Dynamic/Automatic Composition of Web Services using SYNERGY

Adil Kabbaj

04/2019

1. Introduction

Composition of Web Services refers to the problem of combining several Web Services to answer a complex request. Automatic composition represents a great challenge in Service Oriented Architecture (SOA) and Web services technology in particular (Dustdar and Schreiner, 2005). As noted by Y. Syu and al., *« in recent years automatic service composition has been a popular research topic receiving a lot of attentions* » (Y. Syu and al., p. 290). Indeed, beside manual and semi-automatic composition, several solutions have been proposed to automate the process of Web Services's composition : by using workflow, automata, Petri Nets, semantic properties, AI planning techniques, genetic algorithms or multi-agent systems (MAS).

In the case of automatic composition of Web services, the user specifies a goal/request, and the composition engine will select the suitable services and compose them in order to provide an answer to the user's request (to satisfy the goal of the user).

In a semantic description of web services, input and output parameter's types are specified, as well as preconditions and effects. Services matching looks for semantic similarity between output parameter of service s1 and intput parameter of service s2.

Our purpose in this document is to show how SYNERGY language can be used (with some extensions) as an engine for automatic Web services' composition.

In the remainder of this document, we will focus on the work of P. Wang and al. : *« Automated web service composition supporting conditional branch structures »* (P. Wang and al., 2014).

2. Automatic web service composition with user preference and conditional information

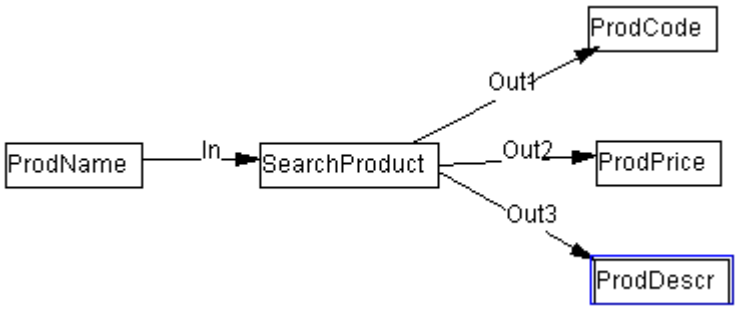
Wang and al. note that « *According to most of the existing methods, the process model of composite service includes sequence constructs only* » (Wang and al., p. 121). They propose *« a method to introduce conditional branch structures into the process model of composite service when needed, in order to satisfy users' diverse and personalised needs and adapt to the dynamic changes of real-world*

environment. » (p. 121). How to use control constructs (like If-Then-Else, Iterate, Repeat-While, Repeat-Until, etc.) to express user preferences and needs in automatic composition of web services is a new challenge. The authors focus on the use of conditional structures. They offer as an example the case of online shopping involving several tasks (product service, submitting an order, paying for the order and shipping) that can be performed by different services (pp. 123-124). In the case of payment service, we may have the following user preference : if the user has enough money he would pay for the order in full, otherwise, he would pay by instalments. The condition may be « AccountBalance ≥ AmountToPay ». Wang and al. note that : « *User preferences are a key component of web service composition. However, this component has been largely ignored in the existing approaches for service composition* » (p. 129). Automatic composition of web services should take into account user preferences which affect the process of service selection ; « *a user prefers a class of services over another according to certain conditions (e.g. 'Lucy prefers to go by air over car, if the driving time is greater than 4 hours.')* » (p. 129). The previous example was that : if AccountBalance ≥ AmountToPay then the user would prefer the PayInFull service, otherwise he would prefer the PayByInstalments service.

3. Using SYNERGY as an engine for automatic web service composition with user preference and conditional information

Let us start with the example provided by Wang and al. (pp. 135-136). They provide six services with their input and output parameters. Here is a reformulation in SYNERGY of the signature of these services :

- Search Product service has a Product Name (PN) as input and produce as outputs : Product Code (PC), Product Price (PP) and Product Description (PD) :



- Order Processing service has Product Code and Product Quantity (PQ) as inputs and Order Information and Payment as outputs :

- Account Query service has Account Number and Account Password as inputs and account balance as output :
  [AccountQuery]-
  <-in1-[AccNbr],
  <-in2-[AccPass],
  -out->[AccBalance]

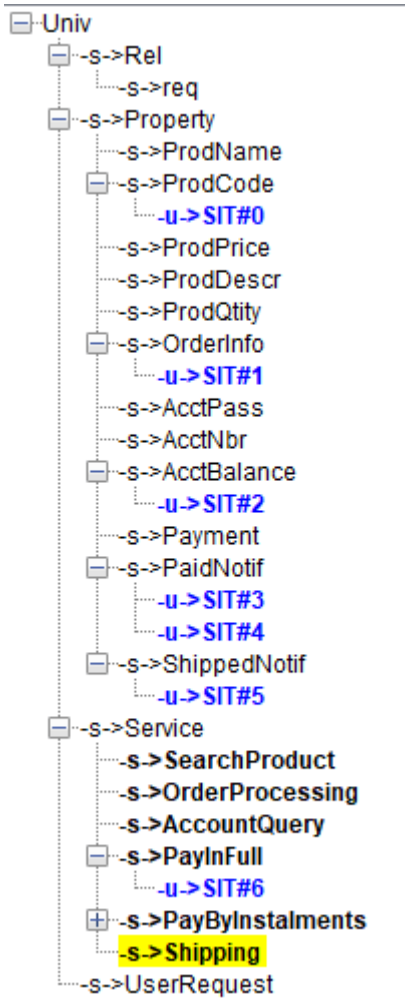
- PayInFull service has Account Number, Account Password and Payment as inputs and Paid Notification as output :
  [PayInFull]-
  <-in1-[AccNbr],
  <-in2-[AccPass],
  <-in3-[Payment],
  -out->[PaidNotification]

- PayByInstalments is the same as PayInFull :
  [PayByInstalments]-
  <-in1-[AccNbr],
  <-in2-[AccPass],
  <-in3-[Payment],
  -out->[PaidNotification]

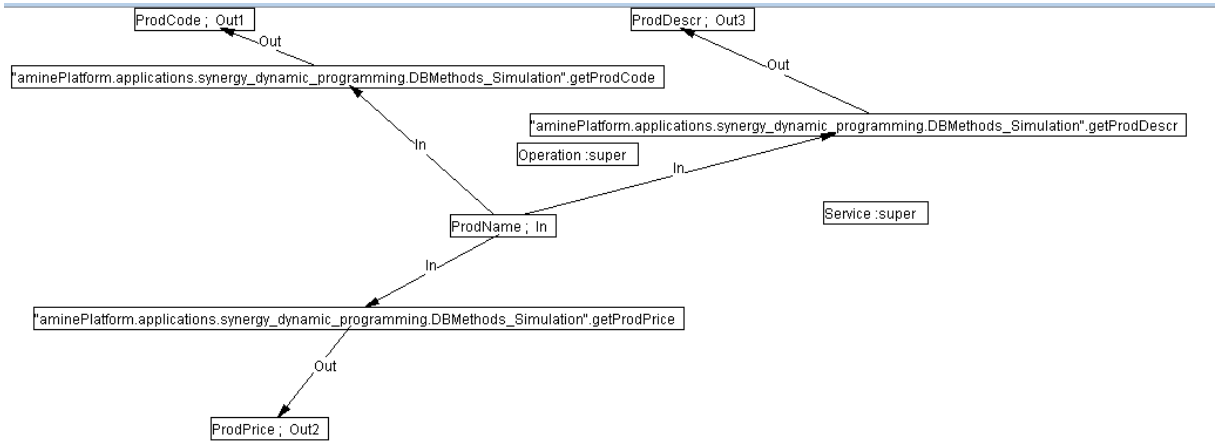
- Shipping service has Order Information and Paid Notification as inputs and ShippedNotification as output :
  [Shipping]-
  <-in1-[OrderInfo],
  <-in2-[PaidNotification],
  -out->[ShippedNotification]

User request is expressed as follows (the available inputs provided by the service requester are specified as well as the outputs expected) :
[UserRequest]-
<-in1-[ProdName],
<-in2-[ProdQtity],
<-in3-[AccNbr],
<-in4-[AccPass],
-out1->[OrderInfo],
-out2->[ShippedNotification]

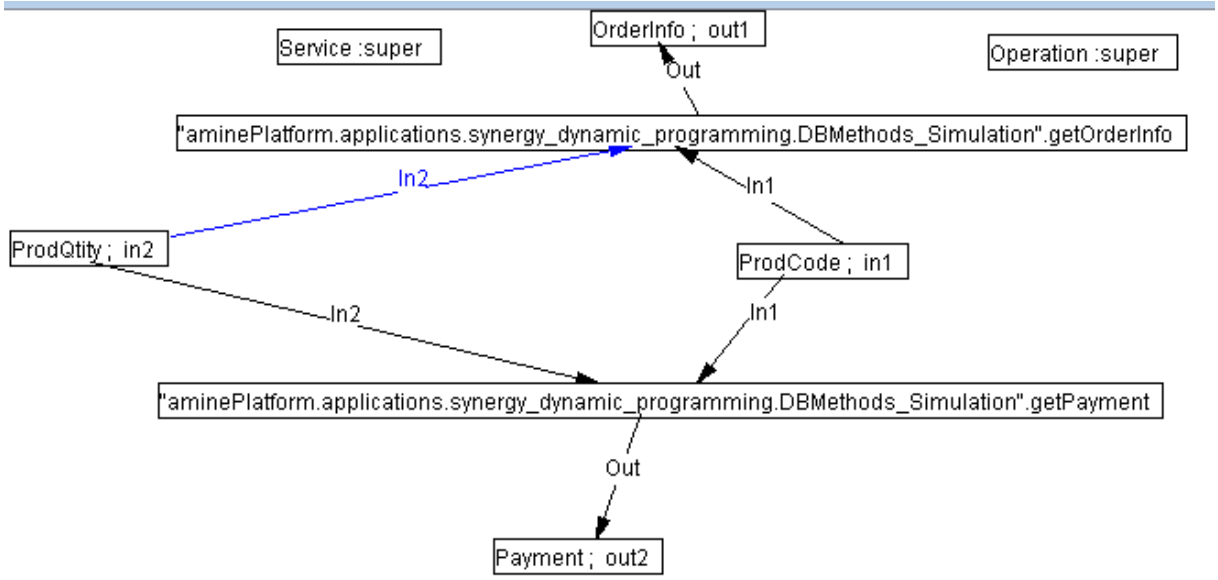
The ontology that supports the example and includes the types and relations used in the above specification of services is available at :
aminePlatform/samples/ontology/synergyOntology/ServiceOntologyExample.xml



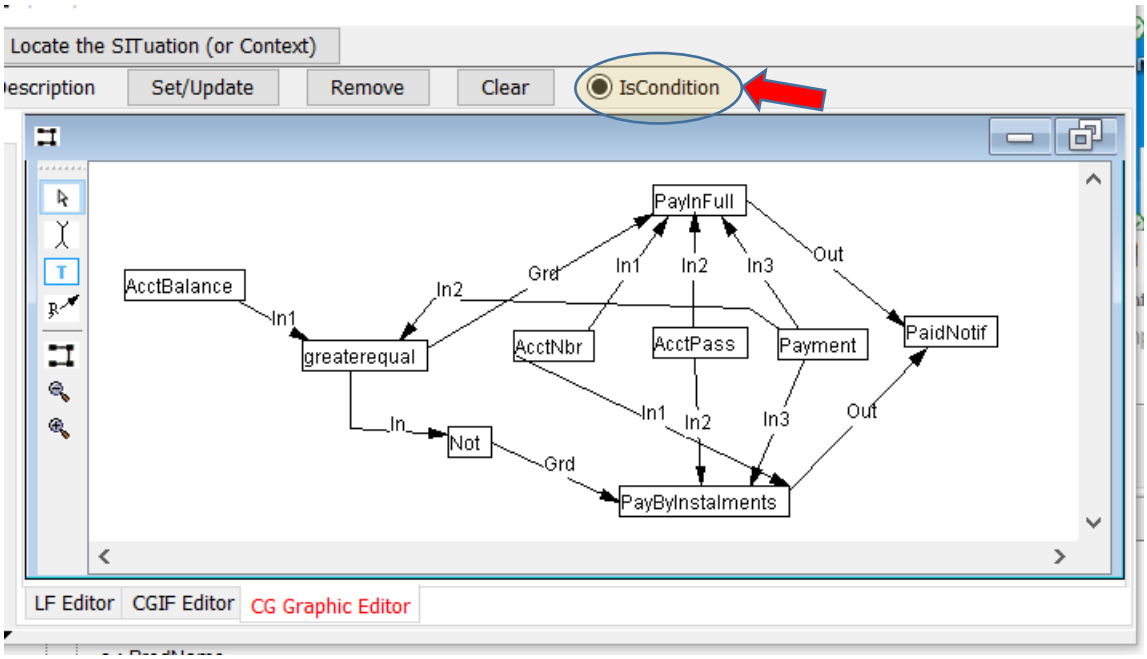
The signature of the six services are specified in our ontology as situations indexed under appropriate types. For instance the signature of SearchProduct is specified as a situation for ProdCode. The signature of OrderProcessing is specified as a situation for OrderInfo. Etc. The type Paid Notification (PaidNotif) has two situations, one for PayInFull and another for PayByInstalments.
Wang and al. didn't specify the content (definition) of each service. We supply this information in our reformulation of the example. For instance SearchProduct service will use Product Name to search (from a DB for instance) product information (ProdCode, ProdPrice and ProdDescr) :



The same approach is used in the definition of OrderProcessing : from ProdCode and ProdQtity, we can get Order Information and Payment.

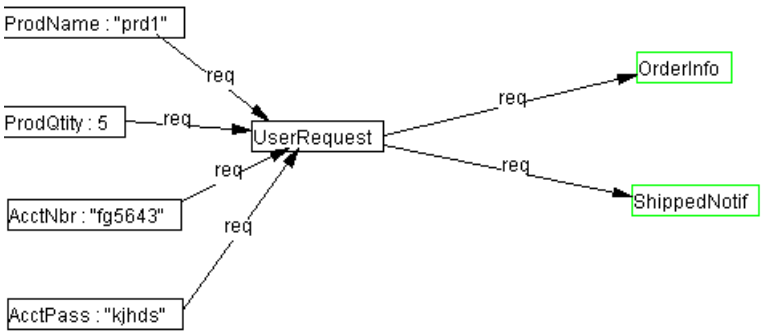


Let us consider now how user preferences, expressed as conditional use of some services, are implemented in our SYNERGY emplementation of automatic composition of web services. The two services PayInFull and PayByInstalments have a « ***condition/constraint situation*** » (SIT#6 in the ontology) that should be verified before the service can be used/activated/executed :



This « condition/constraint situation » specifies that we can produce Paid Notification (PaidNotif) by PayInFull service or PayByInstalements. The choice between the two services depends on the comparison : AccBalance ≥ Payment. If this condition is true, PayInFull can be triggered and activated and PayByInstalments will be blocked. If the condition is false, PayByInstalments can be triggered and activated and PayInFull will be blocked.

SYNERGY is extended to take into account this new kind of situation : « condition situation ». When a concept type is triggered (for instance PayInFull or PayByInstalments in our example), and the type has a « condition situation », then this situation should be joined to the current CG. In this way, the condition (represented by the situation) is considered by the composition process.
Let us see now how SYNERGY allows for the automatic composition of web services : the process starts with the user request :

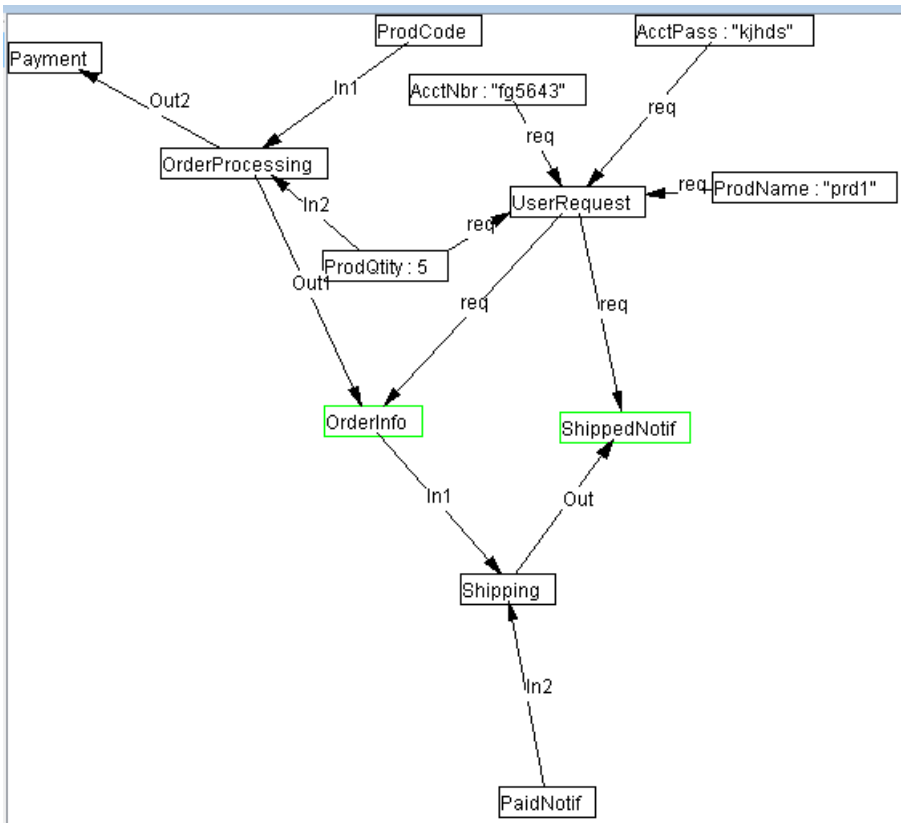


Synergy starts with this request as its current CG. Note that OrderInfo and ShippedNotif are triggered to initiate the automatic composition process. Note also that « UserRequest » is not a dynamic concept and « req » relations aren't

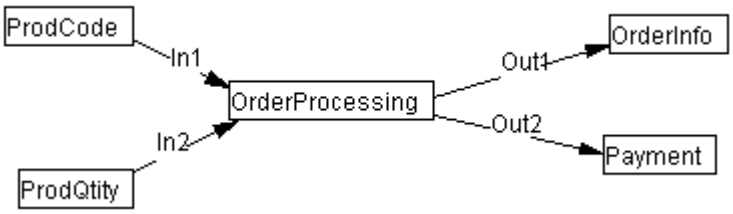
procedural relations. UserRequest provide only the information that the user provides ProdName, ProdQtity, AccNbr and AccPass as inputs data, and he/she asks for Order Information and Shipped Notification. UserRequest didn't specify how to compute outputs from inputs. The above CG is in fact equivalent to this one :

So, it is clear that SYNERG has to perform an automatic composition of web services, guided by the dependence between services and user preferences (expressed as « condition situation »), in order to discover how to go from the input data to the output/required data.

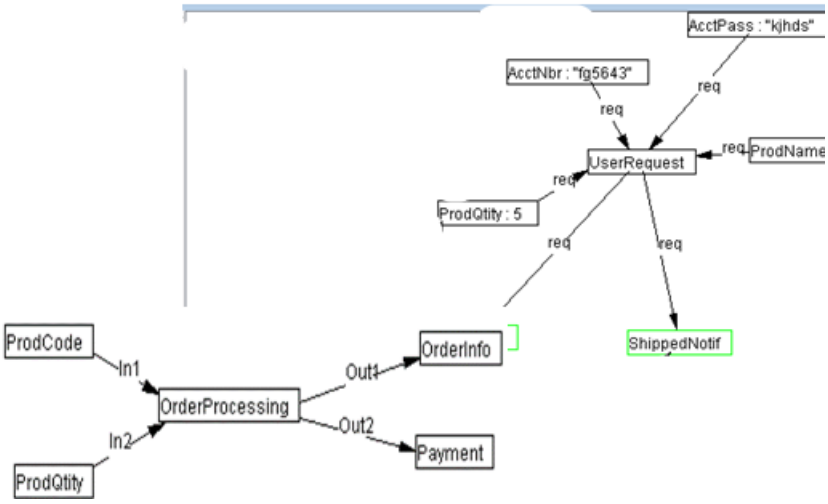
The option/parameter « CanJoinSituation » is selected to allows for the maximal join of a situation when a concept is triggered and its value can't be computed by the current CG. In this example, SYNERGY will join the situation associated to OrderInfo and the situation associated to ShippedNotif in the ontology. The result of these two maximal joins is this :



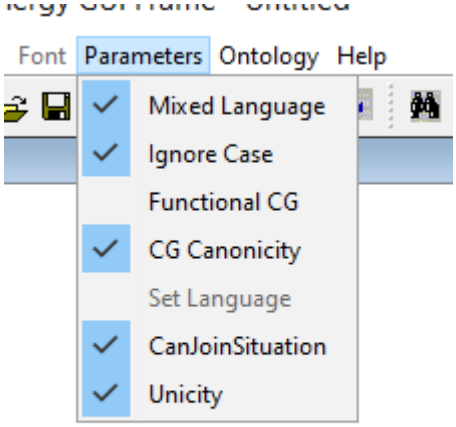
Here, a second extension to SYNERGY is applied : let us consider again the situation associated to OrderInfo that has been joined :



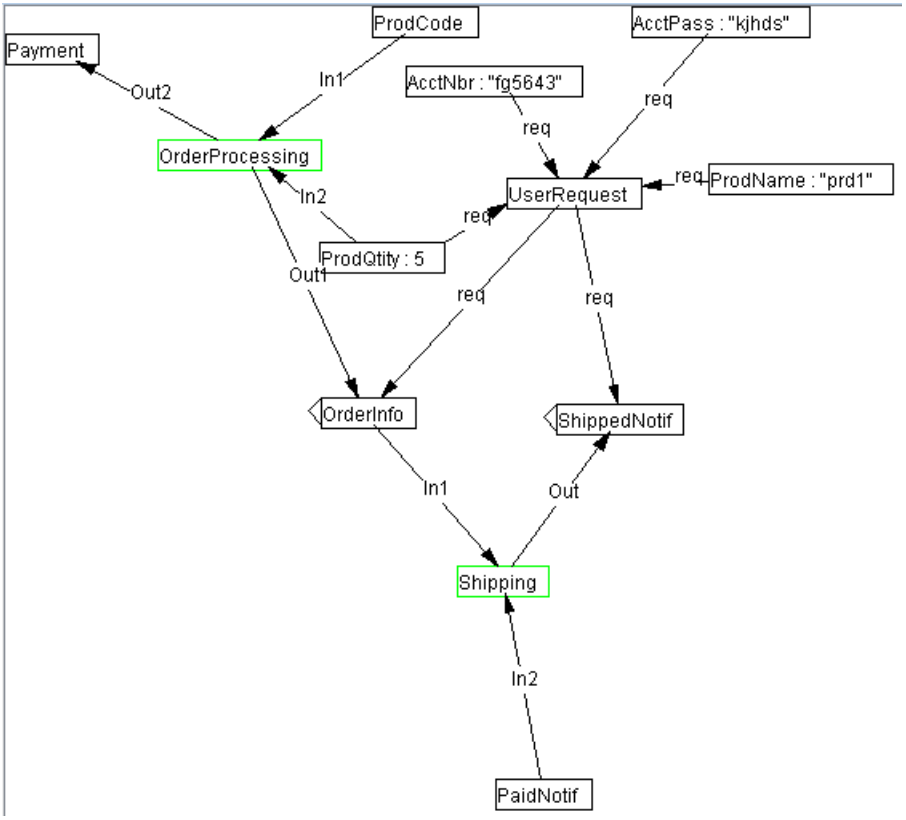
A « normal » maximal join of this situation to the current CG will produce this result :



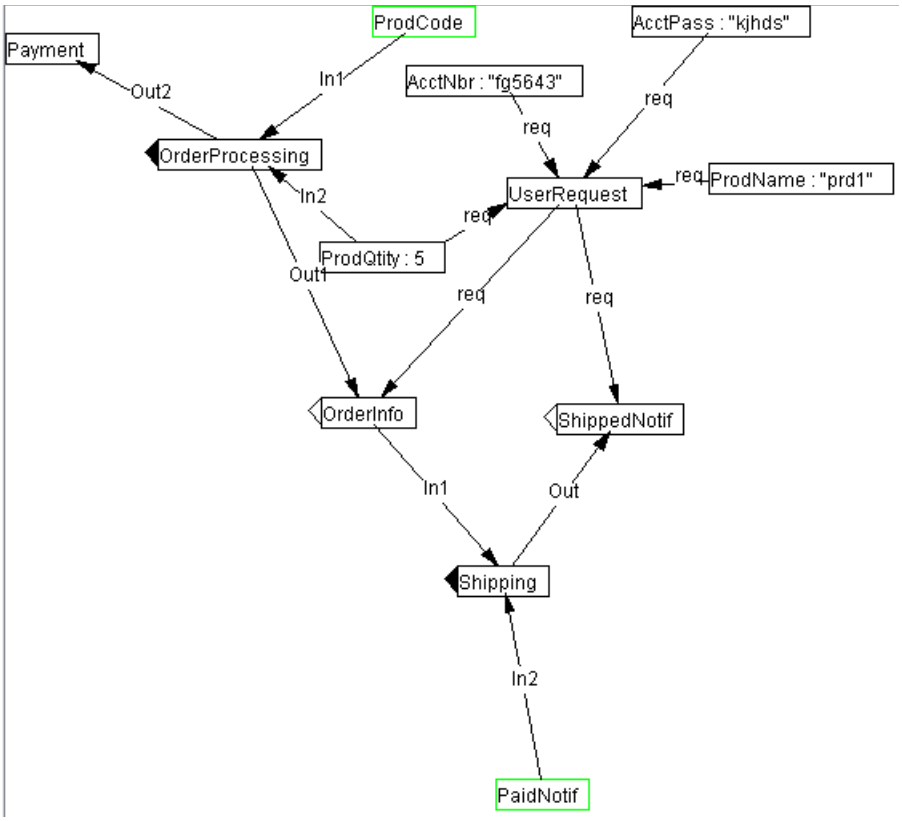
The maximal join will not consider that the concept [ProdQtity] (the second input argument to [OrderProcessing]) is the same as the concept [ProdQtity :5]. But this is exactly what we need in this case ; we want to specify the unicity of the implied concepts. This is done thanks to a new parameter « ***Unicity*** » : if true (selected), the unicity of the implied concepts is considered :



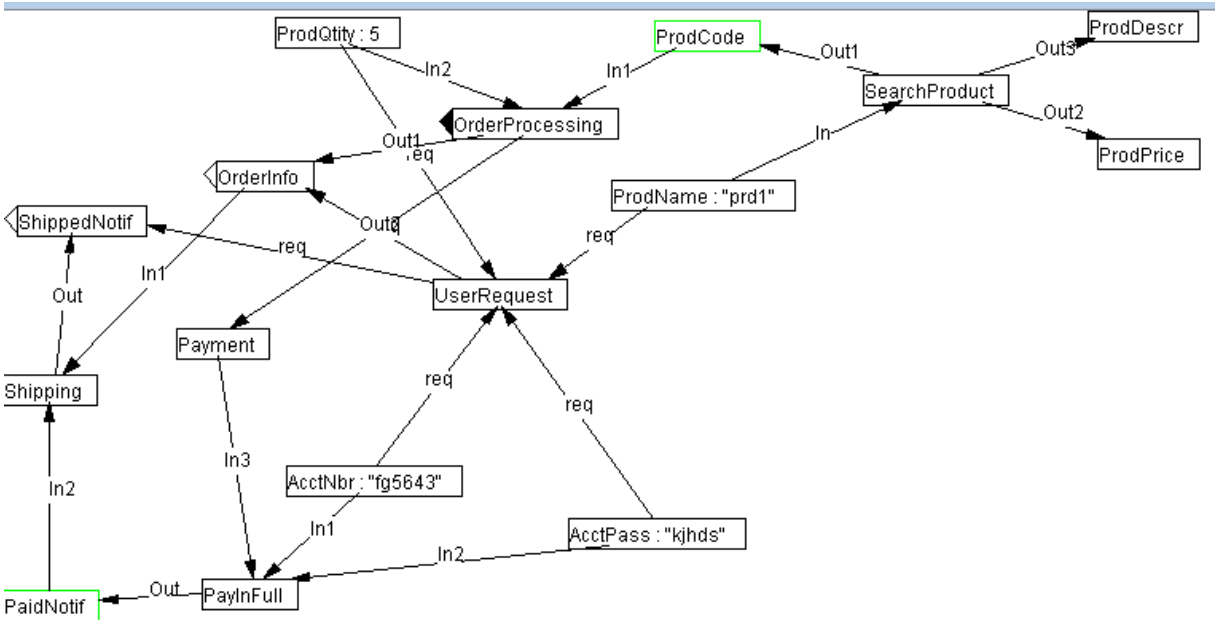
Let us follows now the execution process of SYNERGY :



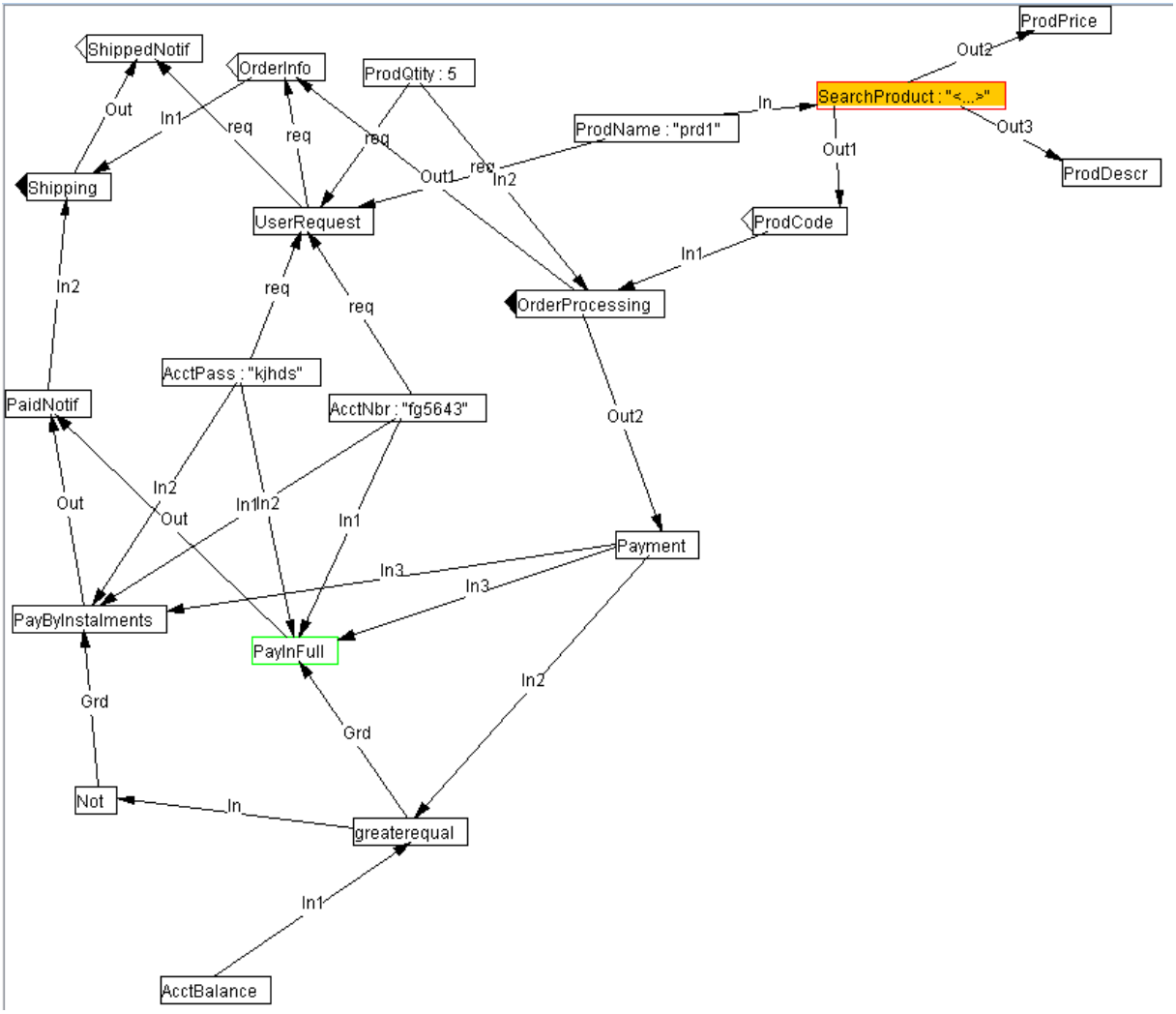
The triggering state of [OrderInfo] is propagated to [OrderProcessing], then to the first input argument of [OrderProcessing] : [ProdCode]. Backward propagation is done also for the second input of [Shipping] : [PaidNotif].



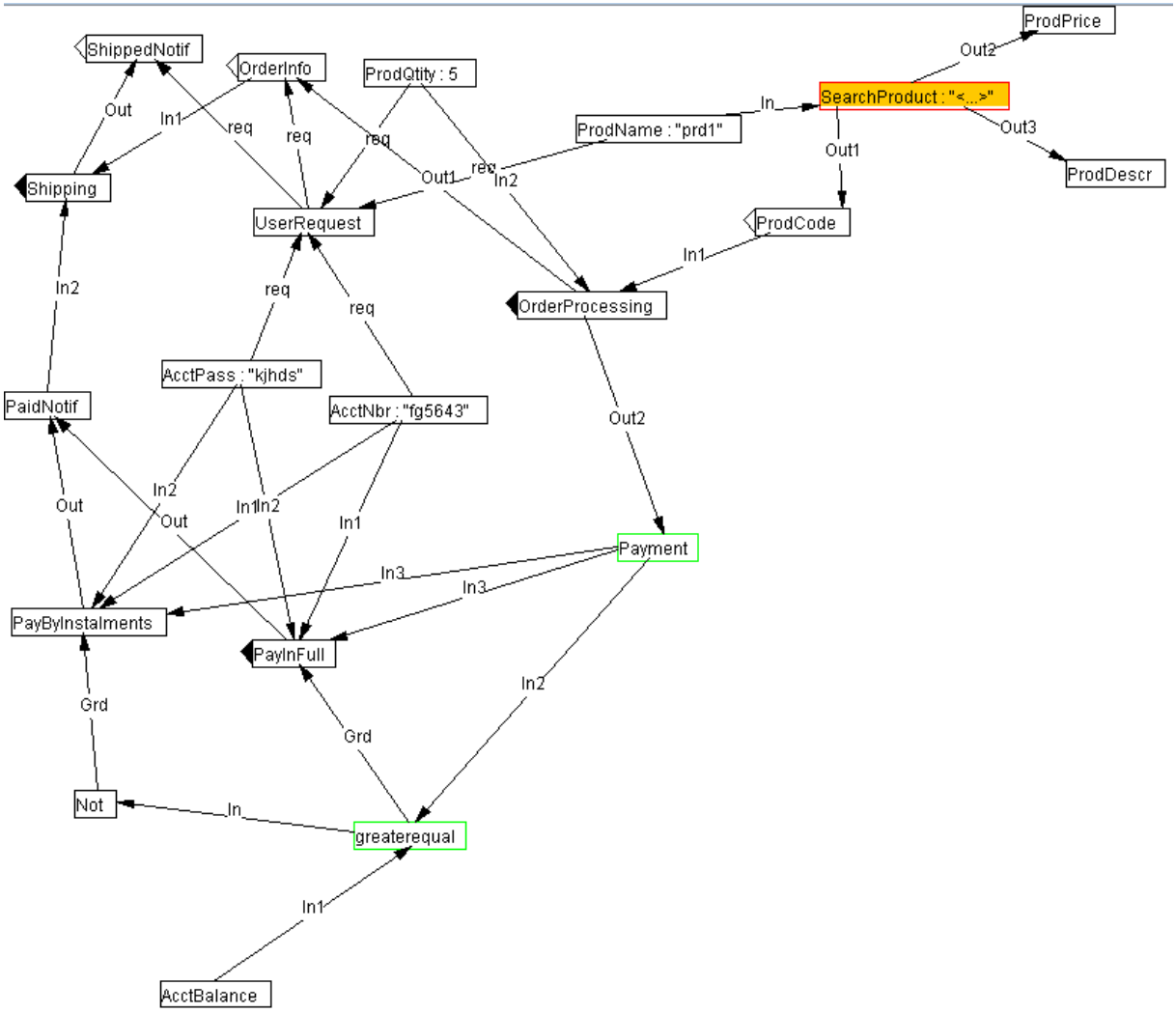
The situation associated to ProdCode will be joined, as well as the situation associated to PaidNotif :



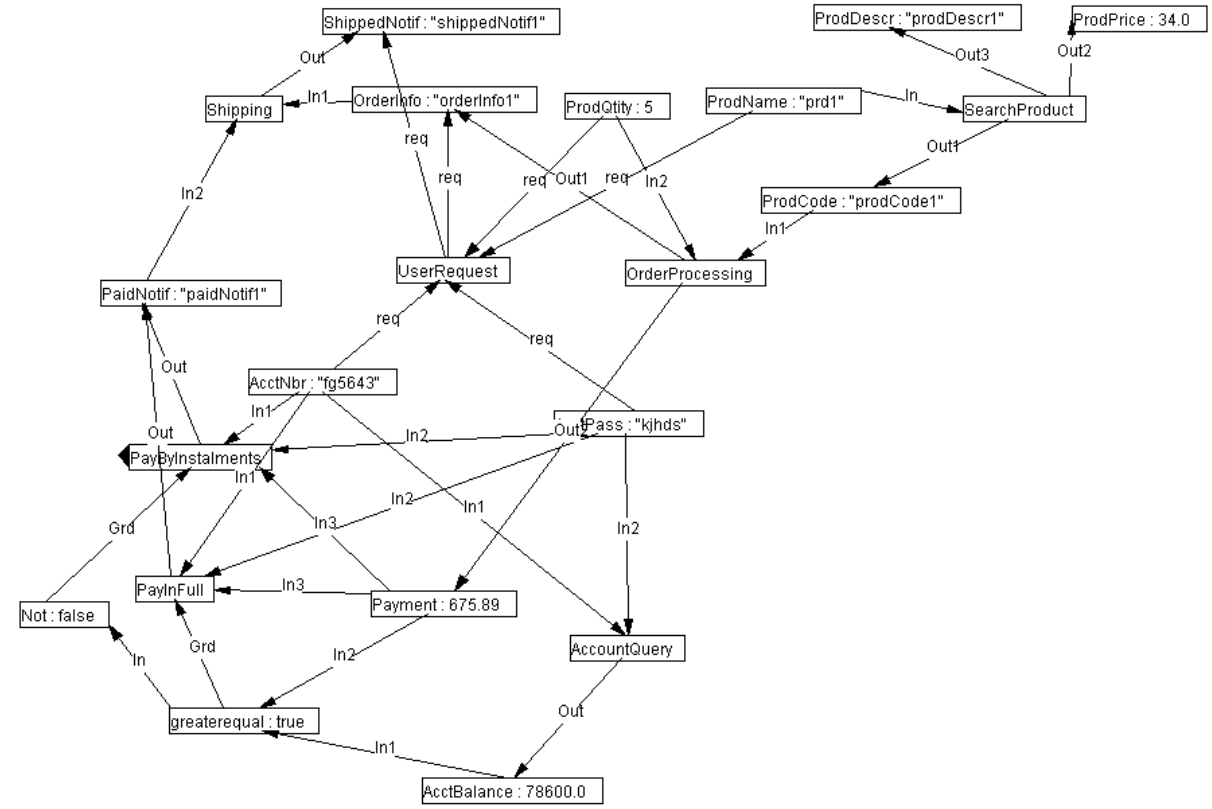
The trigger state of [ProdCode] is propagated to [SearchProduct] and the trigger state of [PaidNotif] is propagated to [PayInFull]. Being in trigger state, and since the concept type PayInFull (in [PayInFull]) has a « condition situation » associated to it, the « condition situation » will be joined to the current CG :



Activation by propagation will continue :



After some steps :



After some cleaning (for instance, eliminate referents), we get the result of the automatic composition of web services : a network of services that represents a plan to provide an answer for the user's request.

References

- Dustdar S, Schreiner W. 'A survey on web services composition', Int. J. Web and Grid Services, Vol. 1, No. 1, 2005.
- Syu Y., S-P. Ma, J-Y. Kuo, and Y-Y. FanJiang, A Survey on Automated Service Composition Methods and Related Techniques, in IEEE Ninth International Conference on Services Computing, 2012, pp. 290-297.
- Wang P., Z. Ding, C. Jiang and M. Zhou, *Automated web service composition supporting conditional branch structures*, Enterprise Information Systems, 8 :1, 121-146, 2014.