Enterprise Service Bus at Comenius University in Bratislava

Pavol Mederly [1 2], Gustáv Pálos [3]

[1] Information Technology Centre, Comenius University, Šafárikovo nám. 6, 818 06 Bratislava, Slovak Republic, mederly@rec.uniba.sk [2] Faculty of Informatics and Information Technology, Slovak Technical University, Ilkovičova 3, 842 16 Bratislava, Slovak Republic [3] Information Technology Centre, Comenius University, Šafárikovo nám. 6, 818 06 Bratislava, Slovak Republic, palos@rec.uniba.sk

Keywords
Enterprise Service Bus, Enterprise Application Integration

1. EXECUTIVE SUMMARY

Comenius University in Bratislava has been making efforts to integrate its IT applications since 2004. After researching technical options for proceeding, in late 2006 we acquired a commercial Enterprise Service Bus (ESB) product as a cornerstone for our integration efforts. We chose Progress Sonic ESB. Having been working with this product for more than a year we can now try to look back and summarize some of the major experiences.

1.1. Enterprise Service Bus Characteristics

The ESB can be shortly characterized as a middleware tool used to facilitate communication of applications and/or services in the enterprise. More specifically, it (1) enables loose coupling of interacting systems and (2) can break up the integration logic itself into distinct easily manageable pieces.

Many of ESBs today are based on standard messaging products. Sonic ESB is no exception – it is built on top of Sonic MQ that provides reliable asynchronous communication. Sonic ESB brings around advanced concepts like abstract endpoints, services, processes, service containers and connectivity based on XML Web Services protocols.

1.2. Use of ESB at Comenius University

In the first year we were using ESB primarily to integrate our Central Database of Persons with other systems at the University – those providing data to this database (Students Records, SAP Human Resources) as well as systems receiving data and event notifications from it. ESB features used intensively in this case are reliable message-based communication as well as protocol, format and message content transformations. The paper describes technical details involved in more depth.

1.3. Experiences and future plans

Main benefit of using ESB seems to be the flexibility of resulting integration solution: by dividing it into a set of distinct elements we can reconfigure and/or change it relatively easily. Other positive attributes of the system are its reliability, manageability and good monitoring capabilities.

Although the documentation coming with Progress Sonic ESB is extensive and well-written we found the vendor consulting services indispensable.

We plan to continue using the Enterprise Service Bus in order to intensify the integration of our Information Technology systems.

2. INTRODUCTION

Comenius University in Bratislava is the largest university in Slovakia. It has more than 30 000 students and 5 000 staff members. It consists of 13 faculties and some other constituents, e.g. two big hostels, Centre for Continuing Education, Academic Library and Information Technology Centre.

University's information system consists of many systems, most important being summarized in the Table 1 below.

Table 1: The most important Information Technology systems at Comenius University

System description	Source	Since
Student Records	developed in-house, currently being replaced by a packaged product	1991
Finances and Human Resources	packaged product (SAP R/3)	2005
Virtual Library (library automation, publications)	packaged product (VTLS Virtua)	1999
Access control system (access to protected areas based on university identity cards)	packaged product	2004
University card terminals (used to update data on university identity cards)	packaged HW/SW product	2004
Internal VoIP[1] network management and billing	developed in-house	2005
Information Technology support: data about computers, users and service incidents	external custom development, currently being replaced by a packaged product	2005
Accommodation	external custom development	2006
Management of applications for accommodation	developed in-house	2008
Internal training courses management	developed in-house	2007
Canteen ticketing system	packaged product	2008
Common authentication system	combination of standard tools (Kerberos, CoSign)	2008

These systems along with other smaller ones not mentioned here are generally not integrated into a coherent whole. We are making efforts to do so with the intention of achieving following goals:

1. To **make university data "compatible"**, that is, to make cross-system queries possible – for example, "how many publications in category C were written by members of department D positioned as teaching assistants?" Such a query is not possible to run without substantial human effort today, because one part of the data - namely publications of employees - is stored in the Virtual Library system, where people are identified by this system's own identifiers (e.g. vtls00000374) while another part – namely positions held by people at departments - is stored in Human Resources system, where people are identified in a different way (by numbers such as 10105389). The codes for departments differ as well.
2. To **automate university's business processes** or at least to make them **more efficient** e.g. by eliminating duplicate data entry. For example, when a student wants to check-in at the hostel, the process takes unnecessarily long time because the data about him or her has to be entered into the Accommodation system. If the system were integrated with Student Records, the process could be much quicker because majority of the data would be already in place. The same can be said about entering (and updating!) student and employee data into Virtual Library, Access control system, Canteen ticketing system and others.

[1] VoIP (Voice over IP) is a set of technologies that allow making phone calls through standard computer networks.

While integrating IT systems, there are some basic technological requirements to be met:

1. The systems being integrated have to be **loosely coupled**; it means that each one assumes only minimal information about the other(s). This is absolutely necessary in order to allow them to evolve independently – or better said, to evolve at all.
2. The integration structure itself has to be **very flexible** (1) to allow the necessary evolution of integrated systems; and (2) to enable to change business processes it helps to automate easily.
3. The integrated system has to be **reliable** in the face of inevitable network, servers or applications outages.

We are currently in the process of IT systems integration at Comenius University. We have achieved goals mentioned above for some types of data and processes, mainly for those connected with personal data about students and employees. Description of the technical infrastructure for integration we used, the Enterprise Service Bus, is the main topic of this paper.

The rest of this paper is organized as follows: In part 3 we outline an integration scenario as an example to show basic integration approaches we have used. In more technically oriented part 4 the Enterprise Service Bus itself is depicted. Part 5 provides our experiences with this technology. Part 6 concludes this paper.

3. AN EXAMPLE INTEGRATION SCENARIO

We illustrate the idea of various approaches to integration on the following scenario implemented at Comenius University in years 2004 – 2008.

Almost every IT system at our university needs to work with data about persons – students and employees. This data is entered into Student Records and Human Resources systems, respectively; or "source systems" for short. The main issue is how to collect this data, consolidate it (e.g. correlate data about one person even if he or she studies at and/or works for more faculties simultaneously) and distribute it to systems that need it; or "destination systems" for short. The situation is depicted in figure 1.

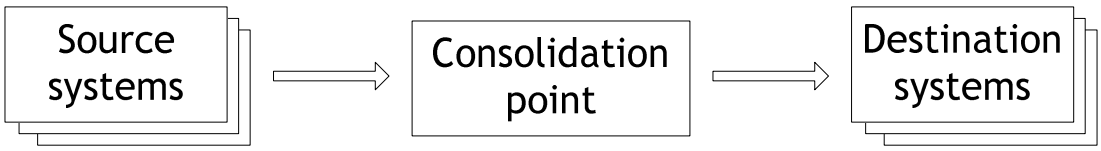

Figure 1: Outline of the integration scenario

The main complication is the fact that data about students is distributed throughout 13 databases managed at the faculty level. The databases are not always available because some of them are stored on standard workstations that are switched off during nights and weekends. Back in 2004 some of the workstations were not connected to any network at all for security reasons.

The data about employees was originally distributed in a similar way. Fortunately, it was consolidated and stored in SAP R/3 system in the autumn 2007.

In order to solve this integration problem we have devised following two auxiliary components:

1. For the on-line storage of data about persons we use component called **Central Database of Persons** (or PersDB for short) that regularly gathers data from source systems and provides it to destination systems. Technically it is a standard database with appropriate application programs for importing and exporting data.
2. In order to uniquely identify each individual we have introduced University Personal Identifier – a number assigned to each person irrespective to kind and number of relation(s) with the university he or she has. These identifiers are generated and maintained by a component called **University Personal ID Generator** (or IDGen for short). ID generation is an operation typically done only once for each record about person in each source system: after creating an ID it remains assigned to that record for its lifetime. This process is semi-

automatic because it needs a human operator to resolve ambiguities that arise when comparing (potentially imprecise and/or erroneous) data about persons.

Main components of the solution are shown in figure 2.

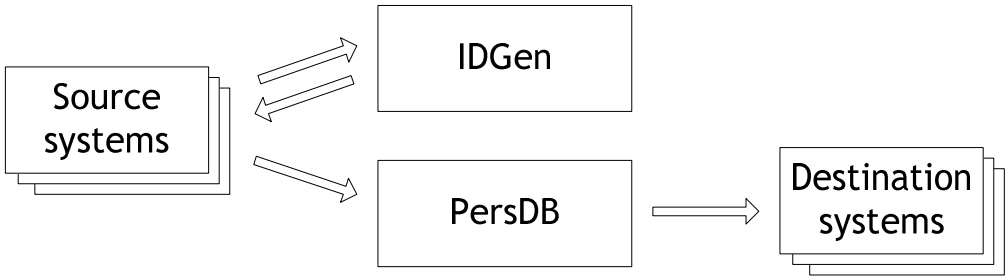

Figure 2: Main components in the integration solution

3.1. First approach: Manual file transfer

First – and very simple – technical solution we created in 2004 was based on manual transfer of data files. Staff in faculty students' offices regularly invoked Student Records system's function "export student data to a file" and sent the resulting file to an operator via e-mail. The operator then imported the data into PersDB. He also regularly ran scripts that exported the consolidated data into various destination systems.

As an aside, the most time-critical among destinations were University card terminals during enrolment period. It was necessary to update data about individual student within one working day after the moment of his or her enrolment.

Because the Personal IDs for students had to be generated only couple of times a year (at least we originally thought so) this process was realized manually by the system administrator as part of routine system maintenance.

Situation is shown in figure 3. For simplicity, it depicts only the transfer of students' data for one faculty.

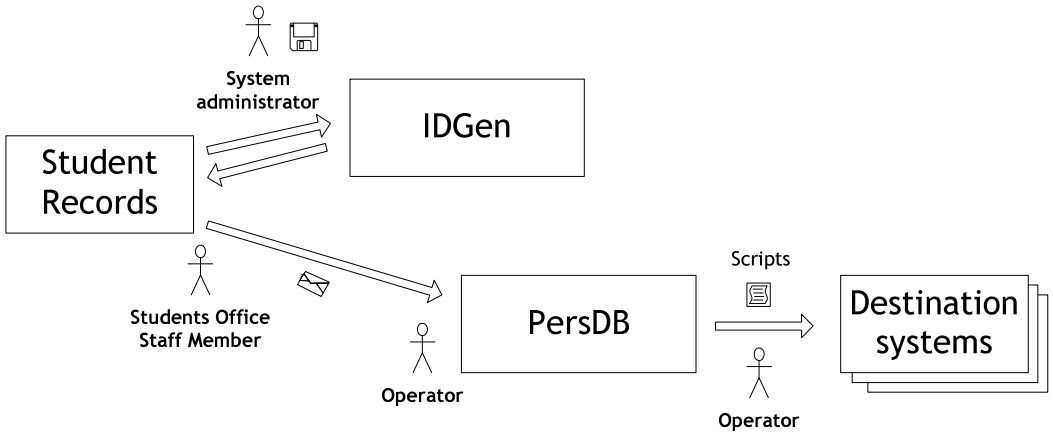

Figure 3: A simple integration by manual file transfer

The main and obvious disadvantage of this solution was that it required frequent manual activities. At the other hand, it was very simple and easy to implement. For more information about integrating systems by transferring files see the "File Transfer" integration style in (Hohpe & Woolf, 2004).

3.2. Second approach: Asynchronous messaging

When preparing for next year's enrolment period we investigated various options how to automate the whole process. One of them was to merely automate transfer of files mentioned above; the other was to write data from Student Records directly into PersDB (see also the "Shared Database"

integration style (Hohpe & Woolf, 2004)). Because what we really needed was a bi-directional asynchronous communication between PersDB/IDGen and Student Records systems, we concluded the best solution would be to use messaging (see "Messaging" style in (Hohpe & Woolf, 2004)).

We installed a message broker – an infrastructure tool that provided the possibility to transfer messages reliably from sender to one or more recipients in such a way that after being sent the messages waited in the broker until picked up by the recipient(s).

This process was automatic. Student Records system periodically sent messages containing data into a queue implemented in the broker. PersDB picked up each message, processed it and sent a reply back to appropriate Student Records system. The replies contained some state information needed by the Student Records system and its users.

The export of data to destination systems was done by set of scripts running automatically.

Because IDGen was used only rarely and still required operator intervention it was decided to run it manually by the operator. Its input was data captured by PersDB. Its output (in the form of messages) was sent to appropriate queues and then processed automatically by Student Records system.

Situation is shown in figure 4. For simplicity the communication with destination systems is not shown.

Note that while PersDB is a Java application that can communicate natively with the message broker, Student Records as a rather old Clipper application had to be supplemented by Java adapter performing this communication.

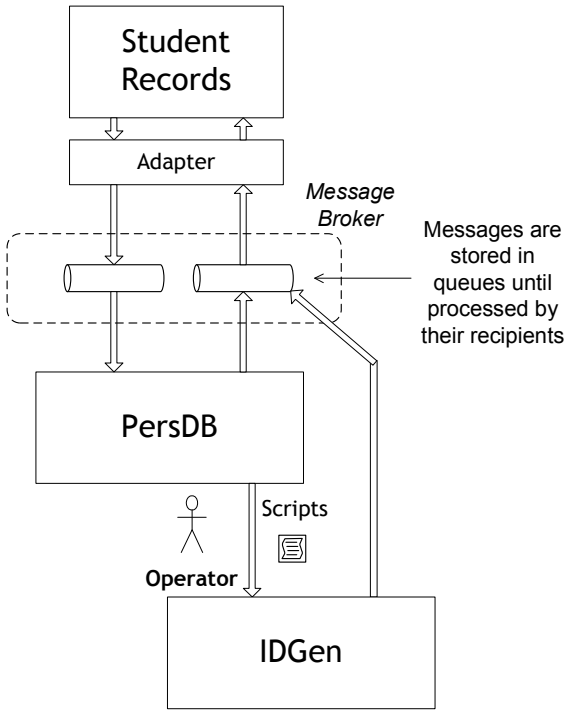

Figure 4: Integration by asynchronous messaging

The solution was almost satisfactory, the only problems being (1) the need to run IDGen manually, (2) the inevitable coupling between PersDB and IDGen i.e. PersDB has to know that it should pass some data to IDGen.

3.3. Third approach: Enterprise Service Bus

The current solution is based on Enterprise Service Bus (or ESB for short). Main value added by the ESB is flexibility: connections among individual components – systems and services – are not encoded

in these components[2] itself but are configured in the ESB. The bus uses this configuration to correctly route messages.

When implementing the ESB we converted PersDB and IDGen into services and put them on the bus. (For detailed explanation see the following part.) In order to run unattended, we split IDGen into two parts: automatic "IDGen service" that is able to process messages not requiring operator involvement and "IDGen operator service" that enables the operator to be involved in the process.

Situation is depicted in figure 5.

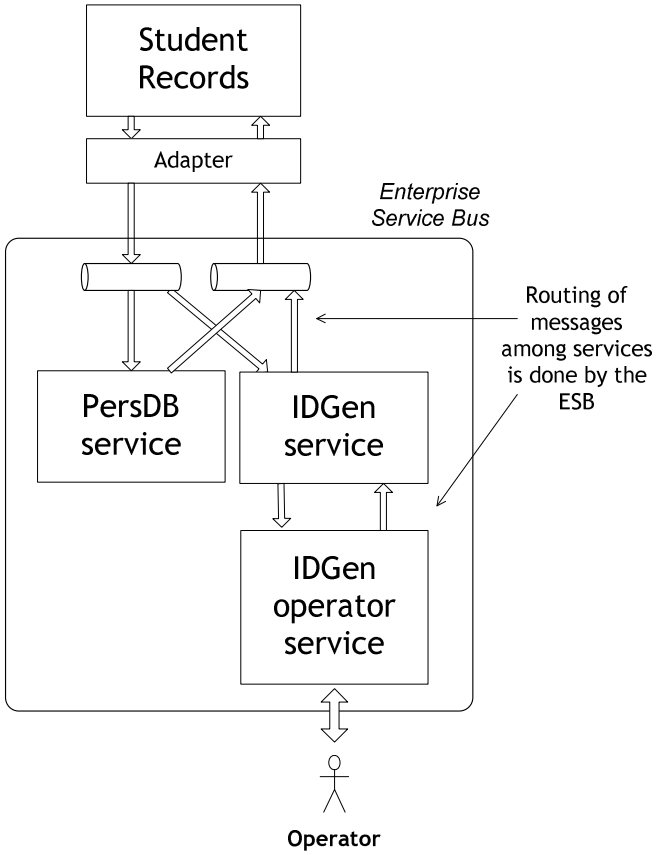

Figure 5: Integration by Enterprise Service Bus

The process is now fully automatic. Student Records system sends data via adapter (basically unchanged comparing with previous solution). As the data enters the ESB, it is routed to PersDB service, IDGen service and to IDGen operator service as needed. The routing is described by the rules configured into the ESB.

In such a system it is possible to easily change integration connections by creating new services and/or modifying the flow of data among them. More details are described in part 4.

4. THE ENTEPRISE SERVICE BUS CHARACTERISTICS

Although there is no generally accepted definition of ESB we can shortly characterize it as **a middleware tool used to facilitate communication of IT systems in the enterprise**. More specifically, it (1) enables loose coupling of interacting systems and (2) can break up the integration logic itself into distinct easily manageable pieces (Papazoglou & van den Heuvel, 2007).

Many of ESBs today are based on standard messaging products. Sonic ESB is no exception – it is built on top of Sonic MQ that provides means of reliable asynchronous communication.

[2] As in case of PersDB "knowing" it has to forward some of the data to IDGen system.

Messages exchanged among applications can contain arbitrary data but typically they are either **command messages** telling the receiver that it has to carry out specific operation, **document messages** indented to pass data to one or more receivers, or **event messages** informing receiver(s) about some specific event that has occurred. For more information see Command, Document and Event Message patterns in (Hohpe & Woolf, 2004).

4.1. Enterprise Service Bus tasks

Main tasks of the Enterprise Service Bus are:

1. (guaranteed) message delivery,
2. message translation,
3. message routing.

First of these tasks is common to standard messaging products and we have covered it already.

Second one, the message translation, can be done at various levels – specifically at the level of protocol, format or content.

Protocol-level translation deals with incompatibilities pertaining to transport protocol: an example can be one of destination systems introduced in Part 3 that is able to communicate only by standard web protocol (HTTP[3], or more specifically, its secure variant HTTPS) but the PersDB itself is communicating by means of messaging (via JMS[4]) only. ESB can deal with this situation by translating between these two protocols. Other typical transport protocols used for communication of IT systems and typically supported by an ESB are SOAP[5], SMTP[6], or simple "file drop"[7]. Many systems communicate by proprietary transport protocols as well.

Format-level translation deals with incompatibilities related to message format. For example the Student Records system sends and receives data in CSV format[8] but the universal IDGen service expects input and generates output in XML[9]. ESB can be configured to translate between these two formats as needed. Again there are many other formats used, many systems having their own proprietary ones.

Message content translation is the most complex form of translation where one or more of the following operations can occur:

1. renaming data items, e.g. attribute "LSTNAM" in a message sent from Student Records has to be renamed to "LastName" in order to be processed by IDGen,
2. adding or removing data items, e.g. if receiving system expects to have attribute "FullName" we should create it (by concatenating title, first and last name),

---

[3] HTTP (Hypertext transfer protocol) is the protocol used mainly by web browsers to retrieve pages from web servers.

[4] JMS (Java Message Service) is a programming interface used by Java programs to send and receive messages through (some) messaging product.

[5] SOAP (originally meaning Simple Object Access Protocol) is the protocol used for communication between XML Web Services and their clients. It is itself carried by HTTP or other protocol.

[6] SMTP (Simple Mail Transfer Protocol) is the protocol used for transferring e-mail between e-mail clients and servers as well as between servers themselves.

[7] File drop means that one system creates a file with the data to be transferred and stores it into specified directory. Then it is picked up by the receiving system or systems.

[8] CSV stands for "Comma Separated Values". It is a text format where each line corresponds to an entity (e.g. a student) and consists of individual attributes (e.g. name, address, date of birth etc) separated by comma (",") or semicolon (";").

[9] XML (Extensible Markup Language) is a simple yet flexible text format used for storing structured data as well as more complex documents. Individual data items are stored as elements and/or element attributes. Due to its flexibility, universal acceptance and multitude of tools for processing it XML is commonly used for IT systems communication.

3. translation of data item values, e.g. Human Resource encodes gender as “1” for male and “2” for female but PersDB expects values “M” and “F”, respectively,
4. more complex translation e.g. aggregating/splitting of data entities, performing computations etc.

Message routing deals with the situations when a message has to be sent to one or more recipients that are either predefined or selected dynamically e.g. based on content of the message. Following rule can serve as an example: “if there is at least one person without University Personal Identifier assigned, route the message to IDGen”.

4.2. Enterprise Service Bus concepts

Now we introduce basic concepts of the Enterprise Service Bus. As the terminology in this area is not unified yet, we will describe the meaning of terms from the point of view of the product we use: Progress Sonic ESB. For more information see (Chappell, 2004) and (Sonic Software Corporation, 2008).

The **service** is a basic functional block accessible through the ESB. Roughly speaking, services can be of two major categories:

1. mediation services – these are typically responsible for message translation, routing and connectivity to external systems,
2. application services - these are providing some application specific functionality, e.g. generating University Personal Identifiers, storing data in Central Database of Persons, extracting data from it etc.

Services can be internal to ESB (that is, running internally in some ESB container, see below) or external – running outside of the ESB.

Each internal service is an instance of a specific **service type** that provides implementation of the service and therefore defines its behaviour, configurable properties and so on. The relation between service type and service can be compared to a relation between class and object (class instance) in object-oriented programming languages or to a relation between component and component instance in component-oriented programming paradigm.

Progress Sonic ESB comes with a set of basic service types (namely for message translation, routing, splitting, joining and so on). It is possible to create own service types of internal services in Java or JavaScript as well as to acquire service types from Progress or other supplier. Of course external services can be created in any programming language; they are communicating with the ESB using one (or more) of the transport protocols mentioned above.

Each internal service takes input messages from an **entry endpoint** and sends responses to one or more **exit endpoints** (in case of normal processing) or to the fault endpoint or rejected message endpoint (in case of problems). Names of concrete entry/exit/fault endpoints are typically not defined by the programmer when creating service the type but specified by the system administrator when instantiating specific service type. When instantiating a service the administrator can specify values of service parameters as well – similar to specifying parameters when creating instance of a component.

Endpoints are abstract names for channels that systems and services use to communicate. In Progress Sonic ESB these channels are usually realized by message queues and topics provided on specific message brokers. Queues and topics can reliably store messages until they are fetched by their recipient or recipients. External services can access these endpoints by means of various transport protocols, namely JMS, HTTP/HTTPS and SOAP. Adapters for other protocols can be created or acquired separately.

Internal services run in **ESB containers** just like e.g. Enterprise JavaBeans (EJB) run in EJB containers on Java Platform, Enterprise Edition application servers. ESB containers provide basic framework for loading, starting and stopping, managing and monitoring services and for communication of these services with the rest of ESB.

Each container is run as an ordinary process containing Java Virtual Machine and can be placed on arbitrary host in the network. For better performance or reliability it is possible to place the container on more than one machine at once.

Progress Sonic ESB provides a possibility to group services into more complex **processes**. Processes control the flow of messages among services. These processes are either centrally managed via the WS-BPEL 2.0-compliant server[10] (an add-on product to Progress Sonic ESB) or decentralized. Although we have some experimental applications running on BPEL 2.0, in production we use decentralized approach mainly because of its simplicity. We plan to start using BPEL in a short future.

Figure 6 shows these concepts in a context of integrating information about persons.

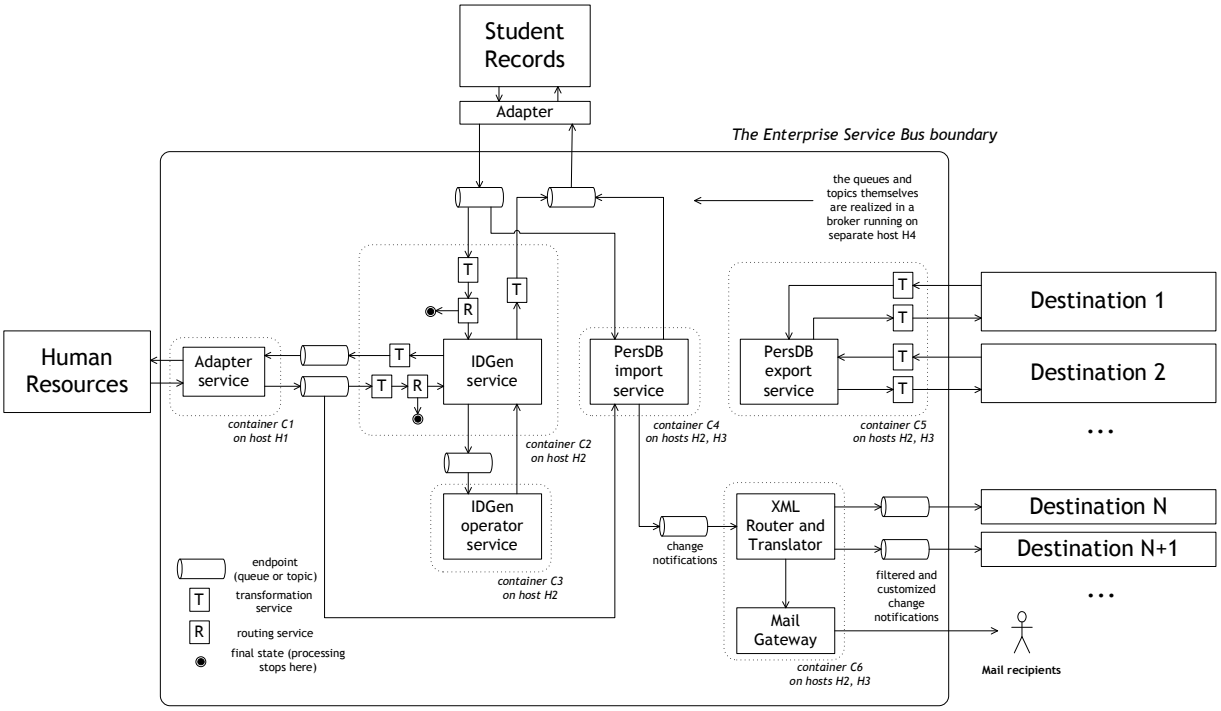

Figure 6: Use of Enterprise Service Bus to integrate information about persons

At the "input side" messages from Student Records (only one instance shown for simplicity) enter the appropriate endpoints in ESB. Subsequently each one is routed according to a predefined process: it is sent to IDGen service (on the way translated as necessary and checked whether there is at least one record without University Personal Identifier) as well as to PersDB import service. The results from IDGen and PersDB are sent back to original sender. IDGen consults IDGen operator service if it needs to. Similar process is followed in case of Human Resources. Due to practical considerations input data are being sent from Human Resources each 15 minutes and from each Student Records instance once an hour (in the enrolment time).

The "output side" is implemented by PersDB export service. As the needs of the destination systems concerning content and format of the data are diverse, there is a pair of transformation services between each destination system and PersDB export service. Each destination system can thus receive data in a customized way.

Individual destination systems get data typically once a day. For some applications this is not adequate so we have implemented a simple notification schema: after a change in data about specific person occurs, the PersDB import service produces a message informing about this event. The message is then passed to XML Router and Translator service where it is compared to a list of

[10] WS-BPEL (Business Process Execution Language for Web Services) is a language for specifying processes consisting of series of XML Web Service calls.

rules. If the message matches one or more rules[11] it is sent to appropriate destinations. Before sending it is transformed according to the need of those destinations, so each system gets filtered and customized notifications. A special kind of recipients is people (instead of IT systems) that get their notifications via e-mail.

The picture is a bit simplified, mainly because:

1. In reality, many transformations (showed as "T" boxes) are composed of more than one elementary transformation. For example, when sending a message from Student Records into IDGen, it is first converted from CSV to XML and after that it is transformed by applying an XSLT[12] stylesheet. The benefit of this approach is that one does not have to create custom transformation service for each situation but can leverage existing (standard) components instead.
2. We have left out some auxiliary services used e.g. for logging of messages.
3. Only the most important endpoints (queues and topics) are shown.

5. EXPERIENCES WITH THE ENTERPRISE SERVICE BUS

5.1. Reliability

Our solution based on Progress Sonic ESB proved to be reliable as expected. Appropriate setting of Quality of Service (QoS) parameters assures messages are not lost in case of network, container or broker failure.

A minor reliability issue is that we experience rare out-of-memory conditions when sending messages larger than default limit of 10 MB, but we expect this to be solved soon by proper broker tuning.

We have bought a single-CPU broker licence meaning we are entitled to run the message broker on one computer only. Should we need to enhance reliability and/or performance, we could create a cluster of brokers with hot-standby and load-balancing features by acquiring appropriate licences.

Containers hosting services can be made redundant with no additional licensing costs simply by setting the configuration.

5.2. Flexibility and maintainability

Strong feature of the approach based on ESB is the flexibility of developed solution. We are able to easily change integration scenarios with respect to

1. types and instances of services used to process messages,
2. mapping abstract endpoints to physical destinations (topics, queues, URLs, …),
3. arranging services into containers,
4. deploying containers onto host machines.

An issue in this area is the complexity of solution created as it can obviously hinder flexibility. Care should be taken to properly separate integration logic from the core business logic and to use the ESB to solve integration problems only. Just because it is technically possible to create almost arbitrary computation does not mean it is appropriate to do so. Wide spectrum of technologies (e.g. process descriptions and BPEL for specifying data and control flow, XSLT, XPath, Java and JavaScript for data manipulation etc) is beneficial because one can choose the best tool in each situation, but the overall solution structure can easily become overly complex. Structuring the solution reasonably as well as choosing appropriate names for processes, process steps, service types, services and endpoints and documenting the solution properly is necessary.

---

[11] like "send me only notifications about new employee hired" or "send me only notifications concerning with address change"

[12] XSLT (Extensible Stylesheet Language Transformations) is a language for specifying transformations on XML documents. It is used extensively in integration scenarios to do content-level transformations.

Although the development environment (Progress Sonic Workbench) provides means to navigate through processes and services, we missed more powerful visualization and browsing capabilities. We are planning to create such a tool ourselves, with the ability to display static structure of the solution as well as some of dynamic aspects of its behaviour (e.g. traces of messages flowing through, processing times for individual services etc).

5.3. Learning curve and ease of use

Learning process was quite long: It took couple of months to be able to effectively use the product but after that time we think we are quite proficient in using it. We have used 4 days of intensive consulting (mentoring) in the form of personal contact and e-mail consultations with the vendor representative. Looking backwards, it seems that a slightly more amount of consulting would be adequate.

After getting acquainted with the product we can say it is relatively easy to use. We regard some of the configuration tasks more complex than necessary, but the situation is getting better as the product evolves.

The documentation is fairly extensive and well-written yet it has some weaker places.

5.4. Manageability and monitoring

We consider the manageability and monitoring a strong point of our ESB. In contrast to monolithic applications, we are able to start and stop individual containers and relocate them to other machines as needed.

We can easily monitor communication among participating systems and services by attaching to any topic and receiving all messages destined to it or by using tracking messages generated automatically as data messages enter and exit individual services.

Other means for monitoring we use are logging (events of interest occurring in containers are logged locally as well as centrally), metrics (e.g. containers and their components can report the amount of memory used, number of messages processed per second etc) and notifications (generated e.g. when some threshold value is exceeded).

6. CONCLUSION

In this paper we described our experiences with Enterprise Service Bus, specifically with the product Progress Sonic ESB that we have been using for more than a year.

The scenario presented in this paper could be regarded as simple and probably not interesting for those that have acquired integrated academic information system and therefore have the specific question of integrating personal data already solved. Yet we consider it to be a good example of the use of the ESB giving some basic ideas what ESB is and how it can be used.

At Comenius University we plan to continue using the Enterprise Service Bus in order to intensify the integration of our Information Technology systems.

7. REFERENCES

Chappell, D. A. (2004). *Enterprise Service Bus*. Sebastopol, CA: O'Reilly Media, Inc.

Hohpe, G., Woolf, B. (2004). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA: Pearson Education, Inc.

Papazoglou, M., van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16, 389-415.

Sonic Software Corporation (2005). *Sonic ESB: An architecture and lifecycle definition*. Retrieved May 2, 2008, from: http://www.sonicsoftware.com/products/whitepapers/docs/esb_architecture_-definition.pdf