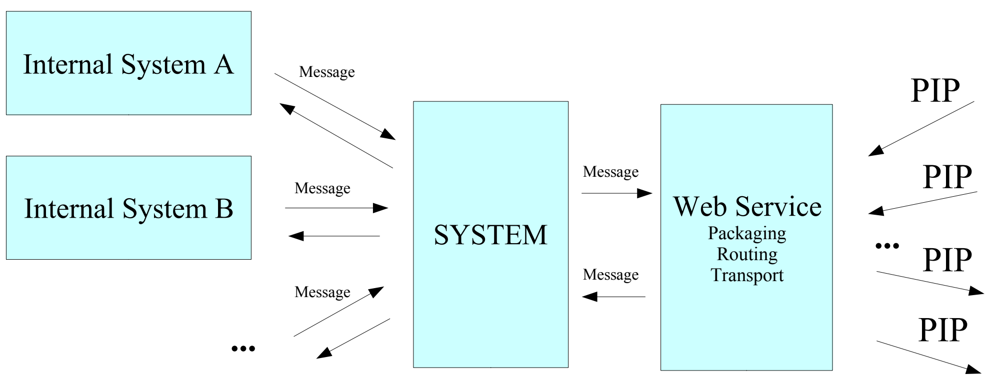

Figure 10 Web Service Interface [WS 1]

On the other side of the designed system, with regards to SAP, the scope of this report is very narrow, since SAP is just one of the internal systems depicted in Figure 3.3, so that in the relationship between SAP and the designed system the communication is much more defined and consequently less suitable for a division in different services.

3.2. IBM WebSphere

IBM has chosen J2EE as its major runtime environment, and has built a complete set of software products and tools around it, known as the Websphere family of products.

There are now more than 20 products in the WebSphere family, and according to IBM it is a "scalable Internet software platform or middleware that can support any kind of e-business, from small sites engaged in content publishing to large sites handling high-volume Web transactions typical of business-to-business e-marketplaces" [IBM 5]

The family of products includes solutions to develop, host, run, and integrate e-business applications, based on industry standards and technologies. The center piece of the platform is the WebSphere Application Server (WAS), which provides different services based on standards (persistence, naming services, etc.) and might be combined with WebSphere MQ (former MQSeries) which through its messaging capabilities allows integration of business processes, data and applications. The WebSphere software platform supports virtually any kind of business applications, and covers the whole application life cycle, from its design and implementation to its deployment.

Web Services are fully supported by the WebSphere family of products, mainly through the WebSphere Application Server, the WebSphere Business Integrator and the design and development tool WebSphere Studio Application Developer (WSAD), but IBM is increasingly offering Web Services related support and capabilities in some other products outside the WebSphere family, such as in DB2 and Lotus families.

Most of the products within the WebSphere family are available for different platforms and operating systems, since IBM itself supports different environments, such as Windows, Linux, AIX, OS400 and OS390. The above mentioned choice of J2EE as major platform helps with this issue, since one of Java's most appreciated characteristics is its "write once, run everywhere" motto, that is, the same code can be executed in different platforms without making it platform-specific as it is usual in some other environments or programming languages.

For our report and prototyping we have used two of the main products of the WebSphere family: the WebSphere Application Server as deployment platform and the WebSphere Studio Application Developer as implementation tool. Both products are presented and discussed in the following sections.

3.2.1. IBM WebSphere Application Server

The heart of the WebSphere family of products is the WebSphere Application Server. It comes in different editions with different capabilities, providing tools and components that allow users to develop web applications and services with dynamic content using Java-based technologies. Applications developed with IBM's WSAD can be deployed to the WAS, either locally or remotely, and debugged within WSAD, thus providing a totally integrated development environment.

The WAS provides the runtime environment for J2EE applications to be deployed. It is a J2EE compliant server, so that regardless of the development tools or platforms used, any J2EE application can be deployed in the WAS. Lately, within their "e-business on demand" view, IBM has emphasized the role of Web Services, making WAS an easy-to-use and up-to-date platform when it comes to Web Services, supporting the latest standards and recommendations.

WAS is available for many different platforms and provides a web browser based administration console which allows deployment, configuration and monitoring of any J2EE application. Within the provided runtime environment, WebSphere Application Server provides every service required by the J2EE specification, such as transaction management, security, etc., as well as some other advanced features for large scale and high availability environments, such as clustering, performance monitoring and optimization tools, availability capabilities, connectivity and scalability.

3.2.2. IBM WebSphere Studio Application Developer

On the development and implementation side of the applications, IBM's main product is the WSAD, which provides tools needed to create, develop, test, and manage all of the resources involved with building web, J2EE and Web Services applications.

In order to support different development or implementation roles within a development team, WSAD allows to have different views of the same development resources and their organization, arranged in customizable "perspectives", each of them suitable for one role (Enterprise Java Bean developer, application designer, graphic designer, XML developer, etc.). Each “perspective” contains a set of “views” which in turn contain the basic working tools, such as file editors (visual and code HTML editors, XML editors, etc.) and many different wizards automating usual tasks (creating Java classes, creating and configuring servlets, etc.).

Those resources are organized into projects that correspond to modules defined in the J2EE specification. The specification compliant structure allows the developer to import and export similar files, which can contain Web projects, Enterprise Java Bean projects or at a higher level Enterprise Application projects, containing every necessary resource for an application to be fully functional and ready to deploy. The natural next step after developing an application is to deploy it to an application server (such as WAS), but the WSAD also includes a subset of the WAS, which allows to locally deploy the developed applications and test them before deploying to a "real" WAS.

The IBM WSAD development environment is based on Eclipse platform technology [IBM 3], which provides several common facilities that help tools vendors to concentrate on their own special areas of expertise. Eclipse is an open source project supported by many different vendors, which are able to extend its core functionality through a “plug-in” architecture. The Eclipse platform provides the basic functionality and services and then each vendor can make use of those features and extend them to include more advanced features. In IBM’s WSAD product, the Eclipse platform has been extended to include J2EE, web and Web Service development capabilities, supporting at the same time some other plug-ins from third parties also related to Java development (such as Ant or JUnit).

For our prototyping work we have used WSAD as development tool, taking advantage of many of its advanced functionalities. Due to the characteristics of the system designed, we have worked mainly with J2EE, Java and XML perspectives.

In a similar way, using the mapping capabilities of the XML perspective, WSAD is able to automatically generate a XSL file, which can be used for the XSL transformation. Providing WSAD with both the initial and final XML files, XML Schemas or DTDs it allows to graphically map the entry with the exit variables and generates the corresponding XSL file. Furthermore, for those situations in which the mapping is not straightforward, but involves some additional transformations (in our case they would be implemented in Java), the mapping editor allows to introduce those external Java functions in the mapping, giving a clear picture of how the transformation is done.
Thus, the WSAD integrated tools and wizards help to increase the development speed while focusing in the core functionality of the system. [IBM 2]

4. Mapping between the Message Formats

For Prototype two mappings were done using XSLT: SAP IDoc message type ORDERS was mapped into RosettaNet PIP 3A4 PurchaseOrderRequest and PIP 3A4 PurchaseOrderConfirmation was mapped into SAP IDoc message type ORDRSP. This chapter describes the mapping principles between these two standards. Figure 11 gives an overview of mapping from ORDERS IDoc to PIP 3A4 PurchaseOrderRequest.

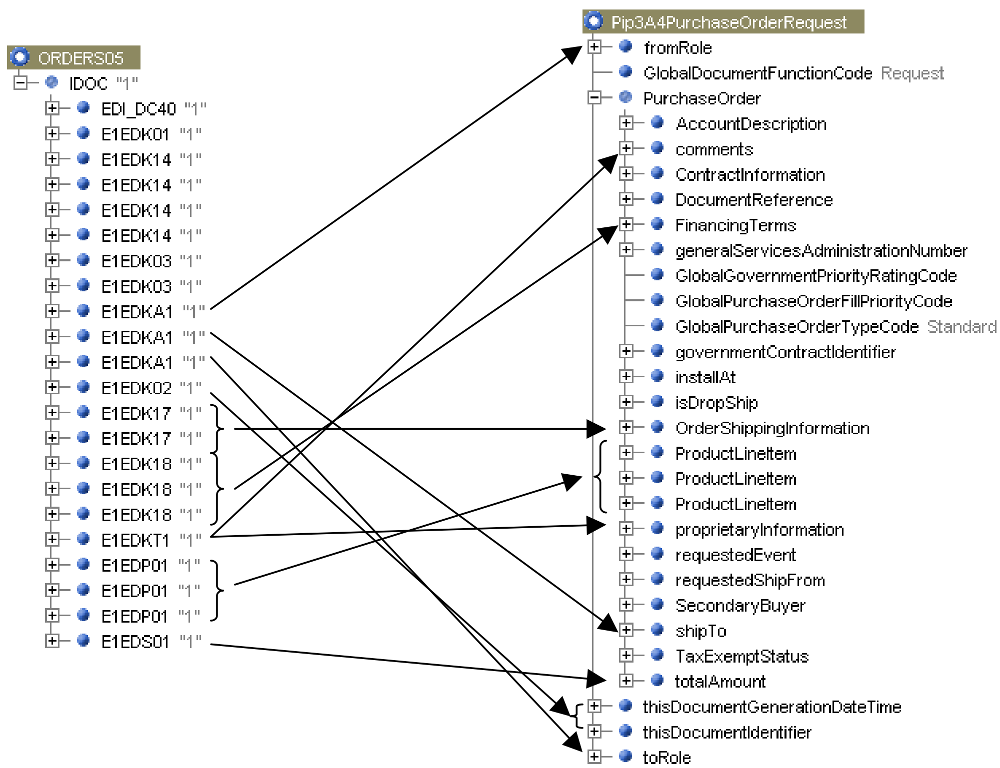

Figure 11 Mapping of ORDERS IDoc into PIP 3A4 Purchase Order Request

4.1. Structure

The PurchaseOrderRequest consists of four basic levels of elements:

1. The base level contains information about the sender and receiver, ID and date of the document and a PurchaseOrder element.
2. The PurchaseOrder level contains elements with information that applies to the whole order and one or several ProductLineItem elements.

3. ProductLineItem level contains elements with product specific information and possibly ProductSubLineItem elements.
4. ProductSubLineItem level can be used to describe a part of a product line item.

ORDERS message types consists of similar elements:

1. Administrative data contains information on sender and receiver, IDoc document number and date.
2. Header data contains elements that apply to the whole order.
3. Item data contain information on purchase order line items.
4. Status information

4.1.1. Administrative Data

The administrative data of an IDoc is not directly comparable to the sender and receiver information of the PIP's. The administrative data of an IDoc is like an envelope telling the ALE layer where the message should be sent. The mandatory fromRole and toRole can be mapped with Header Partner Information in this case.

4.1.2. Partner Information

In IDocs the partners are handled in re-occurring segments with identifying qualifiers. The segment that can be considered as partners in PIP are listed in table 2 together with corresponding IDoc qualifier and level that it can be found in. In SAP it is possible to create additional roles if none of the existing ones match.

Table 2 The PIP partner roles and corresponding SAP partner roles

PIP Partner	Level	IDoc Qualifier	Level
fromRole	Base	AG (Sold-to-party)	Header
billTo	PurchaseOrder	RE (Bill-to-party)	Header
financedBy	PurchaseOrder	RG (Payer)	Header
primaryContractWith	PurchaseOrder/ ProductLineItem		
secondaryContractWith	PurchaseOrder/ ProductLineItem		
installAt	PurchaseOrder/ ProductLineItem		
CarrierInformation	PurchaseOrder/ ProductLineItem	SP (Carrier)	Header
shipTo	PurchaseOrder/	WE (Ship-to-party)	Header / Item

	ProductLineItem		
CustomerInformation	ProductLineItem		
toRole	Base	LF (Vendor)	Header

RosettaNet recommends using GlobalBusinessIdentifiers for customers. The standard that is used is DUNS, Data Universal Numbering System provided by a company called Dun&Bradstreet. This information is stored in a subsegment of the correponding partner segment in IDoc. RosettaNet also recommends using GlobalLocationIdentifiers which are DUNS + 4 numbers.

If ProprietaryBusinessIdentifiers, i.e. any other identifiers, are used, it has to be agreed between the trading partners how the linking the local identifiers is done. This can be done either on source or target system or in an additional mapping table that is called during XSL transformation. Some partner roles like delivery address can be passed on only with address data. It is not relevant or even wise to map all business partner roles with ProprietaryBusinessIdentifiers.

4.1.3. Header Data

In both PIP and IDoc certain data like OrderShippingInformation can either exist in header or item level. The logic that the line level information always overrides the header level information is the same for both PIP and IDoc.

4.1.4. Item Data

RosettaNet recommends using Global Trade Item Numbers (GTIN) for identifying materials. Most recognized and used GTINs are the EAN numbers like UCC/EAN-13 specified in EAN.UCC standards. If the GTINs are not used, either material ID or vendor material number can be mapped into ProprietaryProductIdentifier. The same as for ProprietaryBusinessIdentifier applies to the maintaining and mapping of source and target material numbers i.e. it has to be agreed where and how to maintain the link.

4.1.5. Mapping Problematics

In this chapter some field mapping problematics are addressed. This is not complete list of all cases handled, but a collection of representative cases.

Sub Line Items

PIP 3A4 has segments called ProductSubLineItem. Without an example or further description of its usage it is not possible to say how it should be mapped or can it be mapped. In IDoc there can be for example several schedule lines i.e. different delivery dates for one line item.

Requested Delivery Date

In IDoc there are several possible segments where the requested delivery date can be found. It can be either in the header or the item date segments E1EDK03, E1EDP03 with qualifier '002' or in the schedule line segment at item level E1EDP20.

Terms of Payment

Payment Terms is not a mandatory field, but if it has to be used some additional logic would have to be developed either in the receiving system or in the XSL transformation to in-house format. PIP carries only information on days due net or payment day of month, and payment percentage.

Terms of Delivery

GlobalShipmentTermsCode corresponds to the standard trade definition keys called Incoterms. But the PIP3A4 does not seem to contain field for the mandatory definition part of the terms of delivery.

Time Created

RosettaNet recommends using Coordinated Universal Time for field thisDocumentGenerationDateTime. This feature is not supported by SAP so only informal format with local time can be used. In most cases the time is not relevant, but in time critical cases it has to be agreed how to handle this: with transformation or interpretation rules.

PIP Specific Fields

For example GlobalTransportEventCode defines if line delivery is to be ‘Dock’, ‘Pickup’ or ‘Ship’. This can best be derived from Incoterms by making selection rules in XSL with Choose.

5. Implementation

A prototype for the mapping was constructed using WebSphere application server. Design of the prototype was based on the available source information and computing resources. In this section, an overview of the implementation is given, along with key design decisions. First, the source information used as the basis of implementation design is described. Then the overall architecture of the solution the related sequences of actions within the system are discussed. The remaining five subsections describe the implementation in more depth, including message transformation (mapping implementation) and class design.

5.1. Available Source Information

For the project, we had the following technical data at hand:

- WSDL description of the web services interface to the integration middleware
- XML Schemas / DTDs of RosettaNet PIPs and SAP IDoc messages
- Some sample SAP IDoc and RosettaNet PIP messages

5.2. Overall Architecture

As the general approach to the problem, an external server is connected to the integration middleware. The server can both send and receive RosettaNet PIP messages, and convert them to and from SAP IDocs. The IDocs are obtained from and saved to designated directories by the server.

The server is called SAP Adapter and it consists of two modules:

- A servlet capable of receiving messages, converting them to SAP IDocs, and storing the result to a specific directory. This is called the *Receiver*.
- An application monitoring a specific directory, converting found IDocs to RosettaNet PIPs, and sending them to the integration middleware. This is called the *Sender*.

The service runs using standard Java classes, which are initialized via a controller servlet. The servlet is configured with a directory and other configurations at deployment time. Directory and file structure is described later in this section.

SAP Adapter runs within WebSphere application server. The architecture is depicted below.

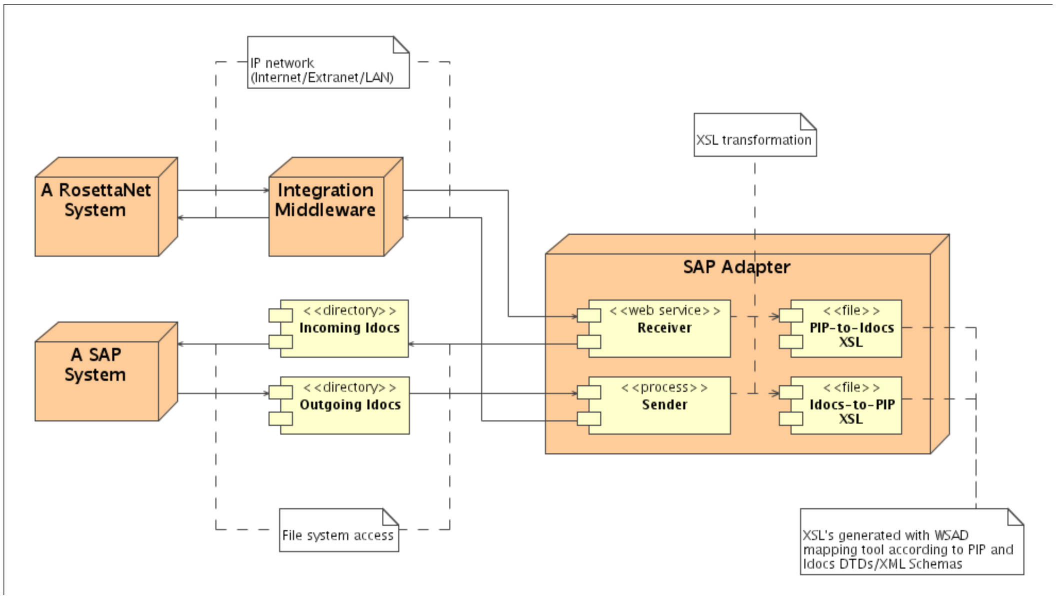

Figure 12 The architecture of the system

5.3. Sequence of Actions

The following is a description of a typical sequence of actions within the system upon receiving a document. The description is divided into two cases, receiving an IDoc document (order request) from SAP system, and receiving a PIP document (order response) from a RosettaNet system.

5.3.1. Purchase Order from SAP

SAP has an extension, called the SAP Business Connector (SAP BC), which has the ability to read the IDoc structure and parse it into XML. The IDoc provided for our test case is first parsed into XML in the SAP BC, before it is sent to us. That means that we are working with an IDoc, which already is in XML format. The received IDoc in XML is then stored in a directory on the WebSphere Application Server (WAS), which runs our application to transform the IDoc to a RosettaNet business message.

SAP originated messages enter the system via a special directory dedicated to incoming IDocs. (We say "incoming" as in "incoming to our system". Of course the documents are "outgoing" from the viewpoint of the SAP system.) They leave the system via network as they are dispatched to the integration middleware. The integration middleware, in turn, sends them to the target RosettaNet system.

Sequence of Actions:

1. An IDoc order request is placed in the incoming IDocs folder by the SAP system.
2. SAP Adapter finds the IDoc file in the directory.
3. Administrative data is extracted from the IDoc to a header Java bean, namely the recipient partner number, using a XSL transformation.
4. Recipient partner number is resolved to a recipient name, according to routing configurations of the system. The recipient name is the same as the name configured in the integration middleware for an external RosettaNet system. The recipient name is recorded in PIP message headers.
5. A PIP document is generated out of the IDoc file using a XSL transformation.
6. The PIP document is sent to the integration middleware along with the headers.
7. The sent PIP document is archived in a separate directory. The original IDoc file is moved to an archive directory.

If anything fails during the transformations or message sending, the original IDoc is moved to an archive directory for failed documents.

5.3.2. Order Confirmation from a RosettaNet system

Order confirmations are first received by the integration middleware and then dispatched to the SAP Adapter. The PIP documents are transformed into IDocs and saved in a special directory dedicated to outgoing IDocs ("incoming" for the SAP system).

Sequence of Actions:

1. A PIP message is received by the system, sent by the integration middleware.
2. Sender of the message is extracted from the PIP message header and resolved to configured data of the external system, according to routing configuration. The data of the system is placed into a header Java bean.
3. An IDoc document is generated out of the PIP message using a XSL transformation. As a part of the transformation, IDoc administrative data is generated according to data in header Java bean.
4. The generated IDoc file is stored in the directory for outgoing documents. The received PIP document is archived.
5. The SAP system may retrieve the document from the outgoing directory.

If transforming the message should fail, the failed PIP message is archived in a separate directory.

5.4. Message Transformation

The transformations described above are accomplished by applying XSL Transformations (XSLT) to the messages. The XSL definitions are based on mapping constructed with an XSL editor.

A sample of XSL Transformations:

```
<Pip3A4PurchaseOrderRequest>
 <xsl:for-each select="/ORDERS05/IDOC/E1EDKA1">
  <xsl:if test="normalize-space(PARVW/text()) = 'AG'">
   <fromRole>
    <PartnerRoleDescription>
     <ContactInformation>
      <contactName>
       <FreeFormText xml:lang="EN">
```

```
    <xsl:value-of select="BNAME/text()"/>
  </FreeFormText>
</contactName>
<facsimileNumber>
  <CommunicationsNumber>
    <xsl:value-of select="TELFX/text()"/>
  </CommunicationsNumber>
</facsimileNumber>
<telephoneNumber>
  <CommunicationsNumber>
    <xsl:value-of select="TELF1/text()"/>
  </CommunicationsNumber>
</telephoneNumber>
```

Listing: Part of a XSL transformation

Additional Java code is required within XSLT for handling the PIP header data. IDocs are complete XML documents, which contain all relevant data, including headers. Transferring PIPs over Web Service interfaces, however, require a separate header sent with the ‘body’ of the message. A tool-generated Java class with associated data container classes presents the header. This header must be mapped to and from IDoc data. The mapping of the header data from IDoc to PIP is accomplished as follows: the header data is gathered from the IDoc to a Java bean using a special XSL transformation. Then Java bean properties are set to the actual header object, prior to sending the PIP message. Similarly, upon receiving a PIP document, the header is passed separately in a header object. The required header data is moved from the header object to a Java bean, which is passed to the XSL transformation. The transformation uses the bean to insert relevant data to the produced IDoc document.

5.5. Routing

The system can mediate messages between a single SAP system and multiple RosettaNet systems. For communicating with SAP, each RosettaNet system must have associated administrative data to be placed into IDocs that are transformed from PIP messages. Administrative data can be configured per system using a property file, which contains key-value mappings for each system and the SAP system itself. Each key is associated to either one of the RosettaNet systems by a symbolic name, which is same as the name in the integration middleware, or to the SAP system by the name “sap”.

A sample listing of the routing property file is presented below.

```
# Static properties of SAP system
sap.businessServiceCode=The SAP System
sap.port=SAPABC
sap.partnerNumber=SAPCLNT100
sap.systemType=LS

# Partner properties
rnstub.businessServiceCode=RN Stub Implementation
rnstub.partnerNumber=41841
rnstub.partnerType=LI
rnstub.partnerPort=TABC

# Partner properties
systemX.businessServiceCode=Customer X system
systemX.partnerNumber=12345
systemX.partnerType=LI
systemX.partnerPort=TCED
```

Listing: An example routing property file.

When transforming an IDoc to a PIP, the correct system data is retrieved using the partner number in the IDoc. Actually the only relevant data is the name of the system (for example “rnstub”), which is used by the integration middleware for routing.

When transforming a PIP to an IDoc, the system is identified by the BusinessServiceCode element in the PIP document. The corresponding IDoc properties (partnerNumber, type, and port) are used when constructing the IDoc document.

5.6. Service Configuration

The important configurations for the service are

1. The locations of XSL transformation style sheets
2. The locations of directories used for input/output and archiving
3. The location of the routing properties for external RosettaNet systems

There are three required style sheets (XSL) for the mapping:

1. PIP-to-IDoc mapping
2. IDoc-to-PIP mapping
3. Header data extraction transformation

The first two are generated using a general mapping tool. The third one is used to extract relevant header information from IDoc and store it to a Java bean. Handling of headers is discussed in the message transformation section above.

The directories for the service are:

Sender

1. The directory from where outgoing IDocs are extracted and transformed.
2. The directory where sent IDocs are archived.
3. The directory where IDocs failing the transformation or sending are archived.
4. The directory where sent PIP messages are archived (optional).

Receiver

5. The directory where received and successfully transformed PIPs are archived.
6. The directory where received but unsuccessfully transformed PIPs are archived.
7. The directory where results of PIP-IDoc transformations are saved, i.e. outgoing IDocs to be handled by a SAP system.

The directory configuration is depicted below in Figure 12.

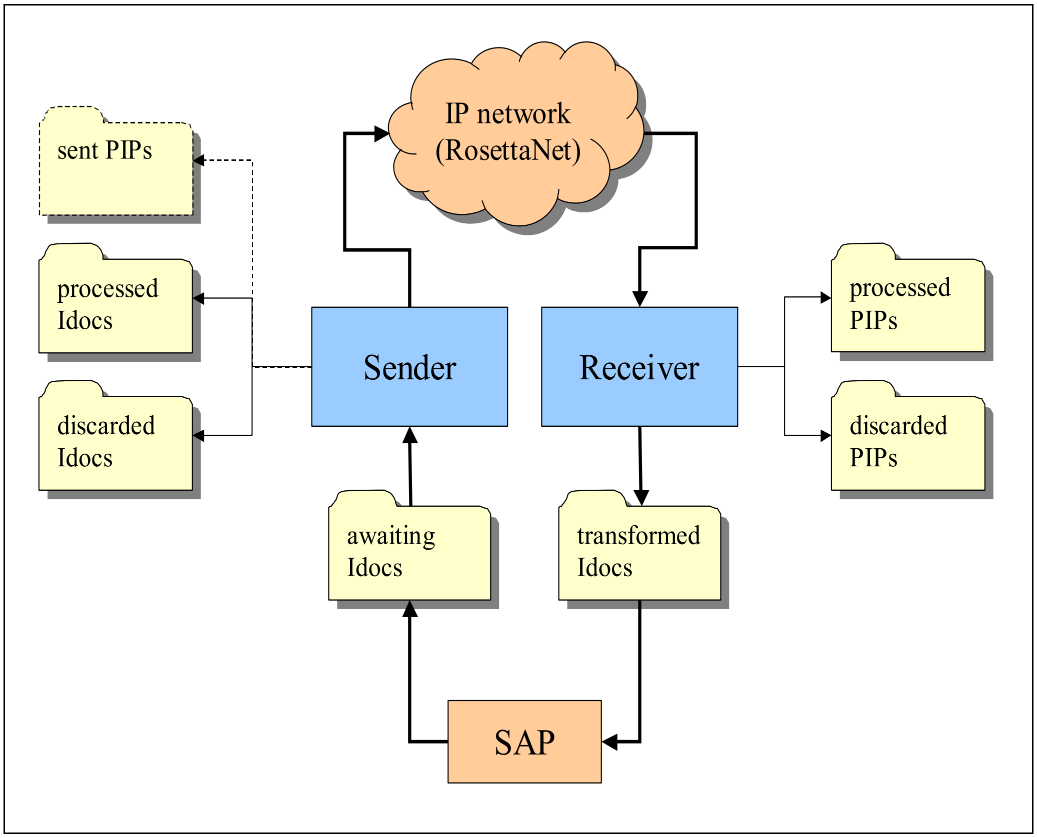

Figure 13 Directory configuration of the service. Black thick line shows the essential data flow in the system.

5.7. Class Design

The implementation was based on web services and web applications. Therefore it was natural to have a control servlet looking after the execution of the business logic. The **ControllerServlet** takes care of reading configuration parameters (directory and file locations), initializing the logic components (with configuration parameters), and starting and stopping the service. It also provides a simple user interface for user interaction with the service.

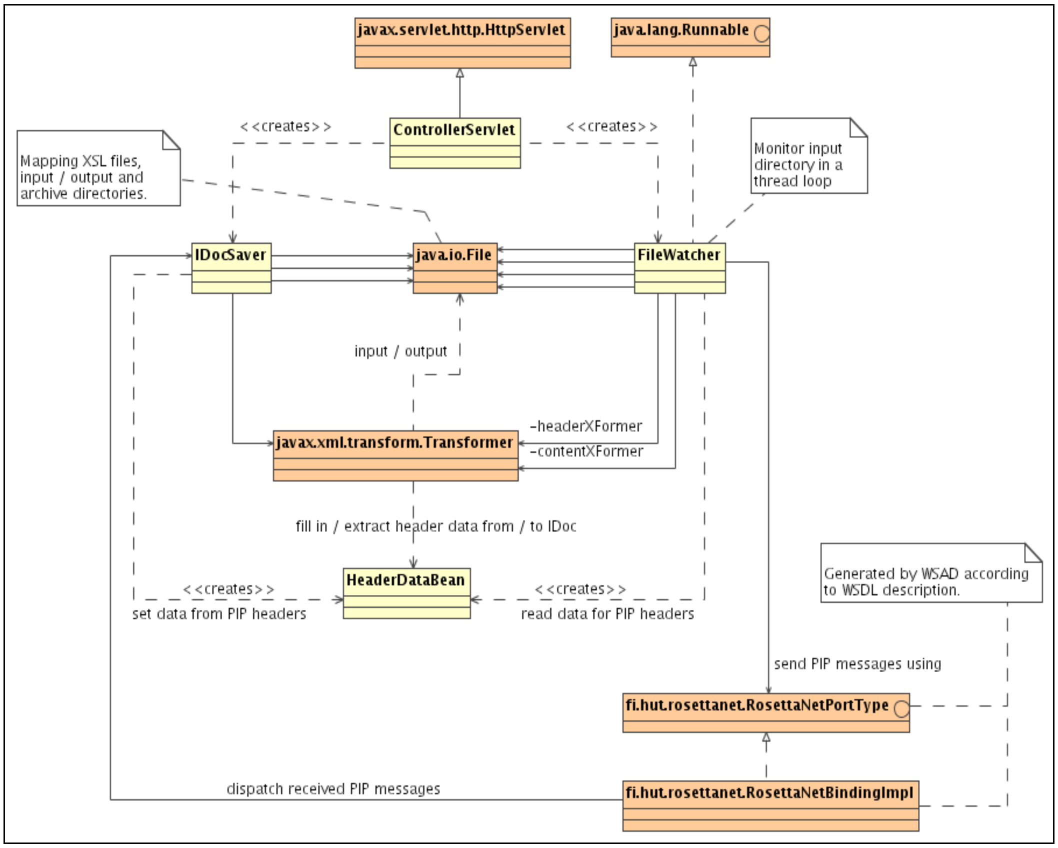

Figure 14 Class Design of the Implementation. The four core classes are colored light yellow

The actual business logic, i.e. handling files and doing transformations, is located in the **FileWatcher** and **IDocSaver** classes. FileWatcher (the "sender" in the architecture) monitors constantly the given "in" directory and transforms all IDoc files in it to PIPs. Then it archives the results and sends the PIP messages to the integration middleware (over the network). IDocSaver (the "receiver", in turn, receives PIP messages (dispatched by RosettaNetBindingImpl) and transforms them into IDocs. It too archives the results and saves the transformed IDoc to the designated "out" directory.

The fourth class participating in the actual logic is the **HeaderDataBean** class, which is used to transport header data in and out from XSL transformations: PIP payload XML does not contain header information, whereas IDocs do. The bean represents essentially PIP header information.

5.8. Test Setting of the Prototype

As we did not have a real SAP system at our disposal for testing, nor did we have any RosettaNet-capable systems in the other end, some special arrangements had to be done for testing purposes.

SAP emulation. Our system interfaces with SAP systems only via the file system. Consequently, it was sufficient to emulate a SAP system only at the file system level. Retrieving proper sample documents from a SAP system and placing them into the "incoming" directory was a sufficient replacement for a real SAP system.

RosettaNet emulation. The role of an external RosettaNet system was to receive the transformed order requests and respond to them. As our system interfaces with RosettaNet systems through integration middleware, it was necessary to construct a dummy RosettaNet service "stub", running as a separate web service accessible by the integration middleware. The stub service was implemented using the same WSDL interface as the SAP Adapter, containing only logic for receiving an order PIP, generating a response out of it, and sending the response back to our system (via the integration middleware). Further discussion of the RosettaNet stub is out of the scope of this paper.

6. Evaluation and discussion

6.1. Mapping

The mapping is evaluated in the aspects of correlation of source and target message structures, data content and format correlation and the mapping aids available. The WebSphere tools are discussed in more detail in the next chapter.

6.1.1. Correlation of the Source and Target Messages Structures

It can be easily noticed that SAP is a partner in the RosettaNet consortium and has taken part in the PIP definition work: SAP has developed many enhancements to meet the requirements of electronic commerce message standards of ebXML and RosettaNet.

The logic in dividing data into header and item data was very similar in both message types. The biggest difference in structure seemed to be that PIP 3A4 contains sub line items and SAP ORDERS or ORDRSP do not have sub segments under line item. Without proper example of ProductSubLineItem usage it cannot be explicitly stated whether this data can be mapped into an IDoc or not. From ORDERS to PIP it is not a problem, because it is not a mandatory segment.

One big difference is that RosettaNet PIPs are not be enhanced without permission from RosettaNet Consortium whereas SAP R/3 has functionality called User exits (also Customer exits) which can be used to populate any custom segments in IDocs. I.e. almost any data that can be found in SAP can be written into an IDoc making it easier to deliver all needed data for transformation into PIP. This applies also to any other IDoc to PIP transformation.

6.1.2. Mapping Aids

Both message types are well documented and the documentation can be found easily. The quality of the documentation is good: they are both up-to-date and accurate. There could have been more examples of what kind of data the PIP 3A4 segments contain; there were quite little available even in the Internet. This can affect the results if some segment descriptions were misinterpreted.

The XML mapping tool in WebSphere Studio Application Developer proved not to be very useful in doing RosettaNet PIP mappings due to the nested structure of PIP. XML mapping tool tends to create template calls from upper level segment to lower level segment. The segment names are often identical like ' CommunicationsNumber' for both telephone and fax. They can also both be called from many different upper level segments and this would lead into too many parameters in nested calls.

Basic software package called <oXygen> by SyncRO Soft Ltd was used for creating the XSLT files. The benefit of this package was the ability to test the transformation immediately after writing the rules and the ability to validate the documents. There are several different XSLT software packages on the market but it was not in the scope of this study to do a comparison of different tools.

6.1.3. Generalizability of the Results

There are restrictions to generalization of this result in the areas of the chosen business case, prototype construction and message type:

Chosen Business Process

> Purchase orders are among the most common business transactions in electronic commerce. In this area there has been standards like EDI for a long time and the business logic is established. This has to be considered when thinking of implementing more rare business messages. But it is also very probable that other common message types like invoices are fairly straight forward to implement.

Prototype Construction

> The prototyping consisted of mapping one constructed source file into a PIP standard message, i.e. this prototype lacks a real business case background

with real target environment of a receiving business partner. The prototype also does not handle the whole business process due to limitation in implementation time. The consequence of this approach is the limited ability to generalize the results into real life environment with unpredictable incidents and error handling.

Message Type

The PIP3A4 is defined for IT and EC businesses and the generalization of the results might be restricted due to this fact.

6.1.4. Effort estimate

One is likely to be able to create and test a basic mapping between SAP IDoc and PIP in about a week given that is that the message contents of sending and receiving side have been specified first and that the person doing the mapping is familiar with both SAP IDocs and PIPs. The creating and testing any additional function modules for mapping on server can be estimated to take two days each. Main workload is on setting up and testing in the sending and receiving systems though most of the work needs to be done only once because some of the results can be re-used for messaging with other partners.

6.2. Implementation

The implementation can be evaluated in two ways: considering usefulness of using Web Services and WebSphere in particular for the task, and evaluating this particular solution to the problem.

6.2.1. Usefulness of Web Services and WebSphere

Web Services provide a good solution to business integration. Standard-syntax message formats (XML) help in communicating (SAP “speaks” XML when used with its Business Connector). Web Services interface and WSDL descriptions make interfacing between systems simple.

WebSphere—both WAS and WSAD—proved to be useful in implementing the integration. In particular, the following features saved us from a lot of work:

- Automatic generation of Web Service interfaces from a WSDL document
- Automatic generation and deployment of Web Service components (web service servlet receiving messages)
- Generation and easy modification of the needed web application and enterprise application configuration files, i.e. deployment descriptors (DDs) for Enterprise Application Archives (EARs) and Web Application Archives (WARs). Also web services deployment descriptors were automatically generated.

In our prototype, we have used WAS as a deployment platform managing the provided server through the web based interface, which allows to remotely deploy and configure applications. For our deployments and configurations we have used just the basic capabilities of the WAS, thus we cannot properly evaluate all of its features.

Unfortunately we ran into an annoying software bug that appeared to be originated from WebSphere itself, or the computing environment it was run. This slowed down the development and testing of the prototype.

6.2.2. The Presented Prototype

The prototype we have constructed as an implementation of the solution to the problem can be evaluated according to a number of criteria. Each criterion is discussed below.

Modularity

The implementation is modular in the sense it separates the logic for receiving and sending messages and mapping of the message formats. It is, however, tied to the file system approach. A more modular approach would have been to have a separate web service solely for the mapping of file formats, and another web service for communicating with the SAP system via the file system. Another glitch in modularity is the handling of header data: a change in the mapping of header data probably requires a change in the HeaderDataBean class, and hence in the logic of constructing / deconstructing header data object of the PIP message.

Changeability

The most probable change point in the system is the definition of the message mappings. In principle the mapping is easily modified with an XSL editor. The resulting XSL only needs to be placed in the correct place and the system needs to be reinitialised.

Performance / Scalability

The implementation was not designed with performance as the topmost priority. However, it can be multitasking when it comes to receiving messages (the transformation uses no shared resources other than the directories of the file system). When it comes to sending messages, the system is inherently single tasking. Multitasking will cause race conditions regarding the input folder and the IDoc messages in it.

The system is scalable to the extent what the application server provides. In theory it can be distributed and clustered, and it may be even desirable to have multiple instances of the system monitoring different directories. However, such conditions have not been tested in practice.

Robustness / Error Handling / Manageability

The implementation is capable of handling errors in a rudimentary fashion. The errors typically deal with malformed documents or network errors. In such error conditions (actually in any error conditions), the document under processing is stored to a ‘failure’ directory (either the one for outgoing documents or the one for received documents). In case of no error, the document is stored to a respective ‘success’ directory. In addition, sent PIP messages can be archived to a ‘sent’ directory. Use of categorized directories makes it easier to monitor the operation of the system and detect error conditions. Additionally, each file stored in any of the directories is labelled with a name indicating the time of processing, in order to identify the correct (and related) files.

For troubleshooting and accounting, the system also writes logs of its operations and their outcomes. The log used is the standard application server log.

The system also tries to avoid error conditions by posing checks to the configuration directories and files upon start-up. If errors are found, the system ceases to start.

Reliability

Related to robustness, the system is not particularly designed to be reliable. Using categorized directories for different types of messages minimizes the possibilities of losing data due to malfunctions. However, no other reliability or availability related mechanisms (such as auto-recovery after an error) are implemented.

Generality

Technically the system is able to apply transformations to a wide variety of message formats (in the domain of IDocs and especially PIPs). As some PIPs or IDocs may require complex state logic or “conversation”, it is not currently implemented, but with little effort, it could be. Serving multiple business message types simultaneously requires use of multiple instances of the service, which may not be desirable. Thus, the system is currently not ready for large-scale generalization.

As a summary of the evaluation of the implementation, it can be reasonably said that Web Services and WebSphere significantly reduced the workload otherwise associated with this type of integration. The designed and implemented prototype proves the basic concept of the implementation correct, but lacks many of the qualities commonly associated with business integration systems.

7. Conclusions and ideas for further research

7.1. Conclusions concerning the mapping of the information

SAP is a partner in the RosettaNet consortium and has taken part in the PIP definition work and this can also be seen when mapping the information between the messages. The logic in dividing data into header and item data was similar in both message types and there are many similar elements in the message standards. SAP can deliver all the necessary information required by PIP in our simple scenario.

Mapping the messages is easier from IDoc to PIP, than the other way around, since all the obligatory information a PIP requires in order to be valid can be acquired from an IDoc, but not necessarily the other way around. An IDoc contains much in-house data, that a PIP cannot deliver. Our scenario, starting the request order process from the SAP end was in this aspect the easier scenario. It could have been harder to create a system where the receiver of the purchase order is the SAP system, see figure 15.

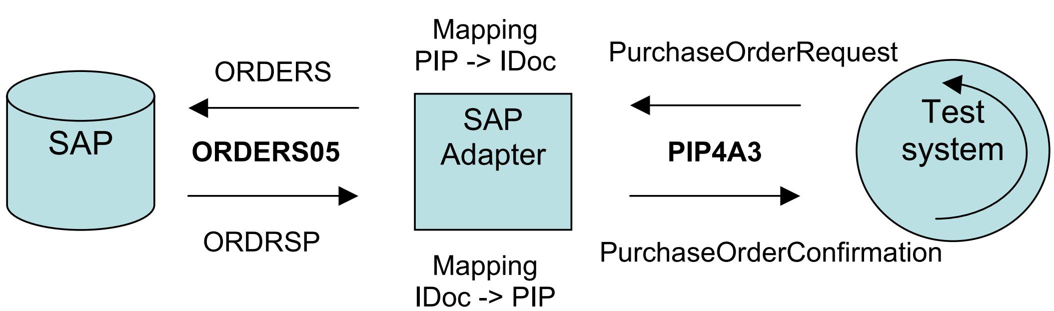

Figure 15 A more challenging scenario where mapping is from general format to in-house format - in this example SAP IDoc.

The message types of both standards are well documented and understandable. The quality of the documentation is good: they are both up-to-date and accurate.

There are restrictions to generalization of these mapping results, since the studied business transaction was purchase order, and it is the most common business messages between two companies.

XSLT proved to be an efficient technology to do the transformations with. To our knowledge, there are no other noteworthy alternatives when it comes to transforming XML documents into others.

7.2. Conclusions concerning the usefulness of IBM WebSphere and Web Services

Web Services seem to provide a good solution for business integration. XML messages can be easily communicated and the Web Services interface and WSDL descriptions make interfacing between systems simple.

Both WAS and WSAD proved to be useful in implementing the integration. There were many useful features that saved us from a lot of work. The WSAD generated very efficiently a huge number or Java classes according to the WSDL description, just by defining the WSDL file at hand. It also automatically generated and deployed the Web Service components. The generation and modification of the Deployment Descriptors and Enterprise Application Archives was easily done in the WSAD, and then they were easily deployed on the WAS.

We cannot though properly evaluate the features of the WAS, since we just used the basic capability of it, i.e. deploying our application on it. We did no configuration changes to the server. For unknown reasons the server seemed a bit unstable and had some problems running, but we cannot draw any conclusions from that. According to our knowledge the server had just a test setup, and had not been thoroughly configured.

As a conclusion about Web Services and WebSphere it can be reasonably said that they significantly reduced the workload otherwise associated with this type of integration. The total amount of Java code we ourselves wrote only adds up to around

1000 lines, which proves the help provided by the WSAD. It is difficult to estimate the amount of code we would have had to write, if not much of it would have been automatically generated by the WSAD, but it is at least several times more.

7.3. Conclusions concerning our prototype

The designed and implemented prototype proves that the basic concept of the implementation is correct, although it lacks many of the qualities commonly associated with business integration systems. The prototype has a simplified architecture; it only supports one type of message and it has to be connected to a single SAP system.

Our prototype is a working prototype, but not of industrial quality. It is a sketch of what could be constructed, a proof that an industrial implementation with similar needs can be created. The system has some restrictions; it lacks some reliability, manageability, scalability and generality qualities.

7.4. Suggestions for further research

An interesting approach for further research would be to create a system, which supports the whole business process, both ways. The environment should have a real sender and receiver, which would clarify the setup situation, and the system could be tested in “real life”. Also an actual real message that is used by some company for creation and testing would be very useful. The system should also contain error handling and solutions for unexpected incidents in the message handling. When using real senders and receivers the problems with firewalls, authentications etc. would also have to be taken into consideration.

The idea of using directories to store SAP IDocs and monitor them could also be changed to a more real world scenarion. A particular case could be the one in which the system is designed to be able to work with different releases of SAP, where the core functionality required from the system is the same, with slight changes in some details such as communication methods. Anticipating the possibility of this situation we believe a better solution would be to provide the core functionality of the designed system as a Web Service available for the internal systems, so that both entry and exit points for the system would be Web Services interfaces, providing additional flexibility (See Figure 14).

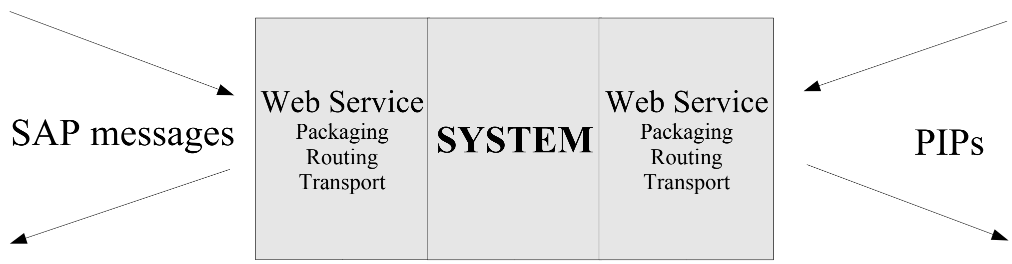

Figure 16 Two Web Service Interfaces

The designed solution and developed prototype are easily upgradeable to the above depicted architecture, but due to the faced constraints we just point at it as a future research and improvement possibility.

References

[Bernroider et. al, 2000] 1 Edward Bernroider, Stefan Koch. Differences in Characteristics of the ERP System Selection Process between Small or Medium and Large Organizations. 2000.

[Brown] Brown, Alan; Johnston, Simon and Kelly, Kevin. Using service oriented architecture and component based development to build web service applications. 2002. Rational Software Corporation

[Chieu] Chieu, T ; Fu, S ; Pinel, F ; Yih, J. Unified Solution for Procurement Integration and B2B Stores. 2003. ICEC

[Corcho] Corcho, O ; Gómez-Pérez, A. Solving integration problems of e-commerce standards and initiatives through ontological mappings. 2001. IJCAI. Proceedings of the Workshop on E-Business and Intelligent Web at the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, USA.

[IBM 1] IBM. IBM Home. [Referenced 2004-04-13] URL: www.ibm.com

[IBM 2] IBM. IBM WebSphere. [Referenced 2004-04-13]
URL: www.ibm.com/developerWorks

[IBM 3] IBM. Eclipse platform. [Referenced 2003-03-27] URL: www.eclipse.org

[IBM 4] IBM International Technical Support Organization “Patterns: Service Oriented Architecture and Web Services” IBM, Apr. 2004.

[IBM 5] IBM WebSphere family of products [Referenced 2004-04-16] URL: http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=highlights

[JAVA] Sun Microsystem. Java. URL: www.sun.com

[Jones] Jones, Teresa. Diversity demands a development strategy. 11.11.2003. Computer Weekly, pg. 32.

[Kasanen et. al, 1993] Kasanen, E., K. Lukka and A. Siitonen. 1993. The constructive approach in management accounting.

[Medjahed] Medjahed, B; Benatallah, B; Bouguettaya, A; Ngu, A; Elmagarmid. A. Business-to-business interactions: issues and enabling technologies. 2003. The VLDB Journal (2003) 12: 59–85.

[RN 1] RosettaNet Consortium. RosettaNet Background Information. [Referenced 2004-04-15] URL: http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity[OID[CE1E0334F39FFF44BB7E86E55025E423]]

[RN 2] RosettaNet Consortium. PIPs. [Referenced 2004-04-15] URL:
http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity[OID[5F6606C8AD2BD411841F00C04F689339]]

[RN 3] RosettaNet Consortium. 2001. PIP 3A4: Request Purchase Order. Version V02.00.00 [Referenced 2004-04-158] URL: http://www.rosettanet.org/PIP3A4

[RN 4] RosettaNet Consortium. 2001. PIP 3A6: Request Purchase Order. Version V02.00.00 [Referenced 2004-04-158] URL: http://www.rosettanet.org/PIP3A6

[RN 5] Schlatcher, Simon. RosettaNet [Referenced 2004-04-158]
URL: http://n.ethz.ch/student/jodaniel/37-310/slides/RosettaNet_Simon_Schlachter.ppt

[SAP 1] SAP. Company Home. [Referenced 2004-04-19]
URL: http://www.sap.com/company/

[SAP 2] searchSap.com.R/3. [Referenced 2004-04-19]
URL: http://searchsap.techtarget.com/sDefinition/0,,sid21_gci212859,00.html

[SAP 3] searchSap.com.IDoc. [Referenced 2004-04-19]
URL: http://searchsap.techtarget.com/sDefinition/0,,sid21_gci852485,00.html

[SAP 4] Karkhanish, Santosh. SAP R/3 Overview. [Referenced 2004-04-19]
URL: http://www.karkhanisgroup.com/SAP1/sap.htm

[SAP 5] SAP Interface Repository. IDoc message types. [Referenced 2004-04-19]
URL : http://ifr.sap.com/catalog/doc.asp?IFR_IDOC_page.htm

[SAP 6] SAP Interface Repository. IDoc type ORDERS01. [Referenced 2004-04-19]
URL : http://ifr.sap.com/catalog/query.asp

[SAP ABAP] SAP ABAP/4 Programming. IDoc structure. [Referenced 2004-04-19]
URL: http://www.sap-basis-abap.com/edi/IDoc-structure.htm

[SAPfans] Sapfans. SAP R/3 Overview. 2003. [Referenced 2004-03-12]
URL: http://www.sapfans.com/sapfans/sapr3con.htm

[Hummingbird, 2003] Hummingbird Ltd. Solutions for SAP. [Referenced 2004-05-06] URL: http://mimage.hummingbird.com/alt_content/binary/pdf/collateral/wp/etl-sap.pdf

[UCC] Global Trade Item Numbers (GTIN) Implementation Guide. [Referenced 2004-04-19 as Google html version of the document]
URL : http://www.uc-council.org/ean_ucc_system/pdf/GTIN.pdf

[WS] W3. Web Services. [Referenced 2004-04-23] URL: www.w3.org/2002/ws/

[WS 2] W3C. Web Services. Working Group "Web Services Architecture" [Referenced 2004-05-12] URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[WS 3] W3C Web Services Working Group "Web Services Architecture Requirements" [Referenced 2004-05-12] URL: http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/

[WS 4] Web Services Interoperability Group. Web Services. [Referenced 2004-04-23] URL : www.ws-i.org

Glossary

.NET: A Microsoft operating system platform that incorporates applications, a suite of tools and services and a change in the infrastructure of the company's Web strategy.

ALE: Application Link Enabling system. ALE uses IDocs (see IDoc) to exchange data between logical systems.

API: Application Programming Interface. A set of routines, protocols and tools for building software applications.

BAPI: Business Application Programming Interface. Message format in SAP.

B2B: Business-to-Business, the exchange of services, information and/or products from one business to another.

BASIC: Beginner's All-purpose Symbolic Instruction Code. Developed by John Kemeney and Thomas Kurtz in the mid 1960s at Dartmouth College, BASIC is one of the earliest and simplest high-level programming languages.

DTD: Document Type Definition. A DTD states what tags and attributes are used to describe content in an SGML, HTML or XML document, where each tag is allowed, and which tags can appear within other tags.

DUNS: Data Universal Numbering System. A unique nine-digit sequence recognized as the universal standard for identifying companies. Also referred as D&B D-U-N-S. Provided by company called Dun & Bradstreet.

EAN: The name of standardization organization that develops standards for unique identification of products, locations, services etc. Originally comes from words 'European Article Number'. See GTIN.

E-BUSINESS: Electronic Business. Is the conduct of business on the Internet, not only buying and selling but also servicing customers and collaborating with business partners.

EDI: Electronic Data Interchange, the transfer of data between different companies using networks, such as the Internet

EJB: Enterprise Java Bean, server-side Java component.

ERP: Enterprise Resource Planning, a business management system that integrates all facets of the business, including planning, manufacturing, sales, and marketing.

GTIN: Global Trade Item Numbers. GTIN is an umbrella term used to describe the family of numbers that identify trade items (products and services) according to EAN.UCC System.

HTML: Hypertext Markup Language. Is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page.

HTTP: HyperText Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands.

IDoc: Intermediate Document. SAP format that is to be used to transfer the data for a business transaction between SAP Systems and partner systems.

JAVA: A high-level programming language developed by Sun Microsystems.

J2EE**:** Java 2 Platform Enterprise Edition.J2EE is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs and protocols that provide the functionality for developing multitiered, Web-based applications

JSP: Java Server Pages, web pages with dynamic content through the use of Java code on them.

PIP: RosettaNet Partner Interface Processes. Define business processes between trading partners

RFC: Remote Function Call. An application program interface to R/3 applications from SAP.

RNIF: RosettaNet Implementation Framework. It specifies the packaging, routing, and transport of all PIP messages and business signals.

RosettaNet: RosettaNet is a consortium for developing B2B specifications in four grand industries. Subsidiary of UCC.

SAP R/3: SAP R/3 is the most used ERP system in the world.

SMTP: Simple Mail Transfer Protocol, a protocol for sending e-mail messages between servers.

SOAP :Simple Object Access Protocol Is a lightweight XML protocol that governs the exchange of information in a distributed environment. SOAP provides a standardized XML envelope for carrying other application specific XML payloads, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses

UCC: Uniform Code Council. The Numbering Organisation in the USA to administer and manage the EAN·UCC system standards in the USA and Canada. See GTIN.

UDDI**:** Universal Description, Discovery, and Integration. UDDI provides a standards-based set of specifications for service description and discovery, as well as a set of Internet-based implementations

URI: Uniform Resource Identifier. The generic term for all types of names and addresses that refer to objects in the World Wide Web (a URL is a kind of URI).

VISUAL BASIC: A programming language and environment developed by Microsoft, based on the BASIC language.

W3C: Worl Wide Web Consortium. Creates Web standards. W3C's mission is to lead the Web to its full potential, which it does by developing technologies (specifications, guidelines, software, and tools) that will create a forum for information, commerce, inspiration, independent thought, and collective understanding.

WAS: WebSphere Application Server. IBM’s J2EE application server

WSAD: WebSphere Studio Application Developer. IBM’s tool for developers.

WSDL: Web Services Description Language.WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate

XML: Extensible Markup Language. A specification developed by the W3C. XML is a pared-down version of SGML designed especially for web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations.

XML Schema: A way to validate data in XML.

XSL: Extensible Style Language, a specification for separating style from content when creating XML pages.

XSLT: Extensible Style Language Transformation, the language used in XSL style sheets to transform XML documents into other XML documents