Charles University in Prague
Faculty of Mathematics and Physics

SOFTWARE PROJECT

XRouter

Soběslav Benda
Miroslav Cicko
Tomáš Kroupa
Petr Sobotka
Bohumír Zámečník

Supervisor: Mgr. Martin Nečaský, Ph.D.

2011

XRouter Documentation – Index

Charles University in Prague
Faculty of Mathematics and Physics

SOFTWARE PROJECT

‹XRouter›

1. Definition

Soběslav Benda
Miroslav Cicko
Tomáš Kroupa
Petr Sobotka
Bohumír Zámečník

Supervisor: Mgr. Martin Nečaský, Ph.D.

2011

XRouter – Definition

Introduction

In the present time there exist a lot of smaller or mid-sized organizations or companies which need to realize their business processes by integrating existing or new applications or systems. They can utilize integration platforms built on concepts of Enterprise Application

Integration (EAI), ie. integration within a single company, or Business-to-Business (B2B) for integration among multiple companies.

Currently available integration platforms offer good support for the Enterprise Service Bus (ESB) concept, which is considered as one of the most mature concepts of EAI. Such platforms frequently constitute an intergration back-bone in the Service-oriented Architecture (SOA) environment. Those concepts and platforms are often just too complex for many scenarios in common practice. Moreover the complete software solutions are usually quite expensive and thus out-of-reach for the aforementioned target group.

Service-oriented Architecture (SOA)

For both economical and technological reasons there is a tendency to make the IT infrastructure composed of many cooperating software components, each providing a single useful service. Such components communicate with each other by passing messages in a format defined by their interface. In other words each component has an interface describing the format of messages it consumes and produces.

By composing such loosely coupled components it is possible to realize more complex business processes. An architecture with such interconnected components called *services* is usually referred to as a *service-oriented architecture*. Some of the advantages are the following: SOA enables connecting software components implemented in different platforms and programming languages, it encourages reusing existing software and it improves the ability to make changes quickly.

For communication between services the XML language is usually employed as it is good at representing structured information. Components, often developed by various suppliers, may use diverse XML formats, be it standard protocols, such as SOAP, or their own proprietary XML formats.

There are several fundamental ways of integrating systems: sharing files, sharing a database, calling procedures remotely, passing messages. Each way has its advantages and drawbacks. An example of the RPC-style services to be intergrated are Web services – computational entities with a implementation-independent interface accessible via HTTP, a platform-independent transport protocol. The other ways of communication are also quite common.

It is possible to identify paths of messages through the whole network of nodes. A message produced by one node travels through the network, can be processed by other nodes and finally can be consumed by some other node or multiple nodes. Nodes on paths between producers and consumers are commonly referred to as *active intermediaries*. Their job can be routing messages, transforming them, or performing further operations with them.

Eg. when using the SOAP protocol and its WP-Adressing extension an intermediary receives a message and according to the message's metadata it decides which node to pass the message to. Then using a transform it updates the message header.

In order to support SOA creation it is appropriate to have available such simple mediating services capable of routing, transforming and further processing messages using declarative languages.

Transforming a company's IT infrastructure to an ideal SOA cannot be accomplished in a day. Introducing SOA and complex ESB products can pose a non-trivial change in many aspects and can be quite expensive. Usually, a more feasible way is to evolve the present informations systems and progressively realize some intergration solutions utilizing services (eg. Web services).

A case study

Imagine that Foo Energy Co., an electric energy producer, owns a complex information system for trading electricity and it needs to extend it by data exchange with an electricity market operator (OTE). Foo Energy Co. runs entirely on the Windows platform and wants to use software using the .NET Framework.

OTE provides a set of Web service interfaces (clients and services specified by WSDL). Foo Energy Co. buys the license for using the services of OTE but it also needs a software to do the actual integration.

OTE configures its prepared producer of messages (a web client) so that it sends messages to the new customer. So Foo Energy Co. must publish a web interface capable of receiving SOAP messages over HTTP(S). The received messages then have to be fed into the existing information system. For simplicity we presume the message data just need to be stored into an existing database of that IS.

One of the solutions to this problem is to create a Web service according to the WSDL of the OTE's message producer. In C#/.NET environment this is no problem. A Windows Communication Foundation (WCF) service can be created and integrated into the IIS service or a custom Windows service. WCF maps a message to C# object model and the Web service logic is also represented in a C# code. Such a simple receiver service can contain a sequence of steps – receiving a message, transforming it, storing it into database, sending a reply with the result of the operation to the client.

The distribution logic is usually a bit more complex. One producer can send messages of multiple types which need to be recognized. Some messages are important for the Foo Energy Co., some can be ignored. The interesting messages might need to be transformed into a format suitable for the database (eg. according to its structure, element or attribute values, or by some more complex relations). Also when a message is not valid, certain actions must be performed. This can happen eg. when the message format has been updated by the OTE. In this case invalid messages usually need to be logged, archived and a person responsible for updating the system can be contacted by e-mail.

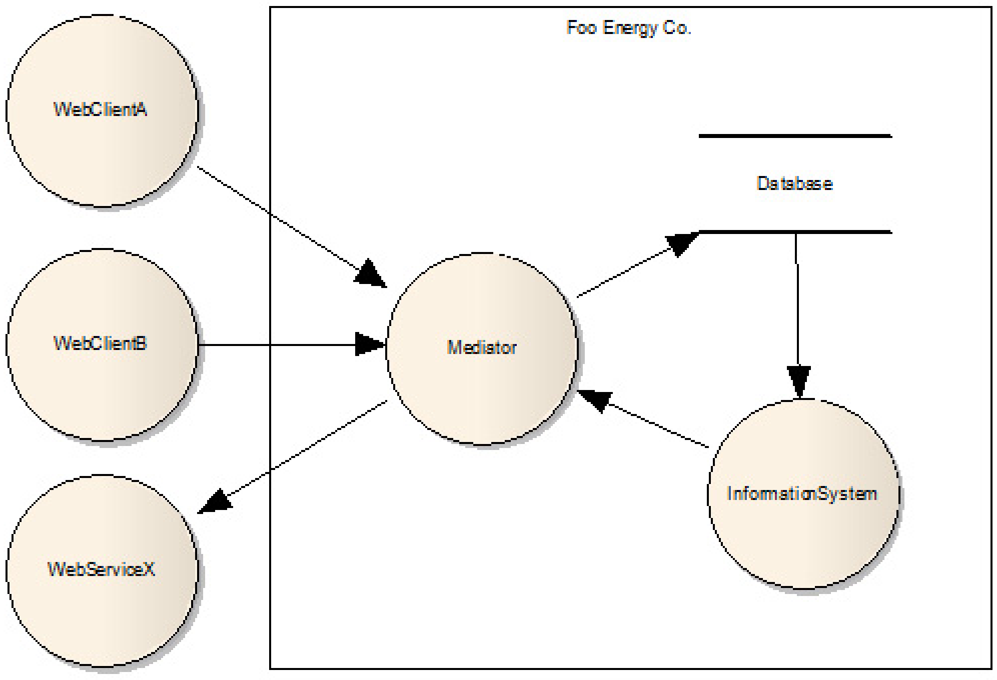
In fact there is usually more than one such an interface, and not only services but also clients (in case of having Web services at the OTE side). Assume that Foo Energy Co. needs to get 10 such interfaces running. After each update of the format an interface must be updated (C# code must be regenerated from WSDL or rewritten and the system rebuilt).

Other problems during solving the integration include the need for making changes in distribution logic efficiently, monitoring the data transfers, organizing XML schemas, etc.

Moreover, in present time new companies are being founded which need to connect to OTE.

Each company may want to change the message formats to provide some specific data it needs. So the formats are changing very often. Also new interfaces are being created. In such a context using WCF for creating Web services is not completely suitable as being too rigid. Frequent changes result in constant need to change the code, rebuild systems and stop them for updating. This all costs a lot of money.

In the described example case study changes to Web service interfaces are quite frequent and it is suitable to use declarative languages understood by trained operators in order to minimize or avoid the need for programmer interventions. For many companies a simple platform handling such not so highly complex cases is appropriate, at a modest price. If it is extensible and it can also be used for mediation it is even better.



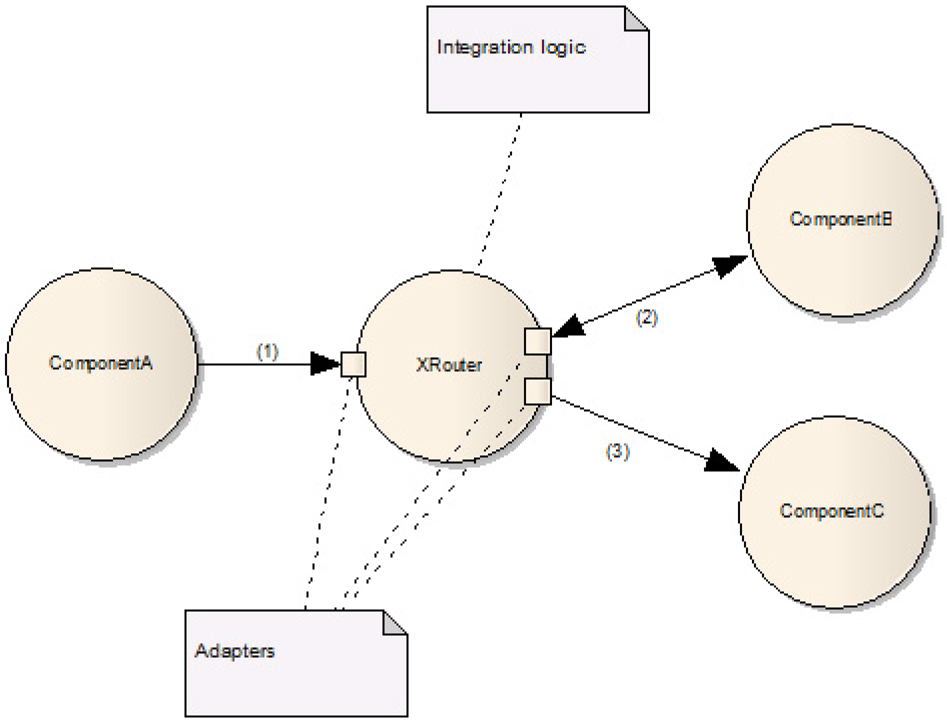
The image demonstrates the usage of an intermediary service – also containing declarative distribution logic (routing, transformations, etc.) – in order to facilitate communication among systems.

What is XRouter

XRouter is a system consisting of a set of tools and services with the goal to provide a flexible support for simpler low-level intergration of applications and for creating a SOA.

XRouter works on top of the .NET Framework and is quite easy to install and operate. Its main benefit is being light-weight, in contrast to comparable products. It is primarily designed to act as a mediator (as described in the case study above), but it can potentially become a part of a more complex architecture.

The main part of XRouter is a message router service. It receives messages, performs some interaction logic (such as validations, transformations, route decisions, etc.) and sends processed messages away. The basis is made of input and output endpoints interconnected by the interaction logic. Endpoints are implemented as adapters which are available in form of plug-ins. One XRouter service instance can contain many adapters of various types, each with possibly multiple logical endpoints.



The image demostrates a simple interaction of the three following components:

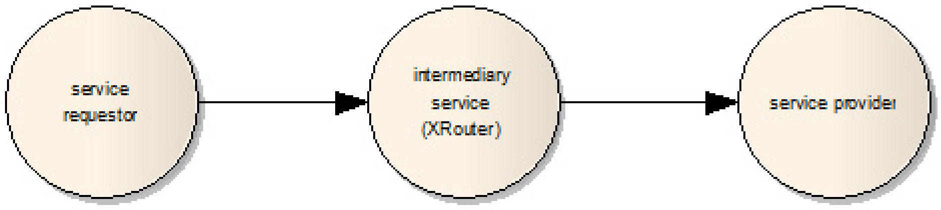
1. We assume that ComponentA generates two types of XML messages (X and Y) as files inside a directory on the filesystem. XRouter receives the messages (when a new file appears it is read and then deleted). During the processing of the messages the message type is recognized and each message is routed accordingly.
2. Messages of type X are routed to the ComponentB. It is an RPC-style Web service (ie. a XML message must be transfomed into the SOAP format) and it returns a reply (of type Z).
3. ComponentC is a Document-style Web service and messages of types Y and Z are routed to it (without any reply).

Besides the routing service XRouter contains some supporting tool for more comfortable monitoring and administration in order to make integrations and solving of issues more efficient.

XRouter in Enterprise Environment

XRouter is able to create the core of an IT ifrastructure supporting SOA and thus enable managing communication and interaction among services, while providing a way to handle errors and exceptions. XRouter can also be readily tailored for creating automatized processes other than integrating applications, eg. automatic filing of documents on a filesystem at an ordinary office PC.

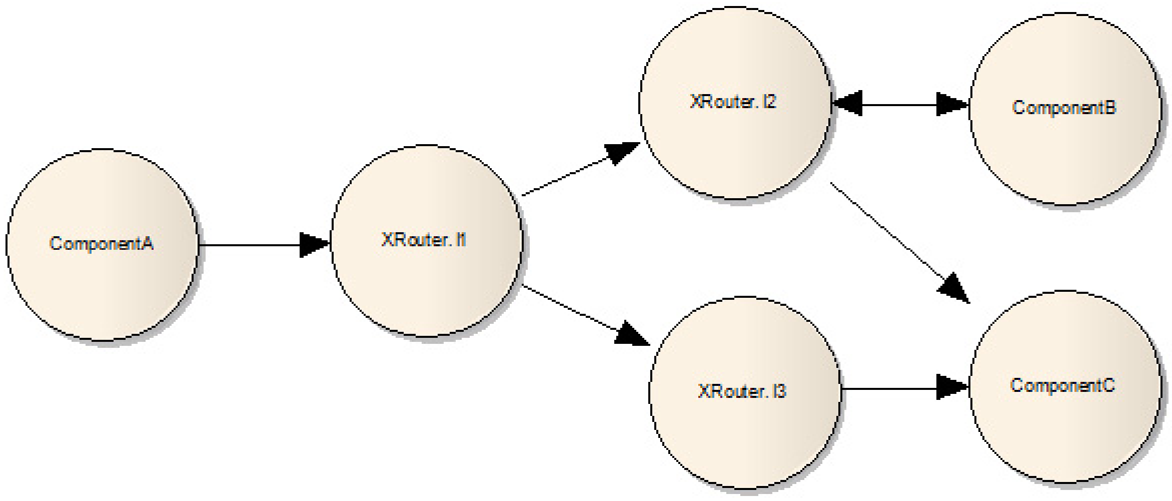
From the SOA point of view XRouter acts as an active intermediary between XML message producers and consumers and provides a way for designing communication between to or among multiple such parties. Its distribution logic enables routing and transformation of messages based on a configuration specified with some standard declarative languages (such as Schematron or XSLT).



XRouter as an intermediary service acting both as a service consumer and provider.

In enterprise environment XRouter can be utilized in various ways and can become a part of some more complex architectures. This is the reason why it is designed to be easily extensible and customizable to various additional requirements of its users.

XRouter can stand as an individual service realizing a simple hub-and-spoke architecture just as it is possible to connect multiple instances of XRouter, distributed at different machines or even locations in order to avoid downtimes.



The image demonstrates mediation using three instances of XRouter service, possibly located at three separate machines. The user might have chosen such an arrangement because the XRouter.X3 requires more frequent changes in configuration and it is undesirable also having to restart XRouter.I2 as its configuration is very stable. The communication between the instances of the XRouter service can be arbitrary and depend solely of the user's will. It might be possible to utilize exiting SOAP/HTTP Web-service adapters or create some new for MSMQ.

In more complex cases XRouter can potentially cooperate with further services, such as an orchestrator, a transaction coordinator, a load balancer, etc.) and be utilized to participate in the ESB concept.

Key features of XRouter

The important features provided by XRouter are described in the following list.

XML messages

XRouter is primarily designed for XML messages and the means for specifying the integration logic exploit this assumption mainly by using standard declarative languages

(XML schemas or XSLT) to enable flexible content-based routing and transformations.

However, in practice messages can contain data in non-XML formats (eg. plain text, CSV). XRouter can wrap such messages into its internal XML wrapper format, so that inside XRouter only XML messages are processed.

Various transport protocols and communication channels

Applications can exchange XML messages via transport protocols or communication means, such as HTTP, FTP, SMTP, MSMQ, filesystem, database, etc. XRouter can be easily extended by creating new adapters (input/output endpoints of XRouter), each implementing message exchange over a single communication protocol.

Along with the core XRouter service there are built-in adapters for reading and writing files on a filesystem, HTTP Web service clients/servers and e-mail sending via SMTP. Adapters act as plugins, so that they can be added and registered without the need for rebuilding and updating the whole system.

Communication styles

XRouter supports both synchronous and asynchronous communication style (where it makes sense). The communication style for a particular protocol is determined by the actual implementation. It is possible to poll resources which are not capable of actively sending messages, such as a filesystem or a database. The following message exchange patterns are possible:

- One-way
- Reliable one-way
- Request-Response
- Request-Optional response

Common message format

XRouter has its own internal XML format to wrap all procesed XML messages making a unified interface to messages during their processing. This XML format, named *token*, also carries inside some metadata needed for routing based on context (including at least adapter identification).

Mediation

Mediation is the fundamental purpose of using XRouter. It can help reducing coupling between integrated components. XRouter provides its internal engine, named MessageFlow, which offers a flexible and extensible means for processing messages. MessageFlow enables to compose actions, such as recognizing message type, routing, transformations, etc, to be performed with a token in order to realize low-level integration.

Orchestration

With XRouter it is even possible to create some simple orchestrations. However, the goal of XRouter is not in providing a full-fledged orchestration environment as those provided by BPEL servers.

Routing

Routing XML messages is a fundamental functionality which enables managing message distribution logic inside XRouter and thus decreases the coupling between the integrated components (ie. service requester does not need to know the service provider or its physical location).

It is possible to route XML messages based on their context (eg. the identifier of the receiving adapter) but also on the actual message content, ie. its structure, element contents, attribute values, etc. For specifying the routing rules it is possible to use Schematron over XPath 1.0, a declarative language not based on formal grammars.

Thanks to the expressive power of those two languages together is is effectively possible to route messages in various XML formats (eg. SOAP with the WS-Adressing extension).
Even some simpler business rules (eg. sums of multiple values) can be used for routing.

Validation

In addition to plain routing the content-based router can also be utilized for validating XML messages with schemas which can be described in the Schematron language. If needed the XRouter system can be easily extended with additional means of validation, eg. XSD schemas.

Data transformation

Some integrated systems communicating via XRouter may expect message formats not compatible with the rest of systems. The basic implementation of XRouter offers trasforming tokens with XSLT 1.0. Further transformations can be implemented in C# or another .NET language in form of plugins.

Service hosting

It is possible to host various services inside adapters, eg. for polling of resources such as filesystem or database, or WCF Web services. The following example case study demonstrates another possible usage of XRouter.

A case study

A company needs, in order to feed their algorithms, to gather measured meteorological data and weather forecasts from selected websites (eg. seznam.cz, idnes.cz). The various forecasts are usually a little bit different depending on their meteorological data providers. The company needs a tool which would grabs data from various sources in order to compute better predictions.

In this depicted situation an XRouter instance could be potentially utilized as a hosting platform (with all its comfort) containing specific adapters which would periodically gather data from the specified web sites and transform it into a standardized XML format. Also the original XHTML could be possibly transformed via XSLT. The gathered data could be then, according to the configured logic, stored into the database of the whether prediction system or sent to some computing Web services.

Monitoring and administration

XRouter provides some user-friendly supporting tools for configuring the XRouter service itself, configuring message flows, managing XML resources and monitoring and management of the service runtime. Configuration is facilitated by a GUI with an interactive message flow visualization.

XRouter Service logs its operation and the gathered log data can be later used for solving problem, debugging, etc.

Extensibility

The system can be extended by implementing new adapters and/or message flow actions according to some well-defined interfaces. The resulting plugins can be added to the installed XRouter system or upgraded without the need for rebuilding and upgrading the system as whole. Please refer to the Programmer's guide for details.

The order of message processing

XRouter assumes that order in which messages are processed is not important. In case the order is important in a business situation and ensure the ordering via an external tool (at the level of the integrated systems). In the SOAP context the Sequence construct of WS-ReliableMessaging is quite suitable.

Security

When realizing B2B solutions securing data transfers is often desirable (eg. wrapping SOAP messages in HTTPS). In XRouter it is possible to create and configure specific adapters which transfer data in encrypted form. The basic implementation of XRouter assumes the environment is already secured.

Performance

XRouter tries to do the best to make processing of messages as efficient as possible. For example content-based routing is built upon a very efficient native C# Schematron validator with partial validation.

Availability

The availability of XRouter is enhanced by a monitoring mechanism which enables in case of a failure to automatically start the XRouter Service and immediately notify the administrator by e-mail.

Scalability

XRouter processes messages in parallel using the modern constructs of MS .NET Framework 4.0, in particular the Task Parallel Library.

Reliability

The reliability of message processing depends on a particular communication style or channel, or transfer protocol being used. Eg. RPC-style Web services exchange messages in the request-response pattern, ie. reliability is in the competence of the integrated components. For some adapters (eg. filesystem) it might use useful to utilize XRouter's own database persistence in order to avoid message losses or duplication.

XRouter does not support transaction processing since from the SOA point of view it can be assumed that XRouter can cooperate with an external service specialized right in providing active coordination of atomic transactions.

Logical view of the XRouter architecture

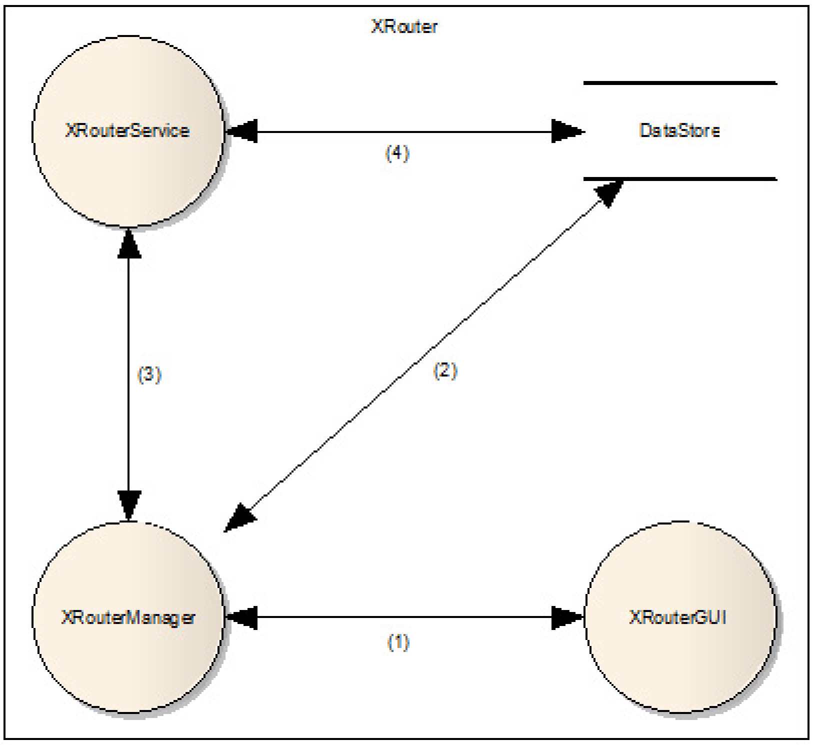
At the highest level of abstraction an instance of the XRouter system consists of an XRouter Service, XRouter Manager, XRouter GUI and a Data Store. The mentioned parts interact with some of the others.

XRouter Service is a Windows service acting as the heart of XRouter. It acts as an active intermediary between the external systems to be integrated and all the XML messages flow through it.

XRouter Manager is a Windows service providing an additional support for monitoring and administration of the XRouter Service. A single XRouter Manager instance can be associated with another single XRouter Service instance installed on the same machine.

XRouter GUI is a thick client for Windows desktop providing a user-friendly interface for controlling the XRouter Service. Multiple instances of XRouter services can be administrated with a single XRouter GUI (even remotely).

Data Store is a storage for persistent resources. It consists of a databse (MS SQL Server) and a filesystem directory of log files produced by the XRouter Service.



The image portrays the parts of the XRouter system and data transfers among them.

1. The XRouter GUI communicates with the XRouter Manager service via a Web service (SOAP/HTTP) in the RPC style.
2. The XRouter Manager accesses the Data Store – it connects to the MS SQL Server database (for reading and writing) and accesses the log directory (read-only) of the XRouter Service.
3. The XRouter Manager can control the running of the XRouter Service via the Windows service interface. It can start it, stop it and query its run-time status.
4. The XRouter Service accesses the Data Store – it connects to the MS SQL Server

database (for reading and writing) and accesses its log directory (write-only).

XRouter Service

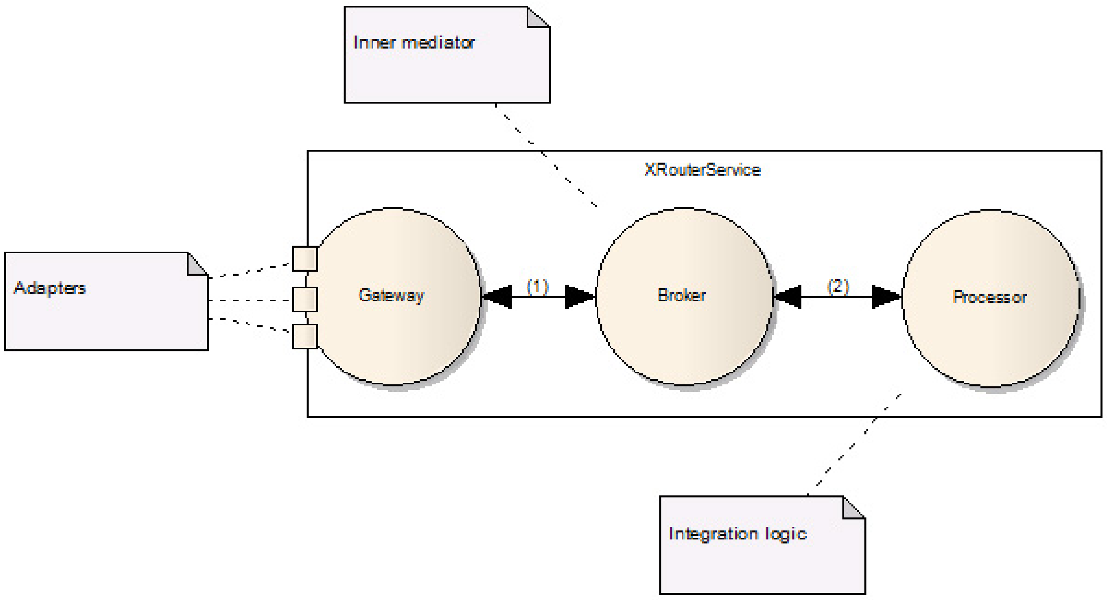
XRouter Service constitutes the core executive part of the XRouter system. It is not monolithic, rather it is logically and physically divided into three interacting parts – Gateway, Broker and Processor. It can also be extended via plugins.

Gateway hosts a collection of adapters for communication with external systems.

Broker acts as a mediator between Gateway and Processor and provides the access to the Date Store for them.

Processor interprests the MessageFlow instructions and perform the actual processing of messages.

It works according to a user-defined configuration which is read once during the start-up of the service instance from the Data Store (database). The application configuration mainly consists of the interaction logic definition (MessageFlow), a list of registered plugins (gateway adapters, message flow action nodes, etc.) along with their specific configuration, and declaration of adapter instances at the gateway and a storage of XML resources. The interaction logic contains instructions how to process which types of messages and uses XML resources such as XML schemas or XSL transforms.



The image displays the main parts of the XRouter Service and data transfers between them. Gateway and Broker exchange messages (1) received from external systems and to be sent to them. Broker and Processor exchange messages while being processed (2) or messages yet to be processed.

Gateway

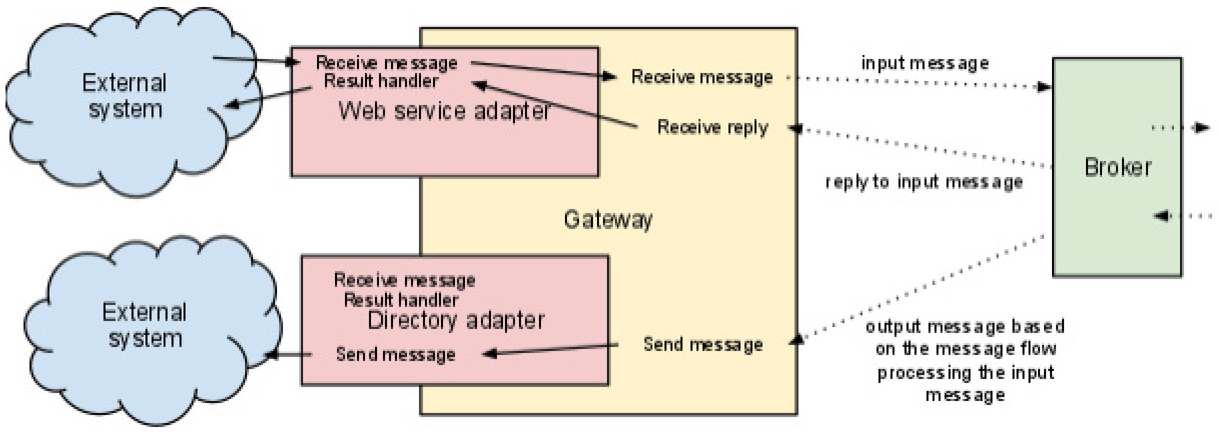
Gateway is a component which manages communication of the XRouter Service with external systems via various communication means.

As there can be many transport protocols or communication channels each one can be managed by a specialized implementation called *adapter*. So an adapter is a component of the gateway specialized in one type of communication with external systems (such as file exchange, Web service, e-mail, etc.). It receives incoming messages and passes them to the gateway for further processing. Also it sends outgoing processed messages (arisen in the MessageFlow) away. The gateway can contain one or more adapter instances of multiple types.

A single adapter can consist of one or more *endpoints* which can be used for identifying various input and output points (corresponding to communication channels). Endpoints increase the granularity or adapters (eg. one adapter may receive messages from two channels with completely different message types) and reduce the number of adapter instances (eg. multiple output directories can be managed by a single adapter instance with multiple endpoints).

The communication style (synchronous or asynchronous) of an adapter depends upon the actual implementation of the adapter. The following variants are possible:

- input synchronous endpoint waits for a reply returned by the Broker after processing the message
- input asynchronous endpoint does not wait for any reply from the Broker
- output synchronous endpoint waits for a reply from the external system
- output asynchronous endpoint does not wait for any reply from the external system



The image demonstrates the usage of an input synchronous adapter for a Web service with a single endpoint and an output asynchronous adapter with a single endpoint for a filesystem directory.

The main job of adapters is to wrap or extract an XML message into resp. from a format suitable for a particular transport protocol (eg. HTTP). Messages travel within XRouter inside a container structure called *token* which is created at the gateway. Each token contains the original input message which initiated the processing. In general token contains a collection of all associated messages created during processing of the original message – eg. a message produced by transforming a message with a XSLT.

From the user's point of view a token is a XML document available for queries in MessageFlow. A token also carries some metadata added by the gateway or some adapters (eg. original input file name). Those metadata can be later used in MessageFlow eg. for routing or transferring information between adapters.

Both the gateway and each adapter instance run in their own threads. Although the adapter threads are managed by the gateway thread, adapters can make their own inner threads at will. The gateway creates and configures adapter instances based on its configuration. Each adapter type is associated with an .NET assembly where it is loaded from.

Broker

Broker is a component which the mediates exchange of message resp. tokens between the gateway and the processor. It provides the other components with application configuration which is loaded from the Data Store at the XRouter Service start-up and then is cached. Moreover, the broker provides the other components with the access to the persistent storage in order to safely store messages during their processing.

Processor

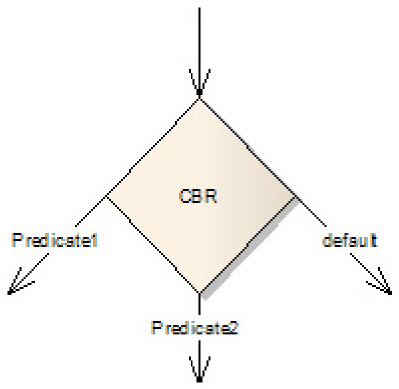
Processor is a multi-threaded interpretter of the MessageFlow which processes messages resp. tokens. It contains a collections of tokens being currently processed and a current MessageFlow. The number of worker threads can be configured.

A particular MessageFlow specifies the integration logic and thus enables defining paths of messages resp. tokens, their routing and further actions to be made upon messages (eg. transformation, sending a specified message from the token to the gateway, or possibly even some custom deciphering).

A MessageFlow is represented as a general oriented graph (as in the Graph theory) and may contain node s of the following types.

Content-based router (CBR) node

From the XRouter's conceptual point of view CBR is a fundamental and the most important component of the MessageFlow. It enables to declaratively configure routing of XML messaged based on their actual content.



The meaning of a CBR node is very similar to a decision node in the Activity diagram in UML. The decision node has a single input edge and two or more output edges (one of them being the default one).

A token which arrives at the input edge is offered to all the output edges but selects exactly one output edge. A decision node thus acts as a crossroads of multiple one-way streets at which a token must continue its travel in exactly one direction. The input of such a node is the whole token and the output is a single output edge.

Each of the non-default output edges of a CBR node is guarded by an entry condition (a predicate), so that is accepts the token if and only if the condition is fulfilled. The default edge then accepts all the remaining tokens. It is important that all the conditions are mutually exclusive. The user who create the configuration is responsible for maintaining this.

A predicate enables to specify conditions (expressions) upon a token. An example of such an expression might be: *„the root element is X and the X element contains the Y attribute and the Y attribute contains the Z value“* and so on. ISO Schematron is used for expressing such conditions. It is a declarative language with an expressive power and a style of usage very suitable for content-based routing. Moreover it can be very efficient in terms of performance.

Example:
Imagine we would like to route SOAP messages with a SOAP header using WS-Addressing (the *To* element) and also according to the actual message content. The following SOAP message contains in the *To* element an identifier of the target service (Payment) which registers a payment only if the sum of the values of the *price* attributes is greater than zero.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa="http://
www.w3.org/2005/08/addressing">
<S:Header>
 <wsa:To>Payment</wsa:To>
</S:Header>
<S:Body>
 <Order>
  <Item price="100"/>
  <Item price="150"/>
  <Item price="134"/>
 </Order>
</S:Body>
</S:Envelope>
```

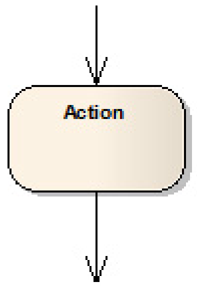
For such an XML message we can create a CBR predicate with the following Schematron schema:

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
<ns prefix="soap" uri="http://www.w3.org/2003/05/soap-envelope"/>
<ns prefix="wsa" uri="http://www.w3.org/2005/08/addressing"/>
<pattern>
 <rule context="/">
  <assert test="soap:Envelope"/>
 </rule>
</pattern>
<pattern>
 <rule context="/soap:Envelope">
  <assert test="soap:Header"/>
  <assert test="soap:Body"/>
 </rule>
</pattern>
<pattern>
 <rule context="/soap:Envelope/soap:Header">
  <assert test="wsa:To"/>
 </rule>
</pattern>
<pattern>
 <rule context="/soap:Envelope/soap:Header/wsa:To">
  <assert test="text()='Payment'"/>
 </rule>
</pattern>
<pattern>
 <rule context="/soap:Envelope/soap:Body">
  <assert test="sum(Order//Item/@price)>0"/>
 </rule>
```

```
</pattern>
</schema>
```

A MessageFlow can contain multiple different CBR nodes, which helps to better organize the MessageFlow and thus logically organize the message paths.

Action node
The action node can perform an action with the input token (or its subset), eg. sending to a gateway, transforming, decrypting, etc. The family of suppoted actions can be extended by creating new actions as .NET plugins. Each action has its own implementation-specific configuration.

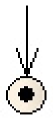


This node can be though of as a function which takes a token and modifies it, eg. adds a brand new message, adds a message received from a Web service, adds the output from a XSL transformation.

The action implementations might throw implementation-specific MessageFlow exceptions (eg. that it a transformation failed) which get written into the token being processed. This way it is even possible to route according to exceptions that arise during processing!

A single action node might host one or more actual action. In case there is more than one action they get performed in parallel. Thus a prepared message can be sent to multiple locations in parallel in order to reduce waiting.

Terminator
Terminator represents a special action which finishes the path of processing a single input message.



The action can be of two kinds:
- synchronous – the original input endpoint requires a reply (eg. an RPC-style Web service)
- asynchronous – the original input endpoint does not need any reply (eg. a Document-style Web service)

<u>Example:</u> Assume that an input adapter realizes an RPC-style Web service. An external component (web client) raises this Web service with a SOAP/HTTP request. The following process can be though of as executing a web method. A SOAP message wrapped inside a token travels through the MessageFlow, ie. the message is recognized, some specified actions are accomplished (eg. raising a series of Web services which add some more messages to the token). At the end, the token enters a synchronous terminator node which finishes the web method call and returns some reply back to the calling external component as it called the XRouter Service in the request-response style.

XRouter Manager

XRouter Manager is a Windows service which monitors an associated XRouter Service instance and supports its administration. XRouter GUI can remotely connect to XRouter Manager in order to control the XRouter Service, view or update its configuration, or check the logs the service produced.

XRouter Manager consists of several parts: Console Server, Watcher and Reporter.

Console Server

Console Server is an RPC-style Web service which provides access to the XRouter Service to clients, such as XRouter GUI. It provides the following operations:

- obtaining the current Windows service status of the XRouter Service (eg. running or stopped)
- starting or stopping the XRouter Service, eg. to make a graceful downtime in order to change configuration or upgrade the program
- getting and changing the current configuration of XRouter Service
- getting log entries, archived tokens and other persistent resources (possibly filtered and paged)

Watcher

Watcher is a simple server which periodically monitors the status of the associated XRouter Service instance provided by the Windows Service Control Manager. It is able to detect when the service might have be down due to a failure, try to automatically start it again and inform the administrator via e-mail. This behavior can be configured (enabled or disabled).

Reporter

Reporter is a simple server which scans log files of the associated XRouter Service and summarizes them into e-mail reports. Eg. once a day at a specified time a report with the total counts of errors and warning that happened during yesterday can be prepared and sent to the administrator.

XRouter GUI

XRouter GUI is a thick client for Windows desktop which acts as an administration console for the XRouter service pair. It enables viewing and editing application configuration, including MessageFlow, adapters and XML Resources. It also provides facilities to view and possibly filter log files and tokens.

Message flow

Thi module enables to visualize the MessageFlow graph, which is describing the interaction logic of message processing, and edit it in a user-friendnly way.

XML resources

While configuring the XRouter processes, MessageFlow in particular, the users need to refer to a quantity of XML documents (basically Schematron schemas for routing and XSL transforms for modifying messages). It is a good idea to facilitate organizing and management of such XML data in order to make the work easier.

Example: A user wants to recognize SOAP messages like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header/>
<soap:Body>
```

```
 <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
  <m:Item id="1">Desc</m:Item>
  <m:Item id="2">Desc</m:Item>
  <m:Item id="3">Desc</m:Item>
  <m:Item id="4">Desc</m:Item>
 </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

The user might have defined the following Schematron schema in order to use it as a predicate in the CBR.

```
<?xml version="1.0" encoding="utf-8" ?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
<ns prefix="soap" uri="http://www.w3.org/2001/12/soap-envelope" />
<ns prefix="app" uri="http://www.w3schools.com/prices" />

<pattern id="soapEnvelope">
 <rule id="r1" context="/">
  <assert test="soap:Envelope"/>
 </rule>
 <rule id="r2" context="/soap:Envelope/@soap:encodingStyle">
  <assert test=".='http://www.w3.org/2001/12/soap-encoding'"/>
 </rule>
 <rule id="r3" context="/soap:Envelope/soap:Header">
  <assert test="following-sibling::*[1][self::soap:Body]"/>
 </rule>
 <rule id="r4" context="/soap:Envelope/soap:Body">
  <assert test="true()"/>
 </rule>
 <rule id="r5" context="/soap:Envelope/*">
  <assert test="false()">Element <name/> is not allowed.</assert>
 </rule>
</pattern>

<pattern id="soapBody">
 <rule id="r1" context="/soap:Envelope/soap:Body">
  <assert test="count(app:GetPrice)=1"/>
 </rule>
 <rule id="r2" context="/soap:Envelope/soap:Body/app:GetPrice">
  <assert id="a1" test="count(app:Item)>0"/>
  <assert id="a2" test="10>count(app:Item)"/>
 </rule>
 <rule id="r3" context="/soap:Envelope/soap:Body/app:GetPrice/app:Item">
  <assert test="count(@id)=1"/>
 </rule>
 <rule id="r4" context="/soap:Envelope/soap:Body/app:GetPrice/app:Item/@id">
  <assert id="a1" test="number(.)"/>
  <assert id="a2" test="not(.=../preceding-sibling::app:Item/@id)"/>
 </rule>
</pattern>
</schema>
```

The schema is composed of the patterns. The first can recognize the SOAP envelope and the second a particular content of the `<Body>` element, ie. the application message. It is not desirable to force the user to specify recognizing the SOAP envelope in each schema (resp. CBR predicate).

A solution is to use the standard Schematron `<include>` element and to store the first pattern into a separate document. Then is is possible to refer to this pattern from every schema which need to recognize the SOAP envelope.

```
<?xml version="1.0" encoding="utf-8" ?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
<ns prefix="soap" uri="http://www.w3.org/2001/12/soap-envelope" />
<ns prefix="app" uri="http://www.w3schools.com/prices" />

<include href="pattern_soapEnvelope"/>

<pattern id="soapBody">
 <rule id="r1" context="/soap:Envelope/soap:Body">
  <assert test="count(app:GetPrice)=1"/>
 </rule>
 <rule id="r2" context="/soap:Envelope/soap:Body/app:GetPrice">
  <assert id="a1" test="count(app:Item)>0"/>
  <assert id="a2" test="10>count(app:Item)"/>
 </rule>
 <rule id="r3" context="/soap:Envelope/soap:Body/app:GetPrice/app:Item">
  <assert test="count(@id)=1"/>
 </rule>
 <rule id="r4" context="/soap:Envelope/soap:Body/app:GetPrice/app:Item/@id">
  <assert id="a1" test="number(.)"/>
  <assert id="a2" test="not(.=../preceding-sibling::app:Item/@id)"/>
 </rule>
</pattern>
</schema>
```

XML Resources enables the user to create and organize the needed XML data and identify them arbitrarily, eg. in terms of the problem domain. For creating XML resources the user is provided with a text editor. For organizing XML resources the user is provided with a software which supports keeping the archive organized. Each document is uniquely identified and those identifiers are offered the user while creating a MessageFlow. In addition to organizing an archive of XML resources is is possible to validate the documents (either as being well-formed XML, or with a Schematron schema).

XRouter Glossary

There follows an alphabetical glossary of the main terms of the XRouter system, which are used throughout the whole documentation.

A

Action
Action is one of the node types in MessageFlow within XRouter Service which performs an operation with a token. The various action implementations are usually loaded from plugins, some important actions are built-in.

Adapter
Adapter is a part of the Gateway of the XRouter Service which can communicate with external systems. The various adapter implementations are loaded from plugins.

B

Broker
Broker is a component of the XRouter Service which acts as a mediator between the Gateway and Processor components.

C

CBR
Content Based Router is one of the node types in MessageFlow within XRouter Service which implement routing of messages according to their content and some specified rules.

D

Data Store
Database (MS SQL Server) and a directory for log records produced by the XRouter Service.

E

Endpoint
Endpoint is and identification of a single communication channel for exchanging messages between an adapter and an external system.

External system
A system, application or component which uses XRouter as a mediator for communication with other such external systems.

F

G

Gateway
Gateway is a component of XRouter which hosts adapters.

H I J K L

M

MessageFlow
A user-defined specification of the integration logic which controls the behavior of the XRouter Service for some message types.

N O

P

Processor

A component of XRouter service containing a multi-threaded MessageFlow interpretter.

Q R S

T

Terminator
A special MessageFlow node which finishes a message processing path.

Token
A container structure which hold the original input XML message, possibly various other messages created during the processing, and some metadata.

U V W

X

XRouter GUI
A user-friendly application for management and monitoring of the XRouter Service.

XRouter Manager
A support service for management and monitoring of the XRouter Service.

XRouter Service
The mediator service itself which transfers and processes the messages.

Y Z

Appendix: XRouter – Project information

XRouter can integrate various systems communicating via XML messages. It is a simple, easy-to-use, configurable, light-weight, efficient alternative to complex enterprise service bus (ESB) solutions. It is well-designed, written in C# for .NET, thoroughly documented and released as an open-source software.

The XRouter project consists of several subprojects:

- **XRouter** – light-weight extensible low-level XML router for .NET
  - plus a configuration management GUI and a simple monitoring service
- **SchemaTron** – Native C# validator of ISO Schematron language
- **DaemonNT** – Windows service hosting made easy
- **ObjectConfigurator** – reflection-based configuration utility
- **SimpleDiagrammer** – interactive visualizer of oriented graphs

Licence

XRouter is released under the terms of the MIT License. See the LICENSE file.

Project members

The XRouter software project was created by the following people (in alphabetical order):

- Soběslav Benda
- Miroslav Cicko
- Tomáš Kroupa
- Petr Sobotka
- Bohumír Zámečník

Other projects

There are some other projects for integration of applications or to support ESB concepts. Our goal is to have our own integration platform for Windows environment, based on our own ideas, experiments and research with the following features and requirements:

- It is developed in the MS .NET Framework and thus can take advantage of its plentiful possibilites.
- It is a physical entity, not just a concept, so that is it close the the Broker architectonic pattern. In contrast to eg. NServiceBus [1] our aim is to minimize the need for programmer's interventions to existing systems and to have a central management.
- Its primary purpose is to perform low-level integration, ie. it acts as a mediator for common integration needs, so it is not needed to introduce a complex environment of a BizTalk [2] type. The integration should be solved in a clean unified manner in order to make it economicly more efficient.
- It is suitable for arbitrary XML formats. Today, XML is one of the most popular languages for data interchange and there exist a lot of tools for working with it. Thus a flexible declarative configuration is possible, in contrast eg. to ESB.NET [3].
- It is open source.

Conclusion

Some statistics:

- cca 27k lines of clean C# code, cca 18k lines of XML
  - C# code: cca 80% of production code, 20% of test code
- cca 7k lines of comments
- over 500 commits
- in total 10 solutions, 40 C# projects, over 400 C# files
  - core (excepts tests, examples and demos): 3 sln, 15 csproj
- output: 7 EXE, 11 DLL

This project gave the authors a very valueble practise and experience in the whole software development process and in team management and communication. The resulting product can be useful in real world. It can be considered a success.

Future work

Supporting the integration of systems is a very non-trivial thing to do and even today it is under an extensive research. Advanced integration products are usualy very expensive. XRouter gives a basis for future development and maintaining your own integration platform which can be customized and potentially even made distributed and which quite easily enables to support the realization of service-oriented or different architectures in the .NET environment.

References

[1] http://www.nservicebus.com
[2] http://www.microsoft.com/biztalk/en/us/default.aspx
[3] http://keystrokeesbnet.codeplex.com/