System Integration loan broker Project

The project involves conceptual and practical work with messaging and web services – the two main topics of the Systems Integration module. You must design and implement a Loan Broker solution as described in Enterprise Integration Patterns[1], chapter 9 (pp. 361-370). Your solution must demonstrate how to compose routing and transformation patterns into a larger solution.

The use case is fairly simple as it describes the process of a consumer obtaining a quote for a loan based on loan requests to multiple banks. The banking operations must be simulated in your solution as the project has its focus on integration issues as opposed to being an exercise in consumer financial services.

Groups

You must work on the assignment in the same groups as the Large System Development course.

Counselling

Status meetings are held with Tine Marbjerg on Wednesday 31/10, 7/11 and 14/11 where your project work will be reviewed and you have the opportunity to ask questions about the project. There will be general help in the classroom from 12.45-14.00.

A review schedule where you register your own timeslot for each review date be found here.

Hand-in

You hand-in both source code and text descriptions (see details in section Further Requirements).

You must specify a Github link to your work to Moodle on **November 20th 2018 23.55** at the latest.

Exam

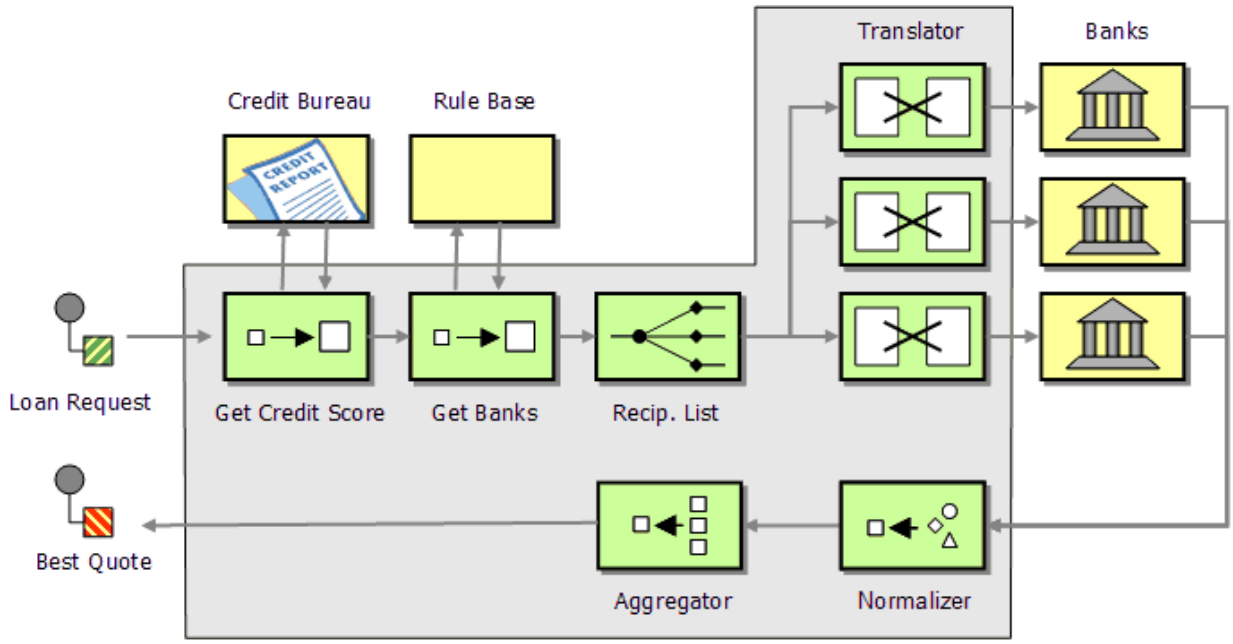
The project is mandatory (80 study points) and your solution will be part of the examination.

[1]Enterprise Integration Patterns. Designing, Building, and Deploying Messaging Solutions by Gregor Hohpe & Bobby Woolf. Addison-Wesley, 2004. ISBN-13 978-0-321-20068-6.

LOAN BROKER COMPONENT

Loan Broker Design

This is the Loan Broker design that you must implement:



The Loan broker component itself is the grey box, and must be implemented as a Pipes and Filter architecture on a RabbitMQ server. You must implement all elements inside the Loan broker as separate processes.

Several external components have already been implemented for you:

- Credit Bureau (SOAP web service).
- Two banks (RabbitMQ implementations using XML and JSON respectively).

Interfaces and specifications for these external components are specified here.

In addition to implementing the Loan Broker component, you must implement minimum 3 external components yourselves:

- the Rule Base as a SOAP web service
- at least two more banks (One bank as a web service and one bank using messaging and RabbitMQ)

The Loan Broker component itself must be implemented as a service (with messaging components inside) that can be accessed through a simple web site or a test client that you make.

The Loan broker component must contact the banks using a distribution strategy. This means using a rule based recipient list where the broker decides upon which banks to contact based on the credit score for each customer request. It would for instance be a waste of time sending a quote request for a customer with a poor credit rating to a bank specializing in premier customers. You must make up the banking rules yourself. Keep them simple. You need to find a way to include knowledge about the banks into the Rule Base web service. The credit score scale ranges from 0 to 800 (800 being the highest and best score).

Loan Broker Process

The loan quote process flow goes like this:

1. Receive the consumer's loan quote request (ssn, loan amount, loan duration)
2. Obtain credit score from credit agency (ssn → credit score)
3. Determine the most appropriate banks to contact from the Rule Base web service
4. Send a request to each selected bank (ssn, credit score, loan amount, loan duration)
5. Collect response from each selected bank
6. Determine the best response
7. Pass the result back to the consumer

Each bank has its own format so you must make sure to translate the loan quote request into the proper format for each bank using a *Message Translator* (as shown in the Loan Broker Design above). Also, a *Normalizer* must be used to translate the individual bank responses into a common format. An *Aggregator* will collect all responses from the banks for a specific customer request and determine the best quote.

All the parts inside the Loan Broker component are connected through messaging. So you end up with a number of independent programs that need to be started up as individual processes.

The Loan Broker component itself (the big grey box) is also a single process.

EXTERNAL COMPONENTS

Loan Broker Web Service

You can request a credit score from the credit agency given the customers social security number (SSN). SSN must follow the structure of Danish SSN's, i.e. XXXXXX-XXXX, with 11 characters (10 numbers and a dash). The service is following the rules about new SSN's (2007) and will not carry out the 11 modulo check, but only validate the structure. If the structure is valid a credit score between 800-0 (hi-lo) otherwise it will return -1. The credit agency is receiving credit information from many sources, thus two validations on the same SSN at different times may differ.

You find the WSDL from the service here:

http://datdb.cphbusiness.dk:8080/CreditScoreService/CreditScoreService?wsdl

RabbitMQ Banks

The two banks are listening on each their RabbitMQ fanout exchange on this server[2]:
`datdb.cphbusiness.dk:5672`

The bank at exchange **`cphbusiness.bankXML`** is XML based. Loan requests have this format:

```
<LoanRequest>
      <ssn>12345678</ssn>
      <creditScore>685</creditScore>
      <loanAmount>1000.0</loanAmount>
      <loanDuration>1973-01-01 01:00:00.0 CET</loanDuration>
</LoanRequest>
```

The loan duration will be calculated from 1/1 1970. The above example therefore has a loan duration of 3 years.

[2] Consult rabbitfun.zip for code examples of how to access RabbitMQ on `datdb.cphbusiness.dk`
The administration web interface can be accessed here: `http://datdb.cphbusiness.dk:15672/`

The corresponding response looks like this:

```
<LoanResponse>
   <interestRate>4.5600000000000005</interestRate>
   <ssn>12345678</ssn>
</LoanResponse>
```

The bank at exchange **cphbusiness.bankJSON** is JSON based.

Here is an example of its loan request format (The loan duration is specifies as number of months):

```
{"ssn":1605789787,"creditScore":598,"loanAmount":10.0,"loanDuration":360}
```

The corresponding response looks like this:

```
{"interestRate":5.5,"ssn":1605789787}
```

OBS! Both banks place their reply on the queue which is specified as reply-to in the header.

FURTHER REQUIREMENTS

All groups must make a minimum viable Loan Broker solution in order to have their assignment approved. In addition to the requirements already specified, there are a number of things that are recommendable to do. In order to improve your chances of a higher grade at the exam, you should include one or more elements in your project (both quality and quantity count):

- Documentation of your Loan Broker solution
- Description of the Loan Broker web service you create, i.e. not the WSDL file, but a textual contract with restrictions to the input parameters, exceptions, etc.
- Identification of potential bottlenecks in your solution and discussion of possible ways to improve performance
- Testability of your solution is (see more on testability pp 440-443) documented by concrete tests (2nd semester students are expected to do more on this topic)
- RabbitMQ is mandatory, but you could use different implementation language(s) for different components/services
- Your design, e.g. separation of business and messaging logic inside the components
- Error handling
- The way all internal Loan Broker processes are started (hopefully in a smart way)