A Message Exchange Architecture for Modern E-Commerce

Barry Dowdeswell[1] and Christof Lutteroth[2]

[1] AARN Innovation Limited,
P.O.Box 82-171, Highland Park, Auckland, New Zealand
barry@aarn.biz
[2] Department of Computer Science, The University of Auckland,
38 Princes Street, Auckland 1020, New Zealand
lutteroth@cs.auckland.ac.nz

Abstract. This paper describes the EDIS business messaging architecture, which is a modern, lightweight system that is used in numerous companies. It explains the requirements for such a system, the problematic issues that have to be dealt with, and also some aspects of the wider context of e-commerce. Furthermore, it compares the presented architecture to similar systems like, for example, MS BizTalk and discusses related research on enterprise architecture.

1 Introduction

In the business world, computers have been used since the 50's and have become an important tool for modern business. All the large enterprises rely heavily on systems for enterprise resource planning (ERP) which integrate and automate many of their administrative tasks. In the 90's the focus of enterprise computing shifted to the Internet, and the idea to use the net in order to make business with customers (B2C) ended in heavy losses for many investors and companies. In the last years another aspect has come to general attention: computer-supported business between businesses (B2B). Again, the Internet is supposed to bring revolutionary changes to the enterprise world, and this alleged revolution is heralded by a plethora of new standards trying make their way into the enterprise.

When looking at these things from a scientific perspective, it is extremely important to distinguish between fact and hype, between what technologies are and what people want them to be. For all what it seems, computer supported B2B is an old concept that dates back to the 80's. At that time, the main standards defining the format of business messages were that of electronic data interchange (EDI) [8], and the infrastructure on which these messages were transported were value-added networks (VANs). Although it may seem, with the emergence of XML-based data formats and the web services technology [15], that EDI is outdated, one has to acknowledge that the overwhelming majority of B2B traffic is still done using the old standards. These technologies worked successfully for twenty years now, while most new standards have still to prove themselves. And

D. Draheim and G. Weber (Eds.): TEAA 2005, LNCS 3888, pp. 56–70, 2006.

no matter how mature new standards will turn out to be, the old standards will continue to be used for quite some years to come.

The system described in this paper is one for business message exchange. It was developed from the mid-nineties onward to be an easily deployable B2B solution that can be adjusted to the changing needs of a company. Its creators have been in the e-commerce business since it began in the 80's, and as a consequence the system supports old standards for message transport and encoding as well as new ones and can interface to different ERP systems. It is extensible and designed with the possibility of future change in mind.

2 Requirements for Modern B2B

Before we discuss the architecture of the EDIS B2B system, we would like to point out the requirements that a modern B2B system has to satisfy. These requirements have changed, since the new wave of B2B standards brings up new maturity and compatibility issues. Standards for electronic B2B have become a strategic factor, which means that decisions about it have to be considered carefully.

First of all, it is not always possible for a company to choose their B2B standard themselves. Many companies depend on trade relationships with bigger business partners, and if such a business partner decides to change its e-commerce system, might very well expect its smaller partners to adjust to it. This makes *flexibility* of a system a basic requirement, especially in today's world where many changes are just about to occur. As we can see, for example, in [14], the strategies with which companies adopt new B2B technologies can differ significantly. The evolving nature of electronic B2B and the steady change that takes place has been subject to many studies, e.g., the one described in [9]. It is important to note that we usually need not just a single standard for B2B message exchange: there exist, for example, different standards for message encoding and message transport, respectively, and those standards can be combined in many different ways. To sum up, a flexible B2B system has to support different standards for different tasks, and it should be possible to combine them in different ways.

Another aspect of flexibility concerns the workflow that a B2B system supports. This workflow, which might, for example, define what happens when a system error occurs, may also be subject to steady change. Therefore, it must not only be possible to configure the technological behaviour of the system, but it must also be possible to change the way the system reacts to different circumstances that may occur.

Different companies often have different ERP systems. Some companies even have their own particular one. Whatever the ERP system is that a company uses, a good message exchange system has to be able to interface to any of them. Since the interface to the ERP system can be a very proprietary one, it may be necessary to create a customized adapter component. A good B2B system will support a developer in this task.

In contrast to B2C, B2B is usually not dealing with huge loads of transactions. Consequently, performance is not as important as in B2C. Whereas in B2C we usually do not expect a single business transactions to have a particularly high value, but rather expect many transactions with relatively small value, the value of a B2B transaction can be extremely high. A flawed B2B transaction can cause immense costs. Therefore we have to make sure that erroneous messages are singled out, either automatically or manually. In order to perform such error checking, the system needs to have some knowledge about the business logic; and since business logic varies from company to company, it must be possible to configure it as easily as possible. For correct business messages manual translation does not make sense: it is a tiring task and that makes it particularly error prone. However, if an erroneous message is detected, human intervention is an absolute necessity. For resolving errors, human intelligence is irreplaceable, and trying to resolve errors automatically would be too high a risk. Nevertheless, a good B2B system should support the workflow of *error handling* done by humans.

A B2B system has to offer high *reliability*. If messages get lost or corrupted, this can have disastrous consequences for the supply chain of a business. Thus, such a system has to rely on mature technology, like, for example, a good database management system. Only if a persistent log is kept, the system can be brought back into a consistent state after failure. Since such a system naturally involves many business partners, errors might not necessarily occur just in the local system, but in any system involved. A B2B system has to work correctly even if other's systems fail, and must also handle and resolve errors created by others. It has to be *fail operational*, i.e., remain in an operational state even when an error occurs. Whatever happens, the system has to log every major system event, like incoming and outgoing messages, message translation and possible failures. Only like this a system can, for example, detect when duplicate messages are received and prevent duplicate processing. A very important issue related to reliability is *availability*. While it may be tolerable in B2C when a service is temporarily unavailable, this is totally unacceptable for a business. In B2C this may cost a couple of small successful trades, but in B2B this can be existential.

Last but not least, a B2B system has to guarantee *security*. This has different aspects: first of all, mechanisms like digital certificates and signatures have to make sure that all the sender of each message is authentified. Once *authentification* is done, we can use it in order to prevent unauthorized messages from being processed, i.e., ensure *access control*. Usually we also do not want others to get to know the content of the business messages, so we have to apply *encryption* to make them unreadable to others. Finally, a business partner might want a unforgeable receipt for the reception and/or processing of a message, which establishes a property called *non-repudiation*, i.e., the business partner at the other end cannot repudiate the reception and/or processing of the message.

3 The EDIS Architecture

Figure 1 shows an overview of the system's architecture. As we can see, the system consists of different modules, some of which form groups that take care of a particular aspect of the system's functionality. On the left side of the figure we have different modules for handling the transport of in- and outgoing message with different protocols. On the right of the figure we have different modules for the translation of messages, which handle the data flow between message queue and ERP system. The message queue is the central data structure of the messaging system, with the transport protocol modules storing and retrieving messages from it.

Many of the system's modules are written in an interpreted scripting language which is largely equivalent to MS Visual Basic. Some of the transport protocol modules and all of the translation modules are implemented as scripts. This has the advantage that these modules are very easy to program, to deploy and to debug. The scripting language offers a high level of abstraction through a comprehensive framework of library functions. It eliminates the need for compilation to a low-level representation, which makes it platform independent. Scripts are controlled by the interpreter and sufficiently isolated from the rest of the system; this makes it possible to ensure safety of execution and control a script's capabilities, i.e., the functions of the system it is allowed to use, making execution also secure. The clean abstract interface of the interpreter and its inherent capacity to control the execution flow of a script make debugging easier. Although interpretation is naturally slower than the execution of compiled code this is, as we have discussed in Sect. 2, not a problematic issue.

Its scripting capabilities make the system highly customisable, since scripts allow the developer to freely develop translation and communications functions

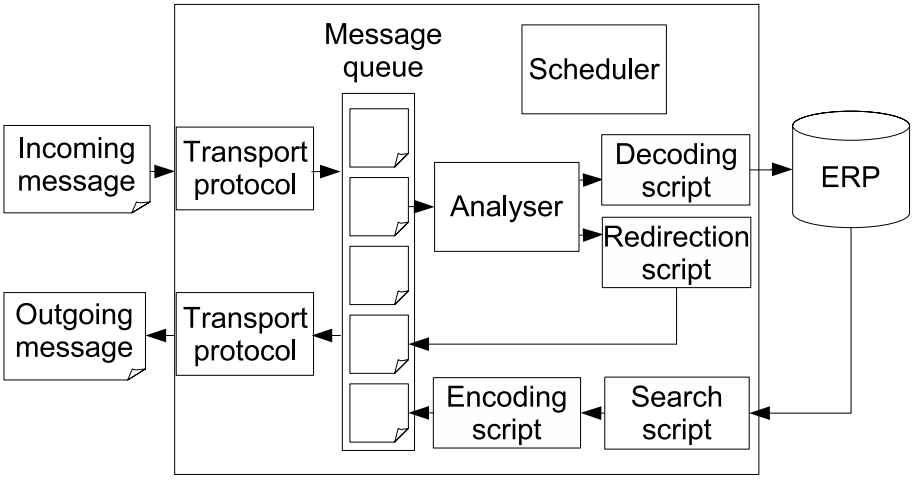

Fig. 1. Overview of the EDIS system

specific to their needs. Note, however, that this flexibility does not infringe the system's integrity: the system has a mature, "closed" core and library that ensures the scripts operate in an consistent and safe environment. While end-users who are developers have full access to the script source, they cannot modify the core modules such as, e.g., the scheduler, the AS2 protocol implementation or the EDIFACT library parsing routines.

All scripts are, directly or indirectly, activated by the scheduler of the system. This module allows it to start scripts at regular time intervals. The analyser module collects elementary data about a message and then delegates the its processing to an appropriate translation script. In the following sections we will describe all of these parts in detail.

3.1 The Message Queue

The message queue is the central data structure in the system. Whatever message passes through the system will be archived in the message queue. It is implemented on a relational database, which controls the access to it by all the other system parts and makes sure that the data remains in a consistent state. Since the message queue is such an important part of the system, we want to cover it in more detail and explain how its relational schema looks like.

Before a message is sent it is usually put into an electronic envelope. Such envelopes can hold multiple messages that are sent to a business partner in a single delivery. The message queue keeps track of envelopes as well as of newly created messages that are not enveloped yet. For each envelope the system has to keep track of the different messages embedded in it. This is achieved with two tables, Queue and QueueXref, which are illustrated at the top of Fig. 2. Table Queue contains all the envelopes and unenveloped messages currently stored in the system, together with fields that explicitly describe some of their fundamental properties. Table QueueXref keeps track of and contains information about the messages that are stored in the envelopes of table Queue.

Let us first consider the records of table Queue: field MsgData holds the raw data of an individual message or an envelope as it was received or created in a BLOB. Field EDI_ref is a running integer number and the primary key of the table. Fields SenderID, PartnerID and ServideID are foreign keys to the tables Partners and Services. If the record contains a message and not an envelope, SenderID identifies the message's sender, otherwise it is left blank. PartnerID identifies the business partner an individual message or envelope is addressed to. ServiceID references the remote service a message or envelope should be sent to; more information about how to transfer data to a particular remote service can be found in table Services. Field DocuType identifies the message format of a message or envelope, e.g., EDIFACT or ANSI X.12. Naturally, all messages in an envelope share the same format. DocuDir contains the direction of envelopes, i.e., if it is inbound and has been received or outbound and will be or has been sent. Field DocuStatus indicates if an entry needs processing or has already been processed, and if the processing resulted in any error.

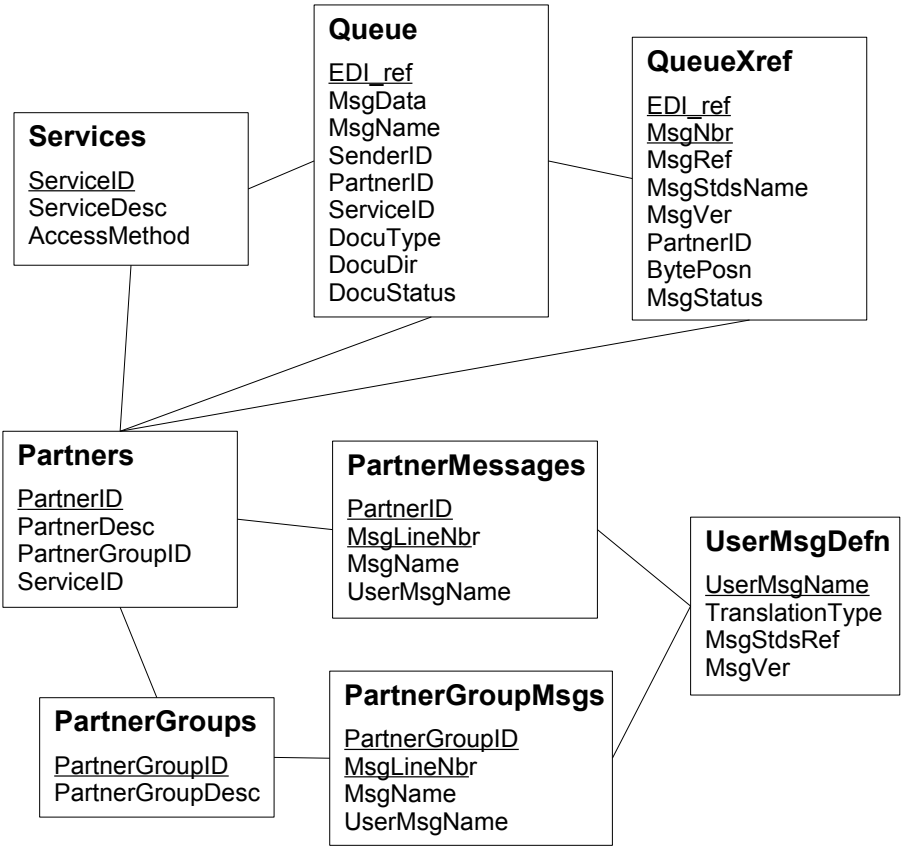

Fig. 2. General structure of the EDIS data model

Table QueueXref contains data about all messages that are stored in one of Queue's envelopes. It also contains field EDI_ref, which refers to an envelope in Queue, and a field MsgNbr, which is the running number of the respective message within that envelope. Together, EDI_ref and MsgNbr form the primary key of the table. MsgRef contains a message reference number that is used to uniquely identify messages of a particular business partner, which is important for detecting duplicates. It helps to make message reception idempotent, i.e., makes sure that messages which are received twice are only processed once. MsgStdsName and MsgVer contain information about the standard and the version of the standard the message adheres to. PartnerID is a foreign key to table Partners and identifies the message's creator. Since each message in QueueXref is embedded in the BLOB field MsgData of an entry of Queue, we also keep track of the starting position BytePosn of each message within the BLOB. Field MsgStatus indicates whether this message has not been or has already been decoded, and if any errors occurred during the decoding process.

3.2 The Scheduler

Many of the scripts of the system are executed in regular time intervals. The part of the system that manages such time events is the scheduler. The scheduler organizes time events in so-called process lines that are specified in the following manner: it is possible to set the days of the week on which an event should occur, the time of day the event should first be sent, the time of day the event can be sent at last, and the frequency in which the event should be rescheduled after having been sent. These parameters allow it to specify the time pattern in which e-commerce usually takes place. All data about scheduling and current events is available in the system's relational database and can be read by other programs, e.g., for monitoring purposes.

3.3 Transport Protocol Modules

Transport protocol modules are part of the system that take care of a specific type of message transportation. As we have mentioned earlier there are different standards and non-standards for the different functions of a B2B message exchange system, and a good system has to be able to support all of them if required. An important business partner might just adopt a new standard, and the system has to adjust to this. This is why the transport subsystem of EDIS has a modular structure that decouples message transportation from other parts.

A module can support sending of messages, reception, or both. It is activated by a system event, which is usually either a timer signal or the arrival of a message. The sending of messages is always triggered by a timer signal, whereas the reception can be either triggered by a timer signal or message arrival. Modules that are triggered by timer signals are usually implemented as scripts, which are interpreted by the B2B system. Modules that react directly to message arrivals are usually implemented as server extensions to, for example, the MS Internet Information Server (IIS). This diversity is necessary in order to deal with the different ways a message can be sent and received, of which we will describe a few.

One way to transport messages is to simply use the email infrastructure. Email communication can be secured, for example, by combining it with transport layer security (TLS), and authentification can be established by using digital signing as described in the S/MIME standard. The transport protocol module is triggered by a timer signal in regular intervals, and each time the module is started, it connects to the respective mail server, retrieves the new messages, removes the mail specific data from them and stores them into the message queue. The messages are marked as unprocessed inbound messages. Besides for retrieving messages form a mail server, the module also checks the message queue for outbound messages that are to be sent to business partners that use email as transport protocol. Once these messages have been delivered to the respective mail server, they are marked as sent. One of the advantages of this architecture is that existing email infrastructure can be used. Rather than having to implement its own mail server, the system can interface to existing ones and thereby make the actual message transport independent of other tasks particular to B2B.

A transport module similar to the one that uses email transfers the messages to and from a business partner's file system using the file transfer protocol (FTP). This protocol can be secured, for example, by tunnelling it over the secure shell protocol. The module is executed in regular time intervals, looks for new files on a business partner's file system and for new messages in the message queue, performs the appropriate file transfer operations and updates the message queue. Although FTP is a standardized protocol, the exact transport process via FTP usually varies between business partners: business partners have their own rules for the file names and file locations particular message types have to be stored to. Consequently, such kind of modules have to be adjustable. This is achieved through the relatively high abstraction level of the scripting language in which these modules are written. Again, the system profits from the fact that it can use any FTP server for message transport.

The module for handling inbound AS2 communication [3] is an example for a module that is executed on arrival of a message. AS2, short for application statement 2, is a business message transport standard proposed by the IETF. It encapsulates message data in a MIME or S/MIME envelope and sends it via HTTP. This is why the AS2 reception module is implemented as a MS IIS HTTP handler. Whenever an AS2 message arrives, the handler is called and stores the received data into the message queue. Since the message queue is implemented on a transactional database management system, it does not matter how many such handlers are working concurrently. The database management system takes care of all the concurrency issues.

Modules handling outbound communication can access the system's relational database in order to get the parameters that are need to send a message. Table Services, which is illustrated in the top left corner of the data model in Fig. 2, contains an entry for each remote service a message can be send to. Field ServiceID is the table's primary key and is used to associate the service with any message in table Queue that should be send to it. ServiceDesc describes each of the available services. Field AccessMethod and other fields, which we will not mention further, provide the necessary technical details for sending messages to the respective service provider.

3.4 Message Analysis

In regular time intervals the analysis module is run. It searches the message queue for incoming envelopes that have not been processed yet. The analysis module parses each message in such an envelope and determines what format the messages are stored in and if the messages are well-formed. If a message is well-formed, some of its basic properties are extracted, like its sender, its receiver, the message type, e.g, if it is an invoice or purchase order, and its unique reference number. These data are used in order to create a record in table QueueXref that describes the respective message of the envelope. In order to decode a message and take appropriate action, we have to make use of the data about our business partners and the messages they send that is stored in the system's data base. The tables involved here are illustrated in the bottom part of Fig. 2.

Once a message has been analysed we need to start a script that handles its further processing. Each kind of message is handled by a particular script, which, in the case of incoming messages, either decodes the message or redirects it. Decoding scripts and redirection scripts are described in Sects. 3.5 and 3.8, respectively. For each message kind there is a record in table UserMsgDefn, which contains a reference to the script that should be used. In field TranslationType this record contains the type of the script, e.g., encoding or decoding, and in fields MsgStdsRef and MsgVer the type of the processed message is described.

For choosing the right script for an incoming message, we use its type and its sender, both of which have been extracted already. Table Partners contains an entry for each business partner messages are sent to or received from. Each business partner supports a well-defined set of message kinds that they can send or receive. The data model allows us to map a type and a sender, which has an entry in partners, unambiguously to a script. One way to define this mapping is to modify table PartnerMessages, which can arbitrarily associate entries in table Partners with entries in table UserMsgDefn. Field PartnerID is a foreign key to table partners; field UserMsgName is a foreign key to table UserMsgDefn. The fields PartnerID and MsgLineNbr form the primary key of the table, with MsgLineNbr being a running number for all the kinds of messages a respective business partner supports.

It would be possible to link all business partners with the message kinds they support using table PartnerMessages. However, usually there are groups of business partners which communicate amongst themselves using a common standard and therefore use the same set of message kinds. In order to simplify the relation between partners and message kinds and make them easier to maintain, the data model supports this concept of groups of business partners by providing tables PartnerGroups and PartnerGroupMsgs. Field PartnerGroupID of table Partners is a foreign key to table PartnerGroups, which contains a description of every group of business partners. It allows to associate a business partner with a group and join groups with entries in table PartnerGroupMsgs. Table PartnerGroupMsgs follows the same pattern as table PartnerMessages, only that UserMsgDefn records are associated with records in PartnerGroups. The advantage of business partner groups is that changes to the message kinds of a group affects all group members, thus preserving compatibility of their communication.

3.5 Decoding Scripts

As we have described in Sect. 3.4, decoding scripts are chosen and executed by the analyser when inbound messages are processed. A decoding script parses a message, extracts all important information and stores this information into the database of an ERP system. In order to perform the translation of messages into ERP records, decoding scripts have to be aware of the message's syntax and semantics as well as the structure of the ERP system's data base. Hence, writing such scripts is not a trivial task, and correctness of such scripts is very important.

A scipt's capabilities, i.e., the operations it is allowed to perform, should be minimal. This way we can avoid many errors and detect some unwanted behaviour. One way to realize this is to configure the interpreter accordingly: it can control the access of the script to other modules while it is running. Another way to restrict access and thereby increase safety is to use the access control mechanism of the ERP system's database: a script should only have write access to those tables that it really needs to modify.

Last but not least, it is very important to handle errors in scripts appropriately. If an error occurs, the error has to be logged and possible modifications that have already been made by the scripts have to be undone. A script has to be *atomic*, i.e., either complete successfully or have no effect at all except on the system's logs. The general structure imposed on the scripts satisfies all these requirements. If a script produces a significant amount of errors, something in the system is most probably wrong. It may be that the script is erroneous, but it may also be that a business partner changed their messages without the system being adjusted to the change. Whatever the reason, such a script will be singled out and put into quarantine, and the errors are automatically reported to the person responsible for the respective kind of message.

3.6 Search Scripts

Search scripts are run by the scheduler in regular time intervals. They search parts of an ERP system's database that contain information which has to be sent to other business partners, like, for example, invoices or purchase orders. For that they run a database query and keep the query results in an ephemeral todo-list, which contains some basic information about every ERP record that is to be sent. When the todo-list is complete, the search script uses the data in the todo-list, like intended message recipient and message type, in order to select a script that can encode the the data into a message. This is analogous to the task of the analyser of selecting a decoding script, which was described in Sect. 3.4, and makes use of the same database tables. Encoding scripts are described in Sect. 3.7. A search script finishes when all todo-list entries have been processed by appropriate encoding scripts.

3.7 Encoding Scripts

An encoding script is activated when a search script finds new data in an ERP database that needs to be sent out of the system. The encoding script is given the location of the ERP data it has to encode, extracts the required information from the ERP database and assembles a new message that is written into the queue. As soon as an appropriate transport protocol module is run by the scheduler, the newly created messages are sent.

In order to keep track of the ERP records which have already been encoded and avoid multiple encoding of the same record, encoding scripts use additional database tables. These tables can either be part of the messaging system's or the ERP system's database. The advantage of keeping them in the messaging system is that ERP and messaging system are decoupled more and the risk of

interference between the systems is reduced. The advantage of keeping them in the ERP database is that the search query of a search script, which must only return unsent records, can be executed more efficiently.

Encoding scripts are subject to the same safety requirements as decoding scripts, which were described in Sect. 3.5. The capabilities such a script has have to be adjusted carefully, e.g., the script should only be able to access those parts of the ERP system that are needed. Like decoding scripts, encoding scripts have to behave atomic and must handle and report errors carefully.

3.8 Redirection Scripts

Like decoding scripts, redirection scripts are started by the analyser after a new inbound message has been analysed. However, instead of decoding the message and storing it into an ERP system, the message is decoded and afterwards encoded as a new outbound message. The encoding creates a new entry in the message queue that will be found and sent by one of the transport protocol modules. This makes it possible for the system to act as a hub in a network of business partners. It can mediate and translate the message exchange between business partners who use different messaging standards, thus allowing them to communicate without changing their system.

3.9 Maintenance Scripts

Another category of scripts is that of maintenance scripts. These scripts are usually called by the scheduler in regular time intervals and automatically perform maintenance tasks which are important for the system. A maintenance script might, for example, backup and archive all system data or produce a report on the system's recent activity.

4 Tool Support

Because the creation of translation scripts like encoding and decoding scripts is a skilled task that consumes considerable time when done manually, the messaging system contains an integrated development environment (IDE), EDIS map, that can reduce the development time of such scripts drastically. By automating large parts of the actual decoding and encoding of messages, EDIS map avoids many potential programming errors and makes the development of translation scripts easier and safer. The IDE supports old messaging standards like EDIFACT and ANSI X.12 as well as newer XML-based message standards.

EDIS map facilitates the creation of translation scripts in several ways. It offers a set of templates for standard translation tasks that can be easily modified and adapted according to the business rules of a particular business partner. New message translation scripts can be created by importing sample messages or message schemas. The IDE makes meta information of several target database systems like, for example, MS SQL Server and Borland Interbase, accessible, so that scripts can easier be programmed to use these database systems. The

tool integrates documentation of messages and can export documentation about message mappings in human readable form. Furthermore, it can automatically generate documentation suitable for regression testing of scripts.

Rather than dealing with a single monolithic script that does all the work involved in processing a message, we associate code snippets to message segments that merely process the data in the respective message segment. Individual segments of a message can be examined, specified and documented. This approach results in a natural decomposition of the translation process.

EDIS map also contains the usual features of advanced IDEs. It supports automatic formatting and syntax highlighting of code and message data, a context-sensitive help and automatic code completion. Syntax checking is performed and syntax errors are reported immediately; also runtime and compile-time errors are reported within the IDE. The integrated debugger allows to trace the execution of scripts in single steps.

5 Related Work

There exist theoretical models for the description of messaging systems, which can be applied to the B2B context. One such model is described in [4]. It is possible to describe messaging as done by the EDIS system with the data type interchange models delineated in this work. In the terminology of [5], EDIS provides the technological means for data exchange in interorganizational relationships between business partners. It is mainly used for relationships governed by a market, although it is also possible to use it for relationships governed by a hierarchy or a hybrid of both. EDIS is not limited to dyadic or "hub and spoke" type relationships, but can be applied to organization networks as well. It provides all the functions of a B2B engine as described in [1] as well as some additional B2B integration functions, like integration of ERP systems.

There are various B2B systems on the market that offer capabilities similar to those of EDIS. One of them is MS BizTalk [11]. BizTalk offers B2B functionality similar to EDIS, although it intends to perform not only B2B messaging but also enterprise application integration (EAI) and, most of all, business process management (BPM). It has been described, for example, how BizTalk can be used in order to manage B2b contracts electronically [7]. Whereas EDIS focusses on enabling business partners to communicate, BizTalk also tries to define and execute high-level programs, so-called "orchestrations", which are supposed to express business processes. These programs can be edited in a visual form and are equivalent in expressiveness to the programming language BPEL [12], which claims to achieve a higher level of abstraction that is closer to real business processes by focusing on "programming in the large". Regarding the original intention of business process modelling as described, e.g., in [13], it is, however, arguable whether orchestrations can really reach to that level, or if they rather just describe the business logic. Such an approach is not inherently more appropriate than the scripting approach chosen in EDIS. The overall architecture of BizTalk is similar to EDIS: the system contains modules for handling different

transport protocols, which are called "adapters", and messages are stored in a central relational database, which is called "message box". A central component, the "orchestration engine", executes and feeds messages into orchestrations according to the message's properties and perform further processing.

A main difference of BizTalk to EDIS is that all messages in the message box are stored in some format based on XML. This can be explained by the point of view that XML inherently provides added value for B2B, as it is also expressed in different academic publications, e.g., [6] and [16], and by Microsoft's current technological strategy. Consequently, incoming and outgoing messages need to be translated to and from XML in so-called "pipelines". For different message formats we need different sets of receiving and sending pipelines. While the usage of XML as a common intermediate format helps to standardise and reduce modules preforming translation tasks and makes it possible to handle message content in a common way, it also introduces the need to translate between different XML formats since XML is not a well-defined message format in itself. Therefore, Biztalk comes with a CASE tool called BizTalk Mapper for creating XSL transformations [18] between different XML schemas. This is slightly similar to EDIS map, but in contrast to BizTalk Mapper, EDIS map supports the translation between essentially different data formats, not just between different flavours of XML, which is a more difficult task.

Another important difference to EDIS is that BizTalk has a very large footprint and depends on various other Microsoft products. Integration with other products may on the one hand provide the user with more functionality and may enrich the way a user can interact with a system, but on the other hand, dependencies between products force customers to spend money on all the required products and binds them to the respective company.

BizTalk, like many other products, make a big point of claiming that they deliver service oriented architecture (SOA) [10]. In general, SOA describes a software architecture for enterprise systems in which components are distributed in a network and can use each other by utilizing a common remote function invocation mechanism. Each component, which is also called a "service", performs a well-defined business task and can be implemented with arbitrary technology as long as it provides the same network interface as the others. When most companies speak about SOA, they refer to the very particular remoting technology of web services, the heart of which is the simple object access protocol (SOAP) [17]. In the context of B2B, web services are just one possible way of many for business partners to communicate; consequently, web services is just one of the transport protocols a B2B system like BizTalk or EDIS can provide in order to fit into a SOA. The software architecture of the BizTalk or EDIS systems themselves is usually not SOA, but rather a structured, component-oriented and non-distributed one. In the case of EDIS, the system consists of modules that mostly interoperate asynchronously using the message queue, which is accessed by SQL. Other commercial systems which follow the trend of SOA and have B2B capabilities similar to EDIS are, for example, IBM WebSphere MQ and Cordys.

There are many studies describing the complexity of the implementation of B2B systems in various companies; see for example [2]. Taking into account past experiences with B2B, it is questionable whether a new B2B software can really revolutionize the way electronic business is done. One should mark that that the ways of electronic business are usually subject to evolutionary – not revolutionary – change.

6 Conclusion

We described the EDIS B2B messaging system, its overall architecture and the design of its important components like, for example, the scheduler, the message analyser and the message queue. We also described EDIS map, which is a CASE tool for creating translation scripts for EDIS. Besides pointing out the general requirements of a B2B system, we also compared EDIS to other popular systems with B2B messaging capabilities. While many other products claim to have a significant impact on a business by offering business process engines and promoting service oriented architecture, EDIS is not intended to promote or change any architectural principles. It rather makes a point of not being prescriptive and enable different companies to communicate without interfering with their business processes.

References

1. Christoph Bussler. The role of B2B engines in B2B integration architectures. *SIGMOD Rec.*, 31(1):67–72, 2002.
2. Caroline Chan and Paula M.C. Swatman. Management and business issues for B2B eCommerce implementation. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. IEEE Press, January 2002.
3. D. Moberg and R. Drummond. RFC4130: MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2). RFC, July 2005.
4. Dirk Draheim and Gerald Weber. *Form-Oriented Analysis - A New Methodology to Model Form-Based Applications*. Springer, October 2004.
5. Wafa Elgarah, Natalia Falaleeva, Carol C. Saunders, Virginia Ilie, J. T. Shim, and James. F. Courtney. Data exchange in interorganizational relationships: review through multiple conceptual lenses. *SIGMIS Database*, 36(1):8–29, 2005.
6. Wilhelm Hasselbring and Hans Weigand. Languages for Electronic Business Communication: State of the Art. *Industrial Management & Data Systems*, 101(5): 217–226, 2001.
7. Charles Herring and Zoran Milosevic. Implementing B2B Contracts Using BizTalk. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. IEEE Press, January 2001.
8. Paul Kimberley. *Electronic Data Interchange*. McGraw Hill, 1991.
9. Chang E. Koh and Kyungdoo Nam. Business use of the Internet: A longitudinal study from a value chain perspective. *Industrial Management & Data Systems*, 105(1):82–95, January 2005.
10. Mircosoft Inc. BizTalk Server 2004 Architecture. Whitepaper, December 2003.

11. Mircosoft Inc. Understanding BizTalk Server 2004. Technical Article, February 2004.
12. Organization for the Advancement of Structured Information Standards. Web Services Business Process Execution Language Version 2.0. Working Draft, May 2005.
13. August-Wilhelm Scheer. *Aris: Business Process Modeling*. Springer, 2000.
14. Arie Segev, Jaana Porra, and Malu Roldan. Internet-based EDI strategy. *Decision Support Systems*, 21(3):157–170, 1997.
15. Aaron E Walsh. *UDDI, SOAP, and WSDL: The Web Services Specification Reference Book*. Pearson Education, April 2002.
16. Tim Weitzel, Peter Buxmann, and Falk von Westarp. A Communication Architecture for the Digital Economy - 21st Century EDI. In *Proceedings of the 33th Annual Hawaii International Conference on System Sciences*. IEEE Press, January 2000.
17. World Wide Web Consortium. SOAP Version 1.2. Recommendation, June 2003.
18. World Wide Web Consortium. XSL Transformations (XSLT) Version 2.0. Working Draft, April 2005.