System Integration 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 solutions as opposed to being an exercise in consumer financial services.

GROUPS

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

COUNSELLING

There will be status meetings with the teachers (Helge-Rolf Pfeiffer and Tine Marbjerg) on Wednesday 29/11, 6/12 and 13/12 where we review your project work. A schedule will be posted on Fronter, when the project groups are established.

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 Fronter in Project hand-in folder no later than December 20th 2016 as one zip file. Use Fronter's group hand-in function (i.e. only ONE hand-in pr group).

File/Project name must have this format: `<Group name>_<Project name>.zip` (example: `03_loan_broker.zip`). Make sure the source code is nicely formatted and all classes copied to one (!) generally readable electronic file (txt-, pdf- or word document).

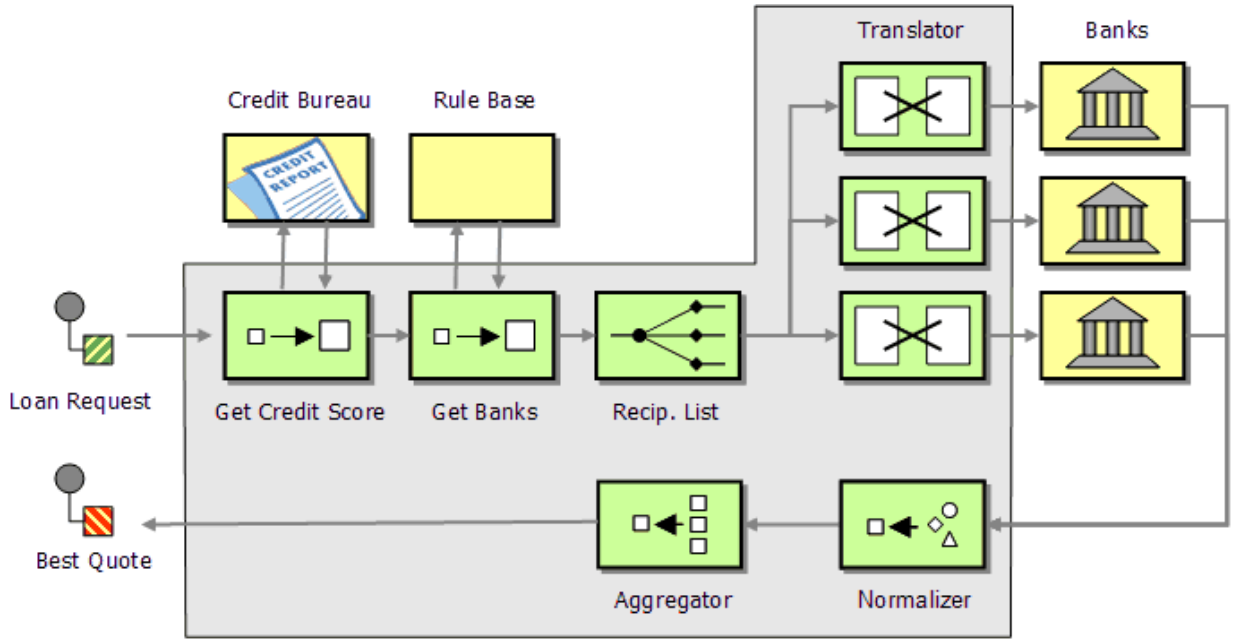
EXAM

Your solution will be part of the examination. This means that it can be used at the exam as the basis for discussion. Also, exam questions will refer to the assignment.

[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

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. 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, you must make the Rule Base as a web service and at least two more banks yourself: One bank as a web service and one bank using messaging and RabbitMQ,

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

The 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 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.freescore.com/good-bad-credit-score-range.aspx

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 sevice
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 overall Loan Broker web service are asynchronous 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 must run as single processes, i.e. all the boxes in the above figure must be a single process, including the loan broker itself (the big gray 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://datdb.cphbusiness.dk:8080/CreditBureau/CreditScoreService?wsdl

RabbitMQ Banks

The two banks are listening on each their fanout exchange (look at the slides for the RabbitMQ lectures, or look at the code examples on fronter).

The bank at exchange `cphbusiness.bankXML` is XML based. Its' 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 (that is of your running code)
- Make a design class diagram and a sequence diagram that document your solution (including comments)
- 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, i.e. not the WSDL file, but a textual contract with restrictions to the input parameters, exceptions, etc.
- Discuss you implementation from the SOA perspective, e.g. how do you solution follow the layers in the SOA reference model, coupling, cohesion, granularity.
- You must use RabbitMQ as your message broker, but the implementation language(s) is/are free of choice (Java, C# or similar).