University of Pisa
MSc in Computer Engineering

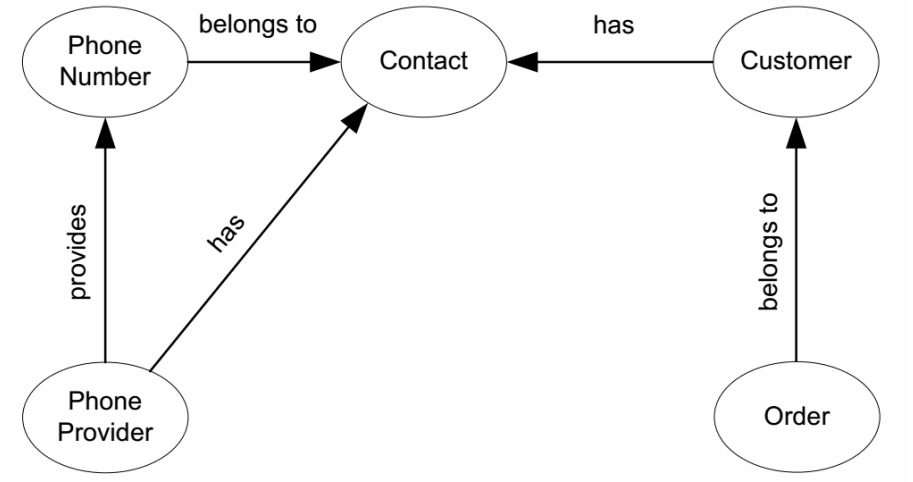
Process-driven Information Systems

LECTURE 23

http://www.iet.unipi.it/m.cimino/wdis/

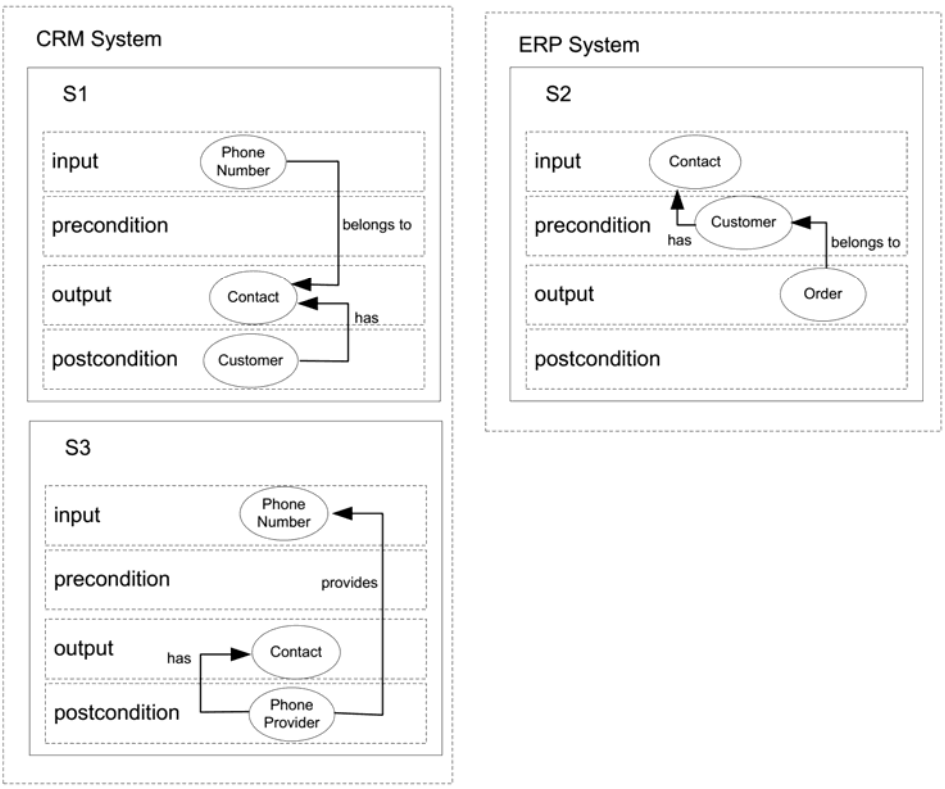
Mario G.C.A. Cimino
Department of Information Engineering

- Let us consider a call center domain, where phone calls by customers come in and call center agents serve these calls using an ERP and a CRM software systems.
- In a call center environment, a customer calls to request certain information. Using the phone number of the incoming call, the CRM gets hold of the customer address, which is, in turn, fed to the ERP to provide information on the customer.
- Another service may take a phone number as input and provide the address of the phone provider as output.



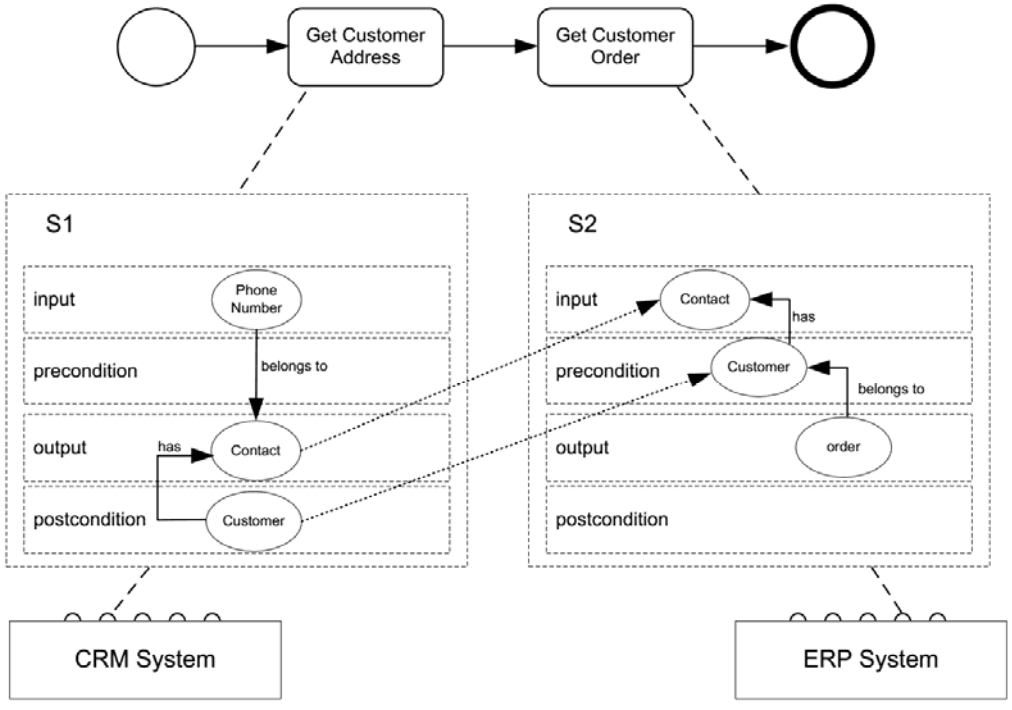
Domain ontology of call centre example

- Syntactically S3 is equivalent to S1 in terms of input-output data
- But semantically they returns different data.
- This functional difference is visible in semantic specifications only



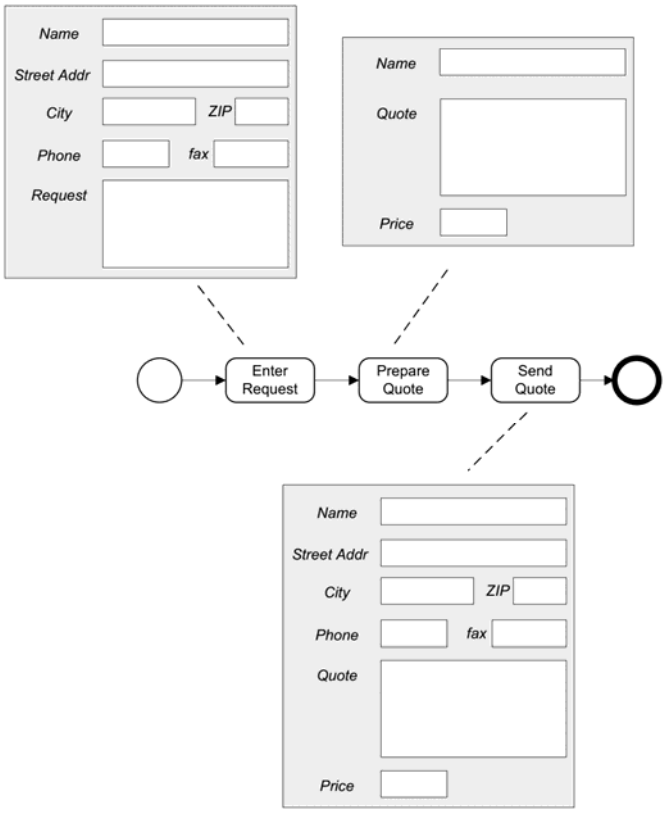
Semantic specification of services

- The semantic information is used to decide the semantic match of the two services, in the context of a given business process



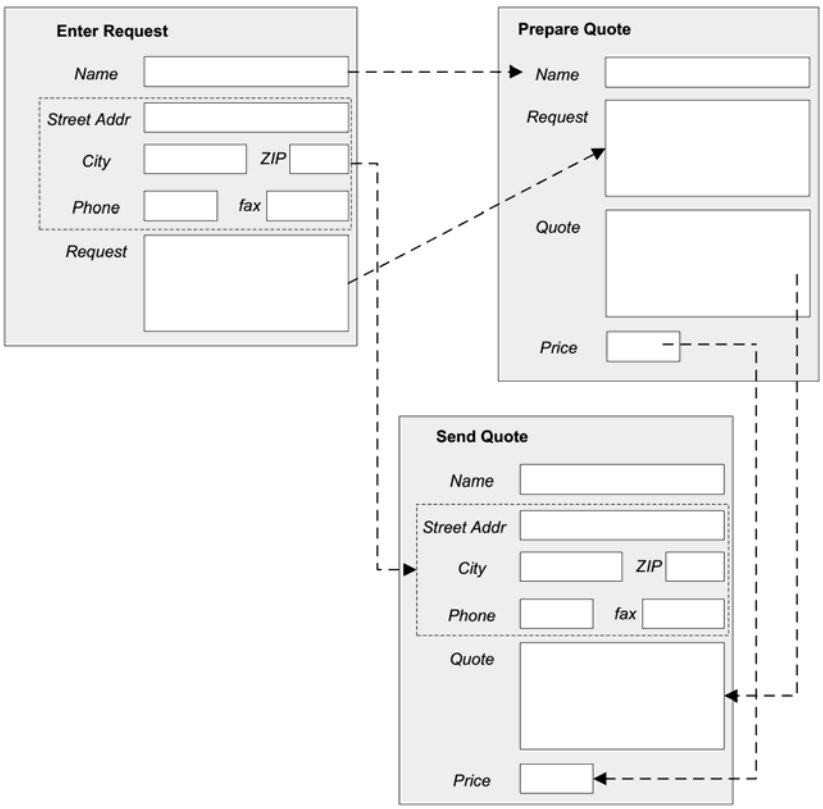
Semantic service composition

- Example: Knowledge-intensive BP are centered on data processed in the context of a particular case
- Examples of case: a product that is manufactured, the evaluation of a job application, the verdict of a traffic violation, the outcome of a tax assessment, the ruling for an insurance claim
- In traditional human interaction workflow, activities are statically ordered. However, if data constraints are represented, the process can behave more flexibly.



Motivating example case handling

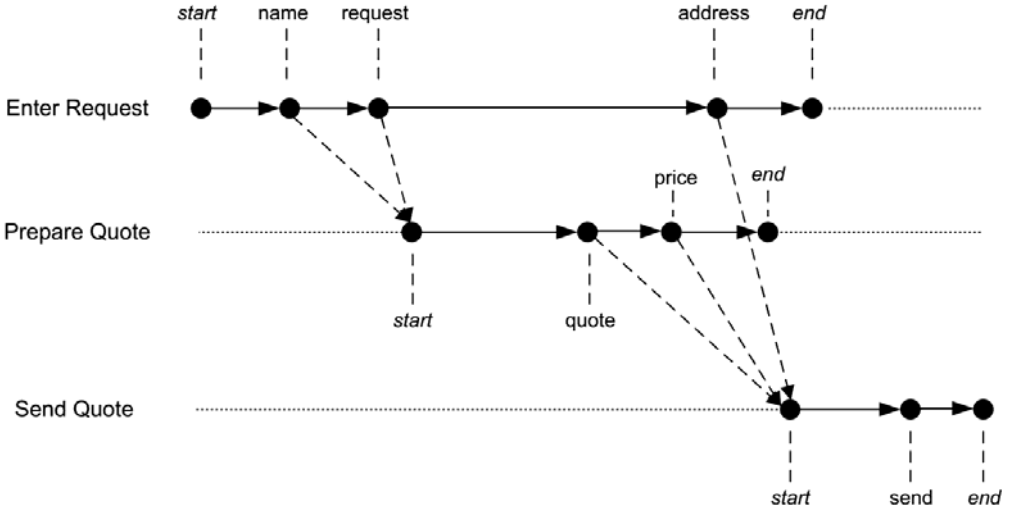
- *case models* allow more concurrency between activities.
- E.g. the preparation of the quote does not require the address information. Thus, this activity can start as soon as the enter request activity has provided the name and the request information



Data dependencies in case handling example

- The execution behavior of case handling can be represented by an event diagram.
- The case starts with an Enter Request activity. As soon as the name and request data fields are provided, Prepare Quote can start. Send Quote can start when the quote, including the price, is provided.

• The quote can only be sent after the request is entered, i.e., after the address information is available.



Temporal behaviour in case handling example: overall execution time is reduced, since prepare quote can start before enter request is completed

- To define such temporal constraints: the activity definitions are associated with data object definitions. The association is typically partitioned into two types: *mandatory and restricted.*
- If data object is **mandatory** for an activity, the data value has to be entered before that activity can be completed. However, it may also be entered in an earlier activity.
- Instead, a **restricted** association indicates that a data value can only be entered **during** a particular activity.
- Data objects may also be free, i.e., associated not with particular activities but with the overall case. Hence, they can be accessed at any time during the case execution.

It follows a summary of aspects to make a process executable

- **Process variables** are managed by the BPMS engine to allow data exchange between process elements. E.g. the purchase order in the order fulfillment process, represents a process variable.
- The **lifetime of a process variable** is confined to the life of the process instance in which the variable is created, and is only visible to the process level in which it is defined and to all its sub-processes. This means that a variable defined in a sub-process is not visible in the parent process.
- We need to assign a data type to each process variable to allow BPMS to interpret and manipulate these variables. In BPMN, the type of each process variable is specified as an XSD (XML Schema definition) type.
- The type of a variable can be simple or complex. Simple types are strings, integers, doubles (numbers containing decimals), Booleans, dates, times, etc. E.g. The object Stock availability can be represented as a process variable of type integer (representing the number of available units of a product).

- Complex types are hierarchical compositions of other types. A complex type can be used for example to represent a business document, such as a purchase order or an invoice.

a)

```
<complexType name="purchaseOrderType">
  <sequence>
    <element name="order">
      <complexType>
        <sequence>
          <element name="orderNumber" type="integer"/>
          <element name="orderDate" type="date"/>
          <element name="status" type="string"/>
          <element name="currency" type="string"/>
          <element name="productCode" type="string"/>
          <element name="quantity" type="integer"/>
        </sequence>
      </complexType>
    </element>
    <element name="customer">
      <complexType>
        <element name="name" type="string"/>
        <element name="surname" type="string"/>
        <element name="address">
          <complexType>
            <sequence>
              <element name="street" type="string"/>
              <element name="city" type="string"/>
              <element name="state" type="string"/>
              <element name="postCode" type="string"/>
              <element name="country" type="string"/>
            </sequence>
          </complexType>
        </element>
        <element name="phone" type="string"/>
        <element name="fax" type="string"/>
      </complexType>
    </element>
  </sequence>
</complexType>
```

b)

```
<purchaseOrder>
  <order>
    <orderNumber>15664</orderNumber>
    <orderDate>2012-10-23</orderDate>
    <status>confirmed</status>
    <currency>EUR</currency>
    <productCode>345-EAR</productCode>
    <quantity>10</quantity>
  </order>
  <customer>
    <name>John</name>
    <surname>Brown</surname>
    <address>
      <street>8 George St</street>
      <city>Brisbane</city>
      <state>Queensland</state>
      <postCode>4000</postCode>
      <country>Australia</country>
    </address>
    <phone>+61 7 3240 0010</phone>
    <fax>+61 7 3221 0412</fax>
  </customer>
</purchaseOrder>
```

The XSD describing the purchase order (**a**) and one of its instances (**b**)

Making processes executable on BPMS: Task variable

- Internal variables of each task, called data inputs and data outputs in BPMN, need to refer to an XSD type defining their structure. Differently from process variables, they are only visible within the task (or sub-process) in which they are defined.
- E.g. a data input for task “Check stock availability” in order to store the content of the purchase order.
- The association between data objects and task data inputs/outputs is defined via a data mapping In most cases, the BPMS will automatically create all the tedious data mappings between data objects and tasks.
- BPMN relies on XPATH as the default language for expressing data assignments, other languages can be used like Java Universal Expression Language (UEL) or **Groovy**.
- E.g. *Activiti BPM* supports UEL, *Bonita Open Solution* and *Camunda Fox* support **Groovy** while BizAgi’s BPM Suite supports its own expression language.

Making processes executable on BPMS: Service Task

- A service task specifies how to communicate with the external application that will execute the task. It is required is that the external application provides a service interface that the service task can use.
- A service interface contains one or more service operations, each describing a particular way of interacting with a given service. For example, a service for retrieving inventory information provides two operations: one to check the current stock levels and one to check the stock forecast for a given product.
- An operation can either be in-out or in-only, thus expecting a request/response message or request only. Each message of a service operation needs to reference a message in the BPMN model, so that it can be assigned an XSD data type.
- For each interface, a concrete implementation is defined: which communication protocols are used b the service and where the service is located in the network. By default, BPMN uses Web service technology to implement service interfaces, and relies on SOAP/REST and WSDL to specify this information.

...on BPMS: Send/Receive Tasks, Message/Signal Events 13 of 24

- They work similarly. A send task is a special case of the service task: it sends a message to an external service using its data input, but there is no response. A receive task waits for an incoming message and uses its data output to store the message content.
- A receive task can be used to receive the response of an asynchronous service which has previously been invoked with a send task. The asynchronous service is provided by the consumer.
- Accordingly, in the send task the producer process acts as the service requester sending a request message to the consumer. In the receive task the roles get swapped: the producer acts as the service provider to receive the response message from the consumer.
- This pattern is used for long-running interactions, where the response may arrive after a while. The drawback of using a synchronous service task in place of a send-receive is that this task would block the process to wait for the response message.
- Message and signal events work exactly like send and receive tasks

... on BPMS: Script task and User task 14 of 24

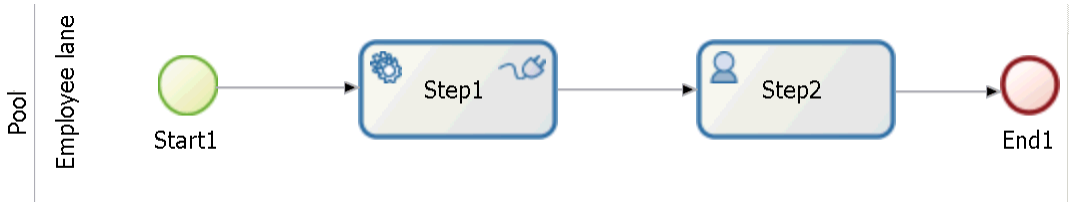
- For **script task**, provide the snippet of code that will be executed by the BPMS, in a programming language such as JavaScript or Groovy.
- The task data inputs store the parameters for invoking the script while the data outputs store the results of script execution.
- For **user task,** specify the rules for assigning work items of this task to process participants at runtime, the technology to communicate with participants and the details of the user interface to use.
- Also, define data inputs to pass information to the participant, and data outputs to receive the results. Process participants are member of a resource class, e.g. sharing certain characteristics, holding the same role or belonging to the same department or unit.
- Specify the implementation technology used to offer the work item to the selected participant(s): (i) how to reach the participant (e.g. via email or worklist notification), (ii) how to render the content of the task data inputs on screen (e.g. via web forms organized through screenflows), (iii) the strategy to assign the work item to a single participant out of the assignment expression (e.g. assign it to the order clerk with the shortest queue or randomly).

… on BPMS: Task, Event and Sequence Flow Expressions 15 of 24

- To write expressions for the attributes of tasks and events, and for the sequence flows bearing conditions. E.g. in a loop task we need to write a boolean expression implementing the condition “until response approved”. For timer events. E.g. “Friday afternoon”. It can be provided a temporal expression in the form of a precise date or time, a relative duration, or a repeating interval.
- These expressions can be linked to data elements and instance properties so as to be resolved dynamically at execution. For example, we can set an order confirmation timeout based on the number of line items in an order.
- To write a Boolean expression to capture the condition attached to each sequence flow following an (X)OR-split. For example, condition “product in stock” after the first XOR-split in the order fulfillment example can be implemented as an XPATH expression.
- There is no need to assign an expression to a default sequence flow, since this arc will be taken by the BPMS engine if the expressions assigned to all other arcs emanating of the same (X)OR-split are false.

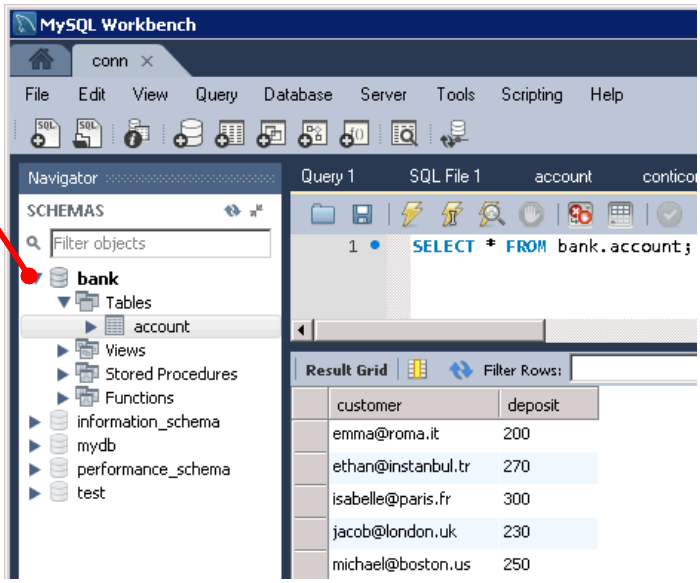
Making processes executable on BPMS: system binding 16 of 24

- The most BPMS-specific properties to configure in order to make a process model executable are those of user tasks and those to link the executable process with the enterprise systems (system binding).
- BPMSs offer a range of predefined service task extensions, called service *adapters or connectors*: performing a database lookup, sending an email notification, posting a message to Twitter or setting an event in Google Calendar, reading or writing a file and adding a customer in a CRM system.
- Each adapter comes with a list of parameters that we need to configure. BPMSs provide wizards with capabilities to auto-discover some of the parameter values. For instance, to use a database lookup we need to provide the type of the database server (e.g. MySQL, Oracle DB) and the server’s URL, the schema to be accessed, the SQL query to run and the credentials of the user authorized to run the query.
- E.g. instead of implementing “Check stock availability” as a service task, use a generic database lookup adapter if available. The task “Notify unavailability to customer” and “Request shipping address” can be implemented via email adapters, without dedicated email services.

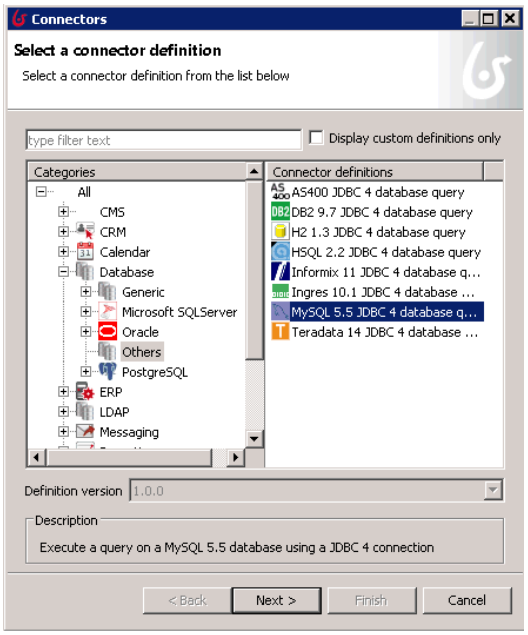


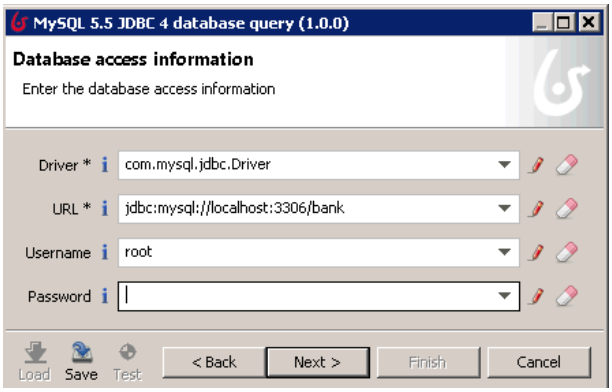
1. **Create the diagram above (for detailed steps see the first tutorial):**
2. New Diagram > complete the flow with the toolkit leaving the default task types.
3. Select *Step1* > *General* Tab > *Task type:* Service.
4. Select *Step2* > *General* Tab > *Task type:* Human.
5. Click on *Save* in the cool bar.
6. **Create the process variables:**
7. Select *Pool* > *Data* Tab > *Process Variables: Add* > Name: *customer* > *Finish & Add* > Name: *deposit* > *Finish*
8. ***Create the pool form***
9. Select *Pool* > Tab *Execution* > Instantiation form > 6.x
10. Tab *6.x Application* > *Add* > Select Tab *Process variables* > Select *deposit*, and *mandatory* > *Finish*
11. ***Create the Step2 form***
12. Select *Step2* > Tab *Execution* > form > 6.x
13. Tab *6.x Application* > *Add* > Select Tab *Process variables* > Select *customer*, and *read only* > *Finish*

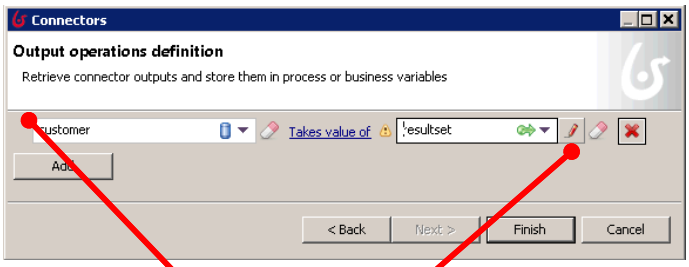
14. **Create the MySQL Database:**
15. 1st method: import the file *bank-dump.sql* into a MySQL server.
16. 2nd method: download the file www.iet.unipi.it/m.cimino/wdis/res/dbms.zip and extract it on C:\wdis. Finally, click on C:\wdis\mysqlStart
17. **Access the Database with MySQL client:**
18. Click on C:\wdis\mysqlClient6.1 > Click on the "+" icon close to MySQL connections > enter a name and click OK.
19. Select the *bank* schema > *Tables* > *account* > right click > Select rows.
20. **Create the DB Connector:**
21. On Bonita, select *Step1* > Tab *Execution* > Connectors out (*) > *Add* > Categories: *Database* > *Others* > *Connector definition* > *MySQL 5.5* JDBC 4… > Next
22. Name: *dbconn1* > *Next.* Enter URL: *jdbc:mysql://localhost:3306/bank* Username: *root* Password: *Next*



(*) *Connectors out* are carried out at the end of the step, whereas *Connectors in* at the begin of the process.





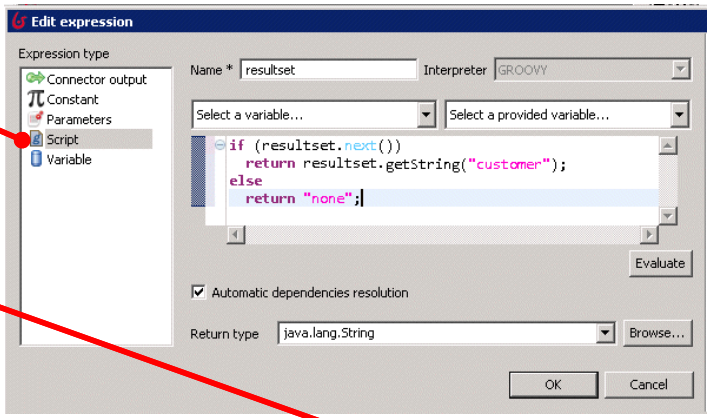


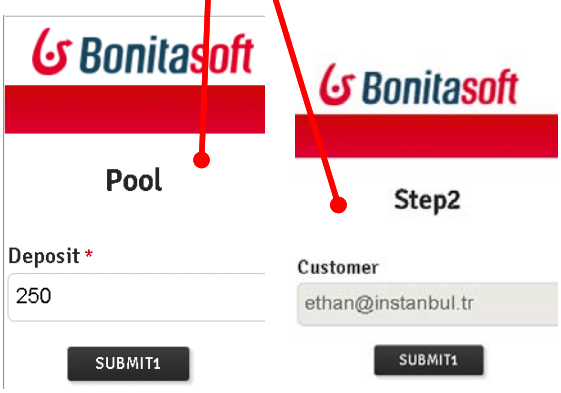
23. Enter the query
24. *SELECT * FROM account WHERE deposit > ${deposit};*
    (for autocompletion of variables press CTRL + SPACE)
23. Select *Next* > *Scripting Mode* > *Next* > Select target: *customer*
24. Click on the pencil icon to open the Groovy editor.

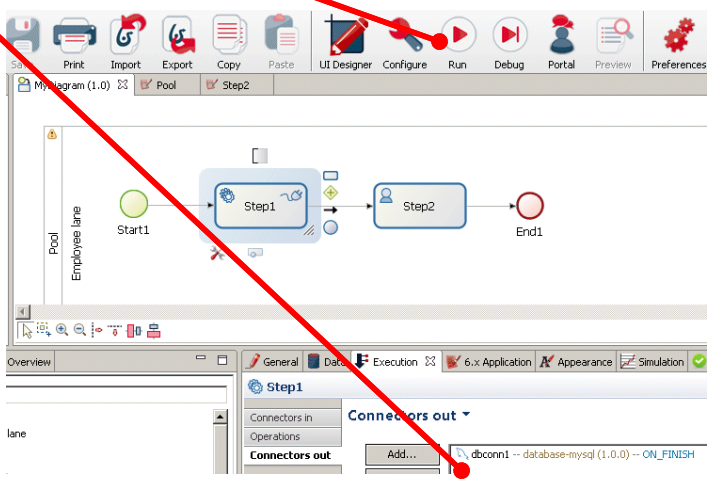
28. Expression type: *Script*
29. In the text area enter

```
if (resultset.next())
 return resultset.getString("customer");
else
 return "none";
```

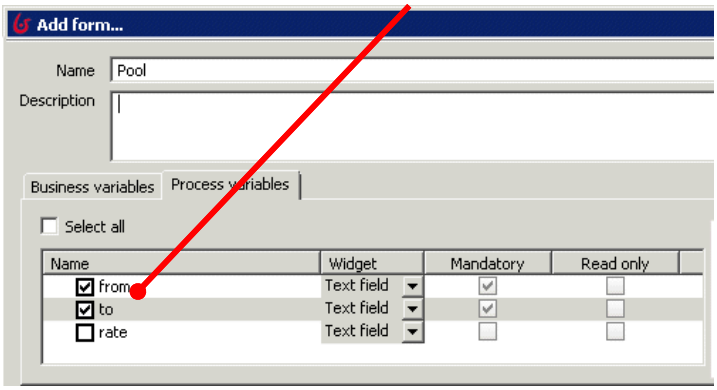
28. Click on OK > Finish.
29. *Click on Start button in the coolbar*
30. The Bonita launches the browser
31. Enter a deposit and SUBMIT
32. At Step 2, a customer with more than the deposit will be shown







1. **Remove the DB connector**
2. Select *Step1* > Tab *Execution* > Connectors out > *Remove*
3. **Remove the Process Variables**
4. Select *Pool* > Tab *Data* > Process variables > select *customer* > Remove > OK, select *deposit* > Remove > OK.
5. **Add the process variable *from, to, rate***
6. *Add* > Name: *from* > *Finish&Add* > Name: *to* > Finish & Add > Name: *rate* > *Finish*
7. **Update the Pool form**
8. Select *Pool* > *6.x Application* > Pageflow > Select *Pool* > *Remove. Add* > *Process variables* > Select *from* and *to*. Press *Finish.*



9. **Update the Step2 form**
10. *Select Step2 > 6.x Application > Pageflow > Select Step2 > Remove. Add > Process variables* > Select *rate* > *Finish*
11. **Add the WS connector**
12. Select *Step1 > Execution > Connectors out > Add.*
13. *Categories: SOAP WebService > Web Service Soap1.2 > Choose the NAME > conn2 > Next*

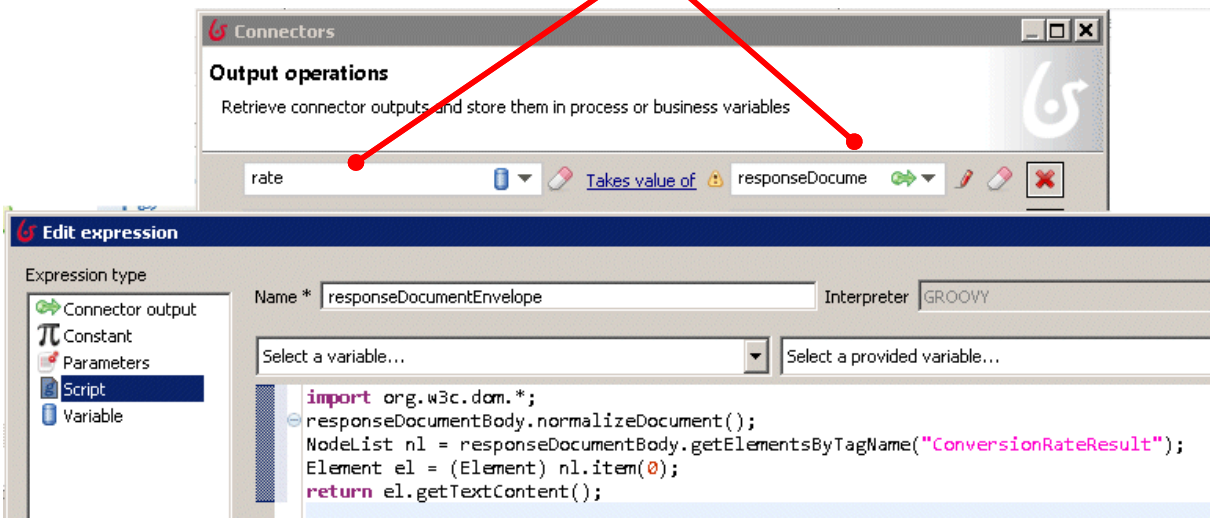
45. Name: *wsconn2* > *Next* > Enter parameters *
    Service NS: *http://www.webserviceX.NET/*
    Name: *CurrencyConvertor*
    Press *Next*
    Port Name: *CurrencyConvertorSoap12*
    EndPoint: *http://www.webservicex.net/CurrencyConvertor.asmx*
    Binding: http://www.w3.org/2003/05/soap/bindings/HTTP/
    Envelope:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:web="http://www.webserviceX.NET/">
  <env:Body>
    <web:ConversionRate>
      <web:FromCurrency>${from}</web:FromCurrency>
      <web:ToCurrency>${to}</web:ToCurrency>
    </web:ConversionRate>
   </env:Body>
</env:Envelope>
```

46. Next > Next > Returns body > Next > Output operations:
    (Ctrl + shift to find parameters values)

(*) Parameters are extracted by WSDL file *www.webservicex.net/CurrencyConvertor.asmx?wsdl* and by using a SOAP client software, e.g. *SoapUI*. This part is not covered. As an example some example of request and response is provided at the end of this tutorial.

47. Next > Next > Returns body > Next > Output operations:



48. Select *rate* on the left. Click on the pencil icon on the right. Edit Expression: *Script*. In the text area (Ctrl + shift to select parameters values if needed):

```
import org.w3c.dom.*;
responseDocumentBody.normalizeDocument();
nodeList nl =
responseDocumentBody.getElementsByTagName("ConversionRateResult");
Element el = (Element) nl.item(0);
return el.getTextContent();
```


Examples of XML request/response messages provided by *SoapUI*:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/s
   <soap:Body>
      <web:ConversionRate>
         <web:FromCurrency>EUR</web:FromCurrency>
         <web:ToCurrency>USD</web:ToCurrency>
      </web:ConversionRate>
   </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   <soap:Body>
      <ConversionRateResponse xmlns="http://www.webserviceX.NET/">
         <ConversionRateResult>1.0558</ConversionRateResult>
      </ConversionRateResponse>
   </soap:Body>
</soap:Envelope>
```

49. Click on *Start* button in the coolbar
50. The Bonita launches the browser
51. Enter *from* and *to* and SUBMIT
52. At Step 2, the conversion rate will be shown
53. Note: The WS may reply with "-1" when the WS is not available (this is frequent for free WS)