SBVR Transaction Editor – Quick Start

The aim of the SBVR Transaction Editor is to obtain the ***TransactionContext.xml*** file
Flypeer needs this file to instanciate and to run transaction in a p2p network based on Flypeer ((http://kenai.com/projects/flypeer)
To obtain this file we will leverage upon a semantic description of composition and upon a semantic search of web services involved in transaction by means of SBVR (Semantics of Business Vocabulary and Business Rules) (http://www.omg.org/spec/SBVR/1.0/).

1. Operation

A three tabs windows will guide the user to the TransactionContext.xml file.

a) The user will type his composition rules in accordance with a structured english (SBVR) in the Production Rules pane in the first tab (SBVR Rule Editor) you can see in figure 1.

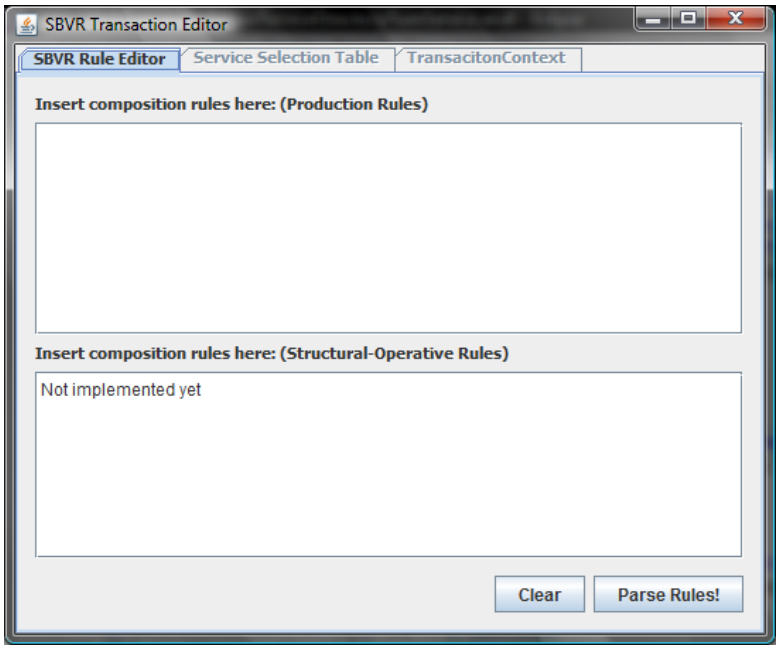

Figure 1

The second pane (Structural & Operative Rules) will be used to let the user typing its structural rules (it is obligatory that..., it is necessary that, ...etcetc) to refine and improve the semantic search results, so that they better fit to user requirements. This is not implemented yet!

b) Each fact type must be paired with a web service. A semantic search will narrow the results set. You can see the results of the semantic search (each typed fact type is searched in the vocabulary linked to a web service, via sawsdl) displayed in the combobox in the table in the Service Selection Table tab, as you can see in figure 2.

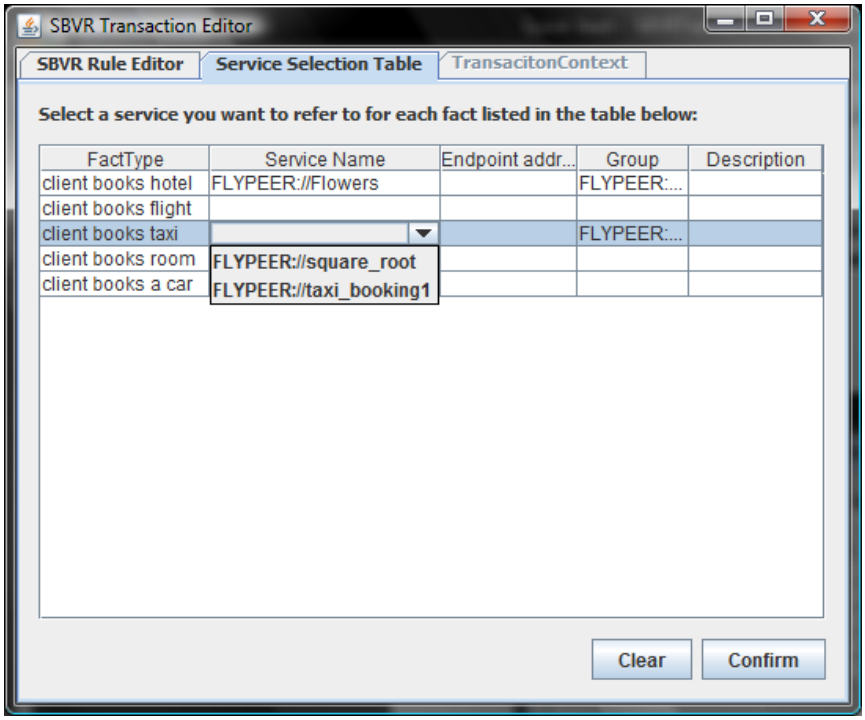

Figure 2

Information upon services are displayed to help the user in his choice.

If no service is found by the semantic search for a given fact type, then all available services will be displayed.

c)When all fact types are paired with a web service, the composition rules will are processed and filtered and the ***TransactionContext.xml*** file is displayed in TransactionContext tab (you can find it also into the *output* folder near the *SBVRTransacitonEditor.jar* file), as you can see in figure 3.

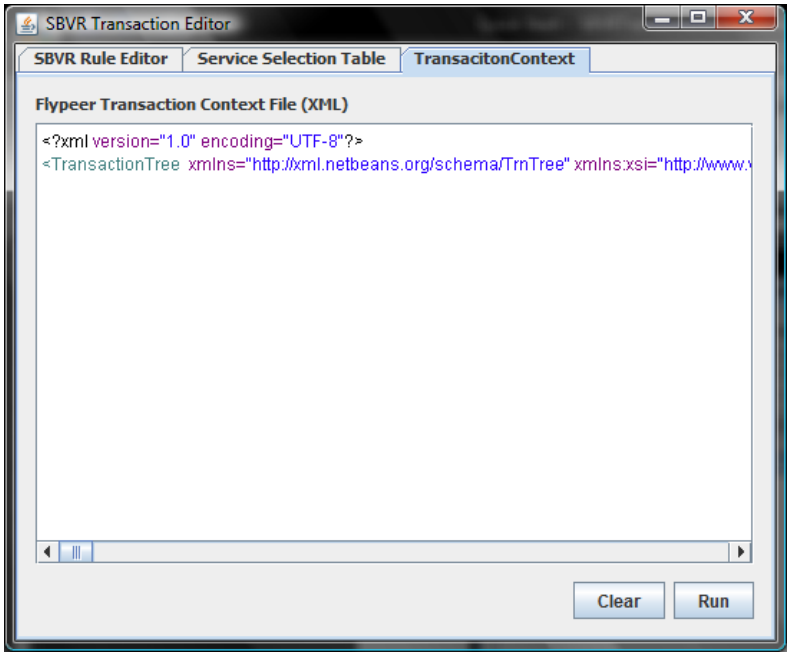

Figure 3

2. Supported rules

Flypeer supports three different basic type of composition: sequential, parallel and alternative.
For this reason three different kind of SBVR rules (called Production Rules) will be used in order to meet the requirements.
We will use the following patterns to express rules:

Rule pattern	Transaction Type
After *<fact type A>* then *<fact type B>*	Sequential Transaction
After *<fact type A>* then *<fact type B>* and *<fact type B>*	Parallel Transaction
After *<fact type A>* then *<fact type B>* or *<fact type C>*	Alternative Transaction

where "After, then, or, and" are keywords for SBVR.
Fact types, instead, are representations of actions the user want to be executed in a transaction (e.g. if a user type after *the customerX books a flight* then *the customex books the hotel* then a web service for booking a flight is executed and, sequentially after, is called the one for hotel booking).

3. Web service Semantic Annotations (aka sawsdl @link)

To add semantic information to web service for searching and consuming them, we will leverage upon a mechanism to enable semantic annotation of Web services descriptions, SAWSDL (Semantic Annotations for WSDL) precisely. SAWSDL permits the annotation of WSDL files with extra information regarding semantic aspects for each element in the WSDL file.
It is perfectly compatible with SOAP protocol.
In this version we will annotate only *<service>* elements in WSDL file, as in the following explanatory snippet of code:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:tns="http://www.example.org/FlowerDeliveryService/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         name="TaxiYYYBookingService"
         targetNamespace="http://www.example.org/FlowerDeliveryService/"
         xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
```

```
//….. ….

<service name="FLYPEER://taxi_booking1"
sawsdl:modelReference="URI_to_file\TaxiBookingSBVRVoc.xml">
      <documentation>This wsdl discribes taxi booking1
service.</documentation>
      <group name="booking"/>
   </service>
</definitions>
```

You can see highlighted in the text the needful sawsdl additions.

3.1.SBVR Service Vocabulary and Schema

The sawsdl:modelReference attribute points to a semantic description for the service (formally an ad-hoc XML file representing an SBVR Vocabulary for a service).

```
sawsdl:modelReference="URI_to_file\ServiceSBVRVoc.xml
```

It is possible to see a model of this file in the example described in the following of this guide.
A Schema for this XML file will be soon provided.

4. Example

4.1.Scenario

A very short and simple example has been provided.
The scenario is a hotel and taxi booking.
An hotel and taxi booking vocabularies (very simple) have been written, and they have been linked to the services via the wsdl files.
You can find these files in the ServiceDirectory folder inside the zip.

- *HotelABookingService.wsdl* is the descriptor for reservation service of the HotelA. We added semantic annotation to this service.
- *HotelBBookingService* is the descriptor for reservation service of the HotelA. We didn't add semantic annotation to this service
- *HotelReservationSBVRVoc.xml* is the xml file we used to annotate *HotelABookingService.wsdl*
- *TaxyYBookingService.wsdl* is the descriptior for two taxi booking services (FLYPEER://taxi_booking1 and FLYPEER://taxi_booking2). They are both annotated with an sbvr vocabulary.
- *TaxiBookingSBVRVoc.xml* is the xml file we used to annotate *TaxyYBookingService.wsdl.*

4.2. How to run the application

In order to run the example, create a folder on your hard drive. Copy in this folder the SBVRTransactionEditor.jar and extract the ServiceDirectory directory in the same folder (this operation is necessary at the moment because a working Service Directory is not implemented yet. we will search the wsdls in this folder).
Then launch the editor:

- **java –jar SBVRTransactionEditor.jar**

4.3. Operation

The user start typing the composition rules.

For example, a travel agency wants start a transaction booking the hotel for its client. Then a taxi must be booked.
Then it specifies first that a flight must be booked (no wsdl and no semantic annotation have been specified for this kind of service) and after that, in parallel hotel must be booked and a car must be rented(no wsdl and no semantic annotation have been specified for this kind of service).

Example of rules:

```
after the client books the room then the client books the taxi
after the client books the flight then the client books the hotel and the client
rents a car
```

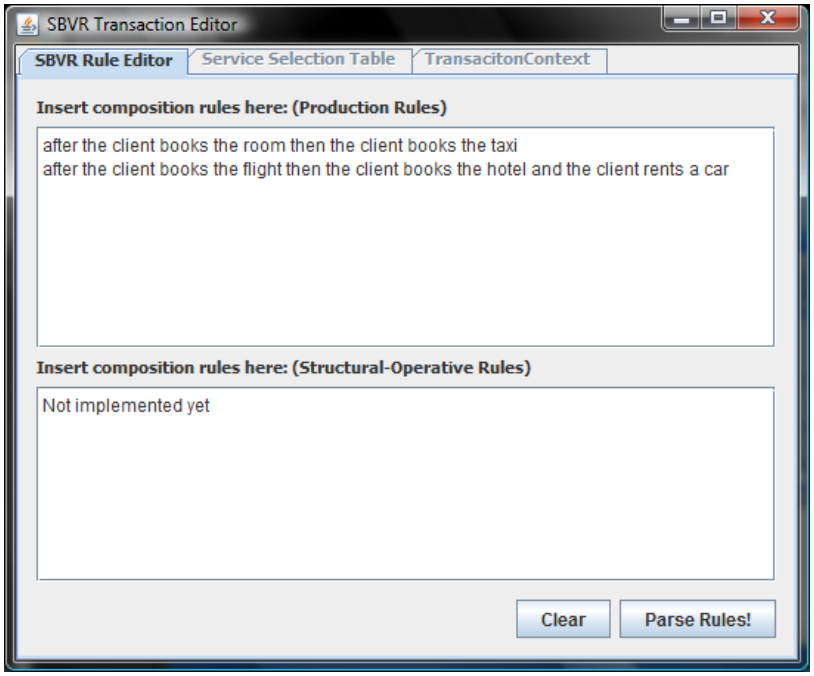

Figure 4

If rules are compliant with sintax, it will be parsed.
You can see the results in figure

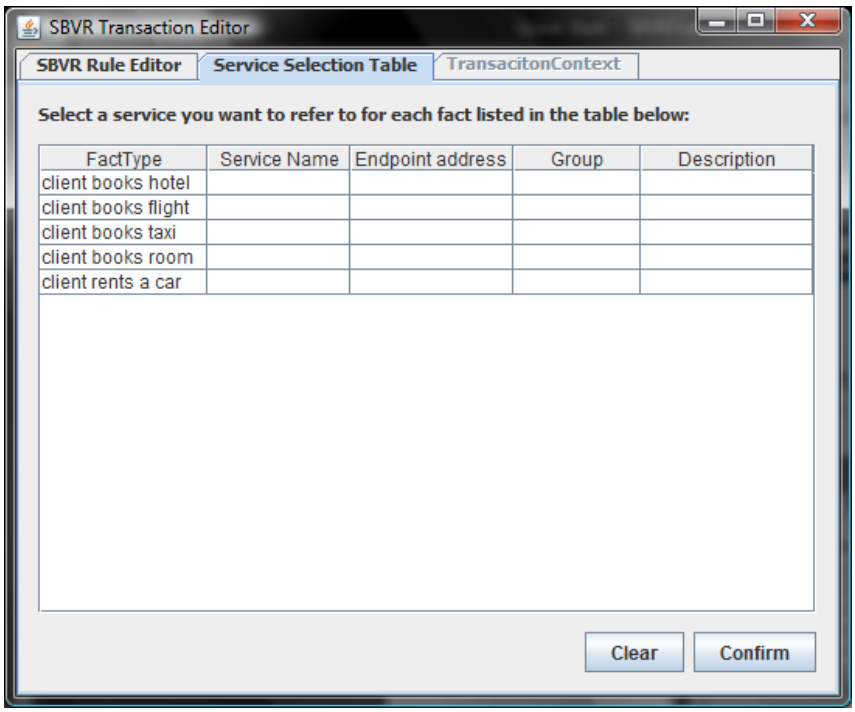

Figure 5

For fact " client book hotel" and "client book taxi" and "client book room",services are found via annotations.
For other facts ("rent a car" and "book a flight") all available services are displayed, as shown in figure 6

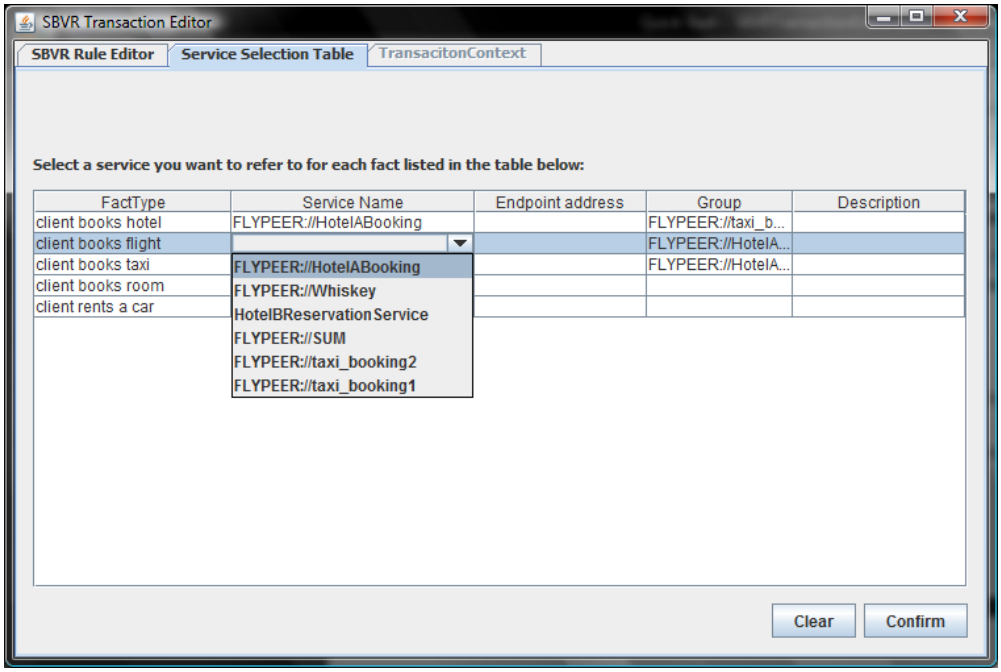

Figure 6

The aim is to enrich the vocabularies and parsing capabilities so user can express service in a simple and natural way.
Moreover, when structural rules will be added, the search results will be improved. In this way user could even not be obliged to choose service: they will be automatically associated.

At the moment, user must choose which service must be bound to each fact (if not you have got an error, as shown in figure 7.

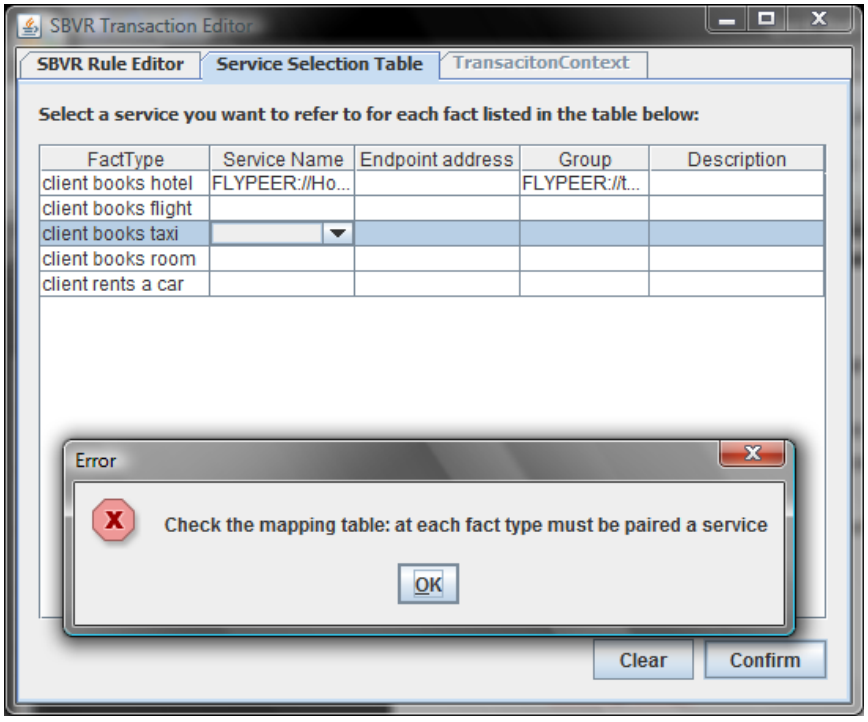

Figure 7

When each fact is paired with a service, then rules are evaluated and the TransactionContext.xml file will be set and displayed in the third tab, as shown in figure 8.

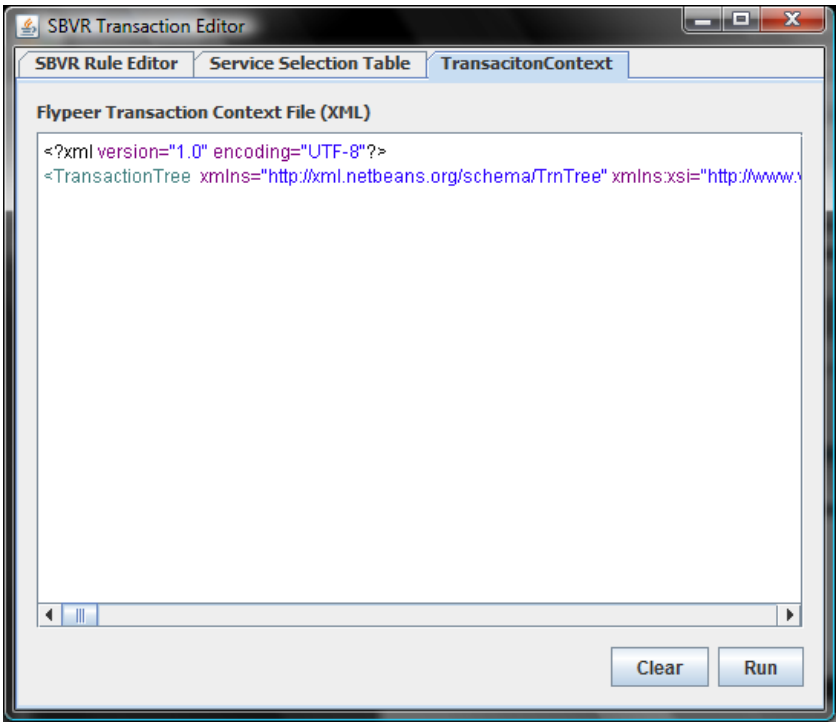

Figure 8

5. Missing Things

a) User can type very many rules and relationship between different rules must be analyzed.
(e.g. if user types two rules:
-after a then b
-after a then c

these must be transformed in:

-after a then b and c
)
No all possible combinations have been implemented yet. Re-composer module must be finished.

b) Fact type-service table in tab 2 is not working properly: no service information are currently displayed

c) A general parser (general SBVR parser) must be implemented to continue Alex Marinos work. (Pane 2 in SBVR Editor tab 1)

d) The error reporting mechanism must be improved;

e) Syntax highlighting, auto completion, and other nice similar features can be added to editor in pane 1 tab 1of the SBVR Editor to help user in typing rules.

f) in tab 3 the XML view has a bug: it displays the file in a single line. (It is maybe possible visualize the XML file with the XML Editor developed by SUAS?)

6. Known Bugs

6.1. The SubtransacitonId and WebServiceId attributes in the TransactionContext.xml file must be correctly generated (at the moment they are random()). It is necessary to know the algorithm used by Flypeer transaction engine in consuming these values.

6.2. WSDL files are at the moment searched in local ServiceDirectory folder. When an UDDI registry – like feature is implemented, network search will be added.