System Integration Loan Broker Project

The project involves conceptual and practical work with messaging and SOAP web services. You must design and implement a Loan Broker solution. The business case for the Loan Broker is described in Enterprise Integration Patterns[1], chapter 9 (pp. 361-370). Your solution must be a messaging implementation using RabbitMQ as a message broker. Your solution must demonstrate how to compose routing and transformation patterns.

The use case represents 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 project focus is on integration solutions and not an exercise in consumer financial services.

Groups

You must work on the assignment in groups of 3-4 students as registered on Moodle.

Counselling

There will be status meetings with Tine Marbjerg on Wednesday 27/9, 4/10 and 11/10 where we review your project work. A review schedule will be posted on github[2].

Hand-in

You must hand-in a report consisting of text descriptions and source code (see more in section Further Requirements later on).

Your report must be uploaded to Moodle in Loan Broker Project hand-in folder no later than October 23rd 2017 as one zip file for the whole group. It is ok with link to github repository.

Zip file name must have this format: `<Group name>.zip` (e.g.: `LoanBroker1.zip`). Make sure the source code is nicely formatted and has relevant comments.

Exam

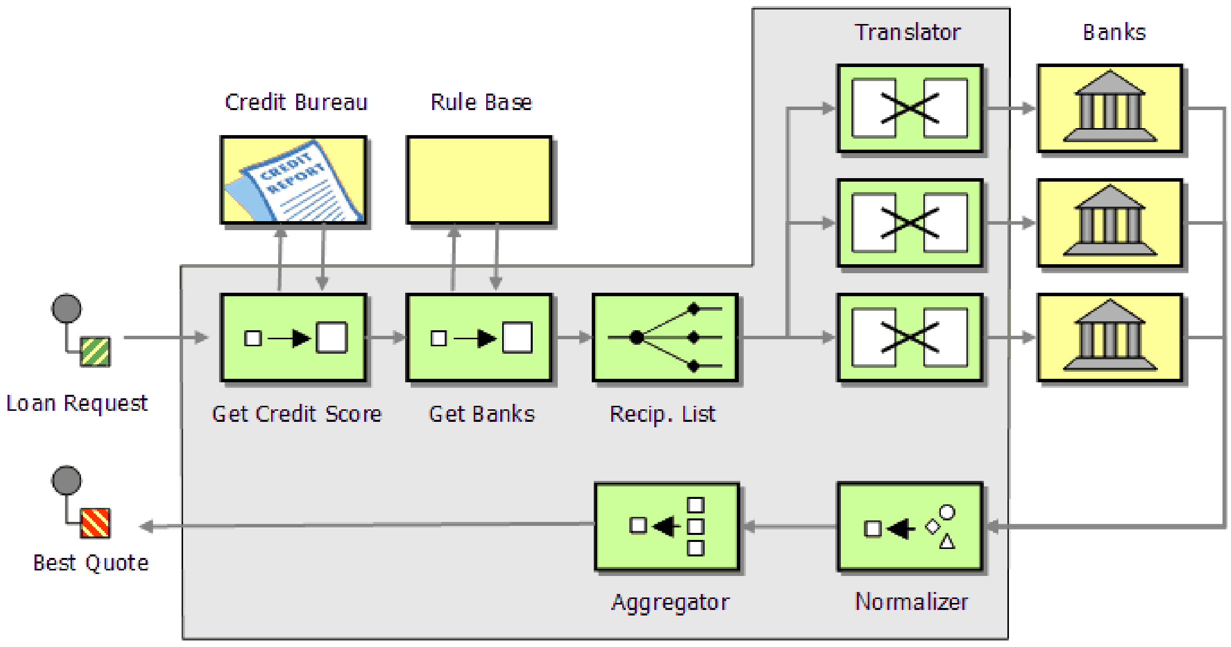
The project is 70 study points assignment and as such a requirement for the examination. Your solution will be used at the exam as the basis for discussion.

[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.

[2] https://github.com/datsoftlyngby/soft2017fall/blob/master/docs/SI_plan.md

Loan Broker Component

This is the Loan Broker design that you must implement:



The Loan broker component itself is illustrated as the grey box. You must implement all the elements of the component in separate processes (pipes and filter architecture). Several external components have already been implemented for you:

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

Interfaces and specifications for these components will be explained in sections later on.

In addition to implementing the Loan Broker component, you must implement the Rule Base as a SOAP web service and at least two more banks: One bank as a SOAP web service and one bank using messaging (via RabbitMQ).

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

The Loan Broker 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 Rules Base SOAP web service. The credit score scale ranges from 0 to 800 (800 being the highest and best score). For inspiration upon credit scoring, you may want to look here: http://www.investopedia.com/articles/personal-finance/081514/what-do-credit-score-ranges-mean.asp

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*. 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 components are asynchronously connected through messaging and each of the external components (credit, rule base, banks) are independent components. Thus, you end up with a large number of small independent programs, using SOAP/WSDL or messaging to communicate, that run as single processes, i.e. all the boxes in the above figure must be single processes, including the Loan broker itself (the big grey box).

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://138.68.85.24:8080/CreditScoreService/CreditScoreService?wsdl

RabbitMQ Banks

The two banks are listening on each their fanout exchange.

The bank at exchange `cphbusiness.bankXML` is XML based. It's 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. Therefore loan duration of 3 years must look as the above example.

The corresponding response will look like the following:

```
<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:

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

The loan duration corresponds to the requested number of months for the loan.

The corresponding response will look like this:

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

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

Further Requirements

In addition to the already mentioned requirements above, the following must be done:

- Make screen dumps of a process flow scenario (i.e. your running code).
- Make a design class diagram and a sequence diagram that document your solution (including comments). Other types of visual documentation are also welcome ☺
- Describe potential bottlenecks in your solution and possible enhancements to improve performance.
- Describe how testable your solution is (see more on testability pp 440-443).
- Provide a description of the loan broker web service you create.
- You must use RabbitMQ as your message broker, but the implementation language(s) is/are free of choice (Java, C# or similar).