Deliverable 2:
Tools for negotiation mechanism specification and validation : Application on the Single Shot

Abstract.
The deliverable 2 concerns the Task 1.2. We illustrate the deliverable 1 methodology on two negotiation protocols. We also present different tools to design mechanisms and generate roles abstract behaviours. We end this deliverable by the first steps toward the tasks 2 and 3 which model generation.

Deliverable Id.:	**1.0**
Date :	**7/12/2007**
Classification:	Public/Private
Originating Partner :	
Author(s) :	**LIP 6 + IBISC**
Relevant Work package :	WP 1…
Relevant Task :	Task1.1…
Release Status :	Draft V0…
Project Co-ordinator:	Lip6
Partners:	IBISC

Table of Contents

1. Introduction

In this deliverable we will illustrate the methodology proposed in deliverable 1 by using two negotiation mechanisms provide by FT. The document is structured as follows:

Section 2 presents in detail the two mechanisms using UML specification.

Section 3 we present a tool called PI4SOA and then in section 4 we describe the single shot mechanism specification using such PI4SOA (steps and choice explanation).

Section 5 is dedicated to the choreography model generation using LTSA extension.

In section 6 we present a specific feature of PI4SAO tool, The compilation, which is the end point projection presented and explained in deliverable 1 and then we present a guide tour through tools that generate model of abstract BPEL. Section 7 is for concluding a archives details description.

2. FT Protocols Specification

FT provided two negotiation protocols that may be used in the Grid4All project. These protocols are also used to validate our approach. They consist in the "single shot protocol" and the "iterative protocol".

The first one was produced conjointly in order to fix terminologies and the second was fully produced by FT and validated by the LIP6 partner.

The specified protocols are represented using the AUML graphical methodology extending UML 2.0 sequence diagrams [Bauer05, Cabac03]. In both protocols, the same roles are involved, that is:

1. Buyer: sends bids to acquire items
2. Seller: sends offers to sell items
3. Auctioneer: receives bids
4. Initiator: the role that initiates the market, may be a buyer or a seller
5. Market initiator: creates and configures the market
6. Market: The market itself, manages bids and offers
7. SIS: Semantic Information Service which is a registry of markets
8. Agreement manager: builds contracts between winning seller and buyer

Both protocols are executed following three main steps:

i. Initialisation
ii. Auction
iii. Matching and agreement

1. Initialisation Step

The initialisation step is the same in both protocols. It consists in the protocol represented in figure 1. This figure shows that an auction is initialised by the initiator role. To do so, the latter contacts the market initiator by sending the information related to the market configuration. This information includes the "stop-registration" and the "stop-bidding" timeouts values, the market channel, etc.

If the received parameters include errors, the market initiator can throw an exception (represented by the option rectangle in figure 1).

The market initiator then creates the market role and sends the market parameters to it. Finally, the market initiator advertises the market to the SIS.

2. Discover and auction step

Participating in an auction implies for buyers and sellers to look for a market that fits to their needs, ex. a market where an item a buyer is interested in is sold. To this aim, each participant sends a request to the SIS, represented by the message "search for market" in figure 2 and figure 3.

If the required market does not exist, the participant initialises and configures an new market following the initialisation protocol (see figure 1). Otherwise, the SIS sends the market's configuration parameters (channel, timeouts, etc).

Once the market parameters values at hand, participants have to register by sending a message to the market. The registration is valid if it has been done before a “stop-registration” time.

An auctioneer is then in charge of receiving buyers’ bids and sellers’ offers. Received bids are valid only if they succeed the buyers’ registration. Moreover, bids have to precede the “stop-bidding” time.

According to the single shot and the iterative protocols, this step is described as follows.

A) Single Shot

In the single shot protocol, the seller which seeks to sell a good, sends its offer to the market. This is represented by the arrow “send offer” in figure 2.

Buyers which would like to acquire this good, send bids to the market. This is represented by the “submit bid” message in figure 2.

In the single shot protocol, buyers can send bids only once.

B) Iterative

In the iterative protocol, sellers and buyers seek to find a market respectively to sell and buy multiple items, i.e. more than one type of good, such that one unit of each item is proposed. The items offered by sellers as well as their quantity do not evolve during the course of the auction.

In this protocol, buyers can send bundle bids on the objects traded at the auction. This is represented by the loop rectangle in figures 3. Moreover, bids may be withdrawn. This is represented by the option rectangle.

Bids are accepted if they are submitted before the “stop-bidding” timeout. This is represented as a condition in the loop rectangle in figure 3.

The auction iterates until an acceptable match is found, or untime a timeout stops the market.

3. Matching and Agreement Step

At the end of an auction, that is when the “bidding” timeout is over, the auctioneer matches the received bids with the seller’s offer. This is represented by the auctioneer internal action “Matching between the offer and the bids” in figure 2 and figure 3.

If the matching fails (in figure 2 and figure 3, see the upper path in the last alternative rectangle), the auctioneer informs the seller and the buyers.

Otherwise (see the lower path), the auctioneer notifies the winning buyer and price to both the seller and the buyers.

Then, the auctioneer creates the agreement manager role which is in charge of building contracts between winning buyers and sellers. The agreement procedure is represented by a continuation in figure 2 and figure 3.

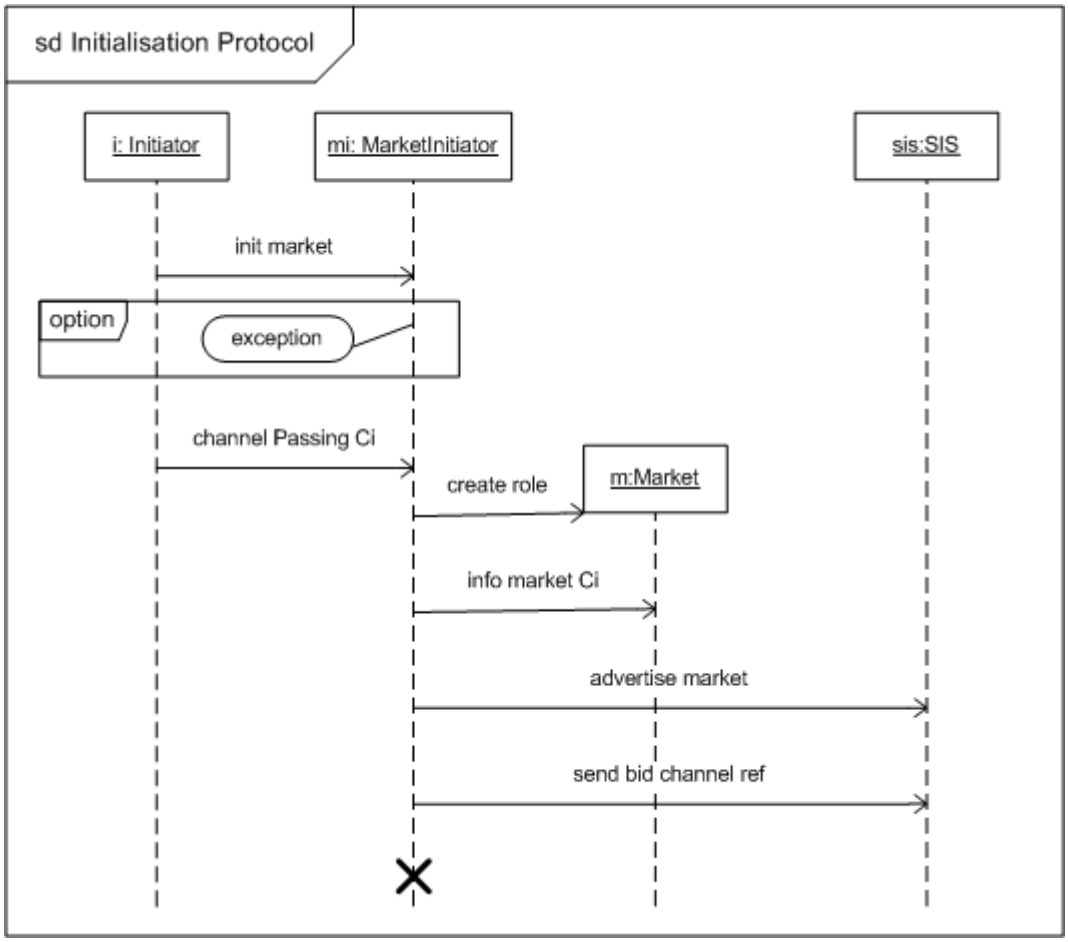

Figure 1: Initialisation Protocol

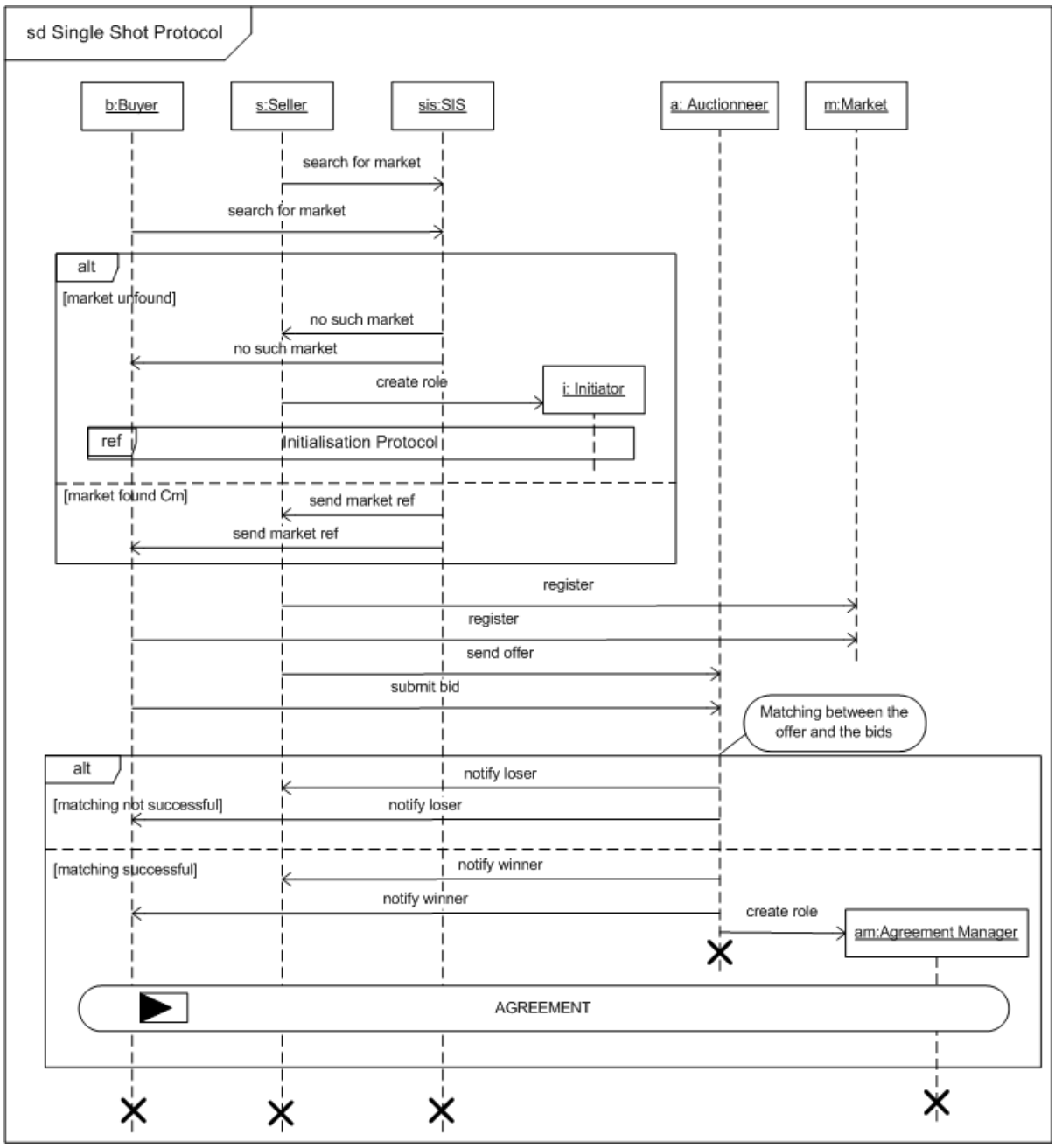

Figure 2: Single Shot Protocol

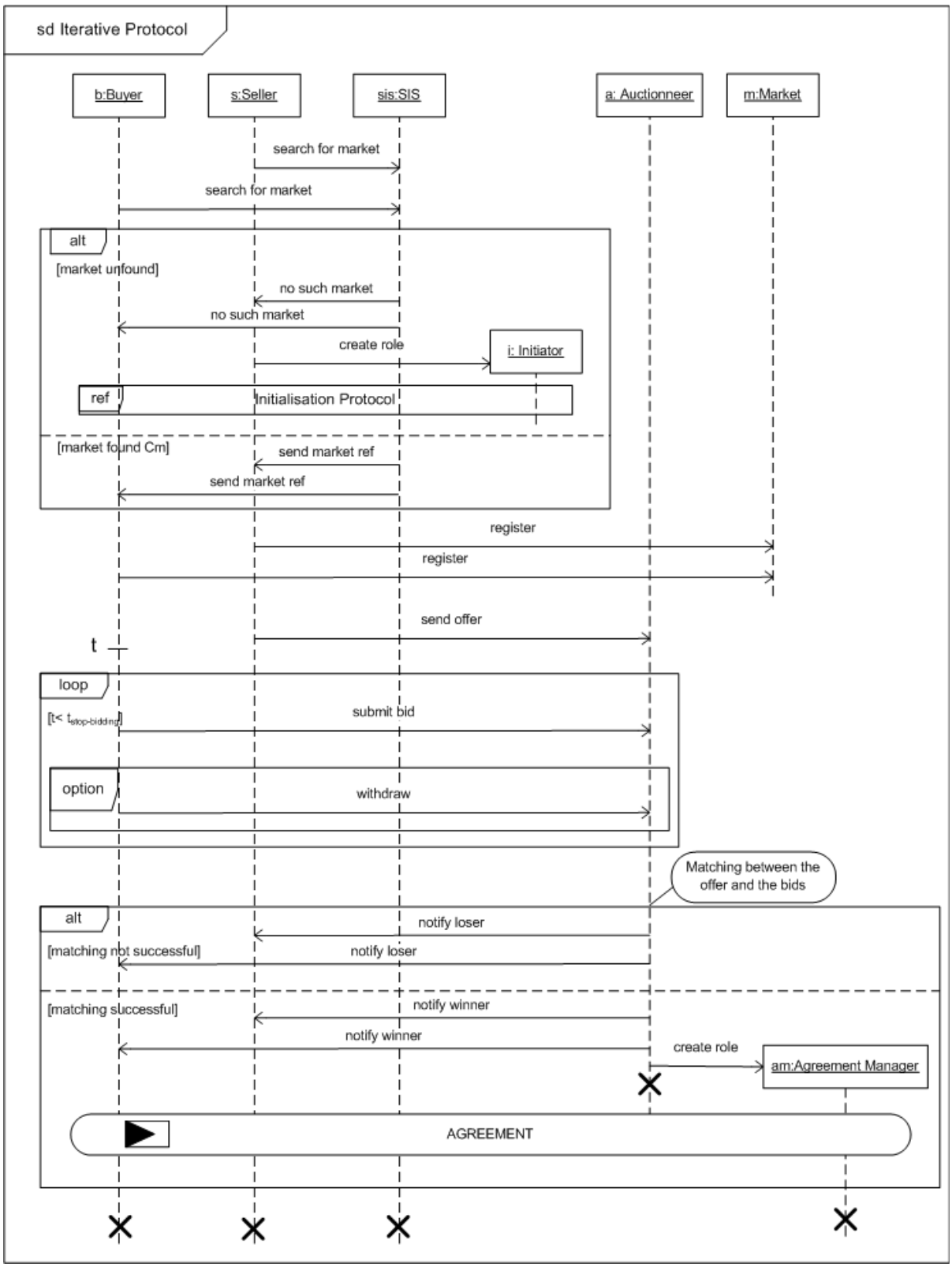

Figure 3: Iterative Protocol

3. The PI4SOA tool

The Pi4soa is W3C open source tool that offers a graphical editor to design WSCDL [1] specification i.e. a peer to peer web service composition. It is now at its forth release 1.6.3 and it works as an eclipse plug-in (there is no standalone tool).
The tool presents a set of features, we will point out in the following two features that deal directly with the project aims. Nevertheless a short description of the main features will be exposed.

WSCDL design feature: the tool offers the possibility to design a WSCDL service (a peer to peer collaboration between). The design steps flow the presentation of the WSCDL in deliverable 1.
The figure 4 presents a screenshot of the eclipse PI4SOA perspective. It is composed mainly by four windows:
the project navigator widows, the choreography widows, the properties window and the output console window. The choreography window is composed by three tabs **roles relationships**, **Base Types** and **Choreography Flow.**
Defining a choreography can be done by three steps:

Step1 : defining the roles and relationships

The first step of defining a choreography is to define or to identify the different roles. The roles as explained in deliverable 1 correspond to a behaviour of one or more partners represented by participant. A role may be defined by a set of independent behaviours represented in PI4SOA tools by behaviour. In this step we also define the relationships between roles. The relationships are oriented. A relationship between a role A and a role B means that A will interact with B and the target of the interaction (the services provider) will be B.

Step2 : defining data, channel and participant types (base type)

In this step we define the different types that will be needed within the choreography. The types correspond (as defined in deliverable 1) by data types and channel types (and what needed for channel i.e. locator and token locator). PI4SOA supports the importation of XML schemas that represent the types that will be used by the choreography information (e.g. important the XMLschema data types for XML primitive types). In this step the channel is defined and also affected to roles. Channel that can support other channel transfers must reference the transported channel type. Tokens are also defined (they are typed data) and affected to channels. The roles are then used to define participant types. Participant represents possible physical partners or organisations that will compose the choreography (in our case it can be used to identify the actors agent of the e-market)

Step 3 defining the collaboration protocol:

At this stage we define the flow of interaction between behaviours of roles. This collaboration protocol is defined by (i) a set of variables (information types, channel types, tokens, etc.) that composes the states of the whole choreography (ii) a set of choreography elements. One of the choreography must be declared as the main one and can be composed by other choreographies. (iii) one or more exceptions, finalize blocks both choreographies and blocks are defined by a set of activities. A choreography is defined by a flow of structured (sequence, while, choice and **workunit**) interactions. Interaction is between two behaviours of two roles. Interaction are allowed only between two related roles (related by a declared relationship). The interactions concern an operation (in the sense of WSCDL definition) and is composed by one or two exchanges. An exchange is an oriented message (sent and/or received). The exchange within an interaction depends on the operation model (request-response, request, response) which is

directed by the exchange model fixed by the used channel to make the interaction.

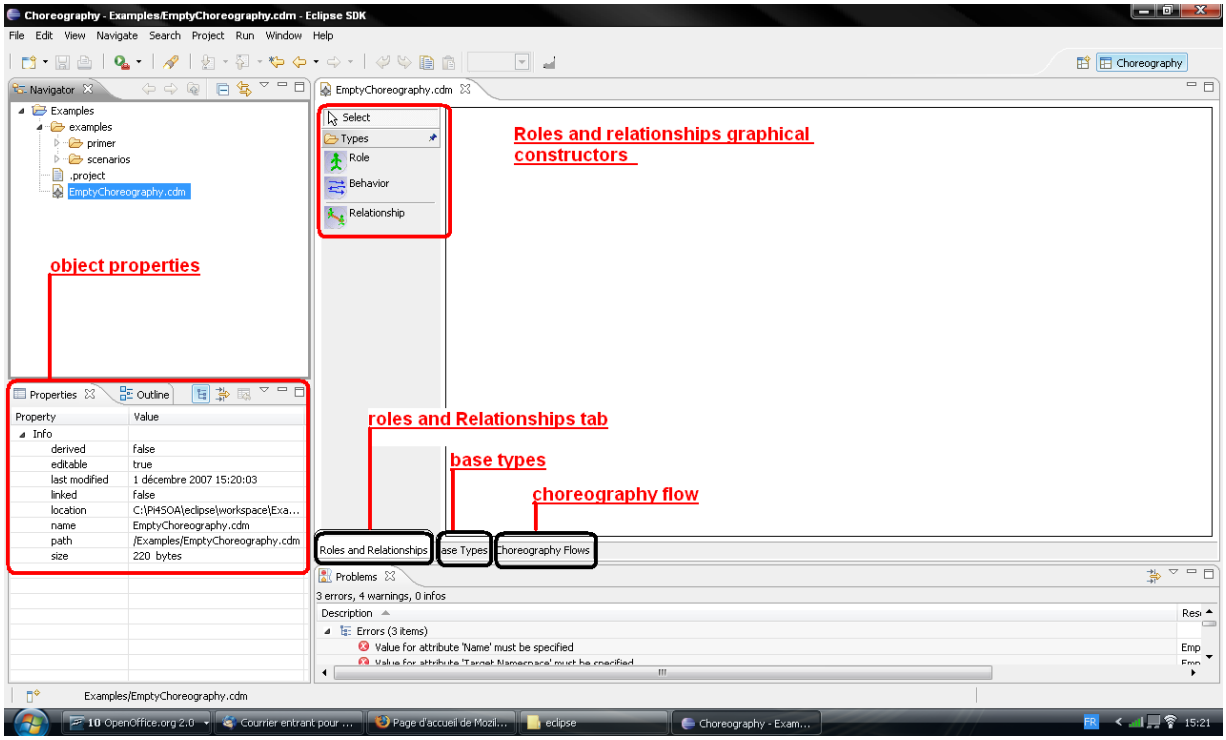

Figure 4: The PI4SOA eclipse plug-in perspective

In the flowing we describe the single shot mechanism by illustrating the different steps. The single shot specification using WSCDL.

4. Single Shot specication using PI4SOA tool

1. Roles and relationships:

With the single shot we identify four roles :

- **The initiator (Initiator_R[1])** is the market instance creator. It can be a seller, a buyer or a third party. Its role is defined by only one behaviour initiating market an then be contacted by the market to finalize the negotiation when a winner is defined.
- **The Market (Market_R)** is a partner who offers an auctioneering service. Each instance of this role corresponds to a market instance. It is contacted by the initiator to initiate an instance and then contacted by bidders for trading. It is defined by one behaviour that can be summarized by four steps (they will be largely commented when we describe the flow) : instantiation, registration, biding and then concluding by informing the bider and the initiator of the negotiation issues.
- **The bider (Bider_R):** the bider role can be either a seller or a buyer. In the mechanism, the type of market selling or market does not have any importance on the mechanism behaviour. That is why we aggregate buyer and seller in bider. The bider has only one behaviour that can be summarized in two steps: finding a market instance in the SIS and then bidding in that market.
- **The SIS (SIS_R) :** the SIS role plays the role of intermediate between bidders and markets instances. It is composed by two behaviours: the *SIS_Reg_B* and the *SIS_Get_B*. Why we need two behaviours for the SIS? There is two main reasons to define a role with more than one behaviour: First because it corresponds to two completely independent applications offered by the role that we call local independence (i.e. for the role it can offer two independent services) consulting market do not depends operationally on market registration. The first condition is necessary to split the role in two behaviours within a choreography but not sufficient. In addition to the local independency we need to proof that there is no other behaviour of an other role that needs to interact with the two behaviours within the same choreography. In the case of the SIS, the registration service and the discover service do not share any related behaviours ; The registration is for the market while the discover is for the bider. That is why we separate these two aspects of the SIS in two behaviours.

Roles behaviours are represented in the figure 5 screenshots of the PI4SOA tools.

[1] The name of the role in the single shot WSCDL proposed specification

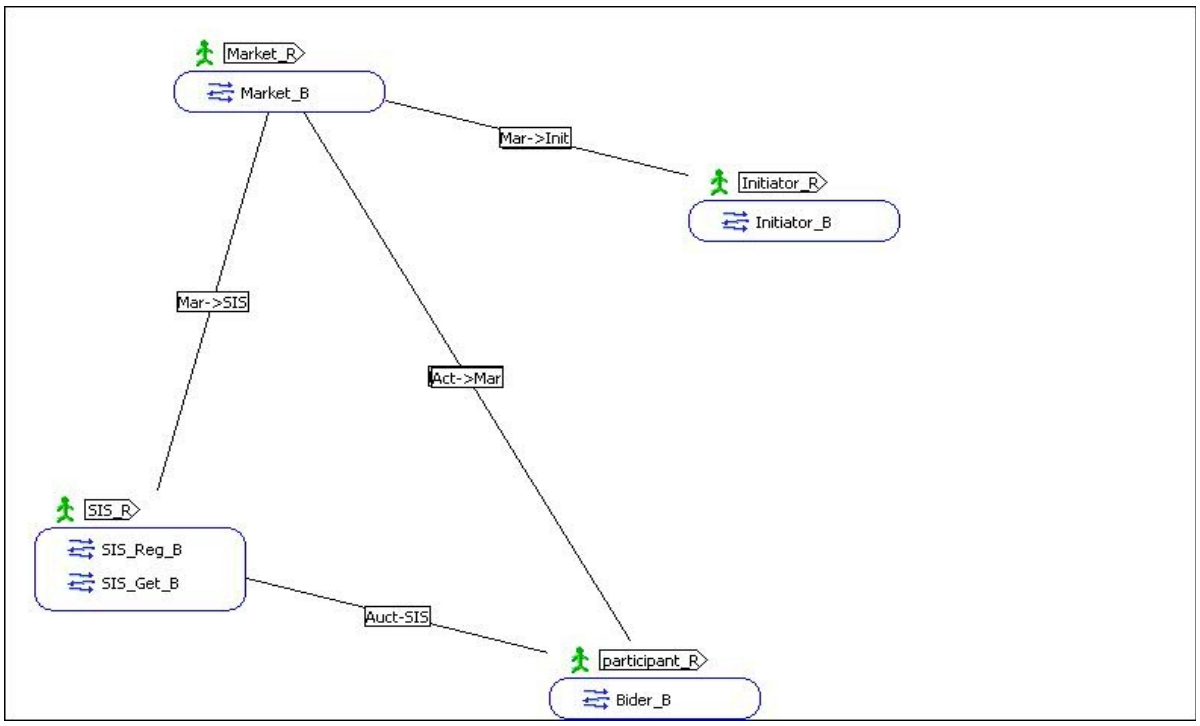

Figure 5: Roles, behaviours and relationships that compose the single shot

2. Defining the base types :

The base types deals with all types needed for the choreography. The tool separates them in classes. Here we emphasizes three classes of types:

A)Information types

They are application types that correspond to exchanged messages and local state types. Types can be defined by reference to an XML schema that can be imported within the **name space** element. In the case of the single shot, we have identified the set of types. Those types are essentially for message exchange types like **BidType** which corresponds to the bid message type. Message types use only reference to primitive XML types. This restriction is due to a bug in the tool that do not support correctly XML schema importation (warnings are generated). See in the archives the html description of the protocol for a complete presentation of the information types

B)Participant types

Participant types allow to define a set of participant types by grouping roles. Participant types can be used when defining the choreography to declare more that one instance of a set of grouped roles (i.e. participant). Normally a participant type refers to a possible organization that participate in the choreography realization.

C)Channel types and locators

The last part of the base type is the channel type definition. In this part we define the needed channel types for the choreography. While interaction happens always between two roles and on a given channel with a choreography it must belong to a specific behaviour. Each channel type makes a reference to a token. So before defining channel type one must define the set of tokens. Tokens make reference to a declared information types. To identify a specific part of the information type that will correspond to the token we declare a token locator for each token (using an Xpath expression). Once Channels types are declared we can specify the

passing channel. A channel that can be used to transport data of an other channel must make reference to the transported channel type within its definition. The figure 6 illustrates the point using the market channel: the market defines a channel that will be used by the Bider to contact him. The bider may send it and contact channel to be informed of the negotiation result. The figure shows how the **B2M** market channel is declared as channel that can support transfer of **BiderC** data .

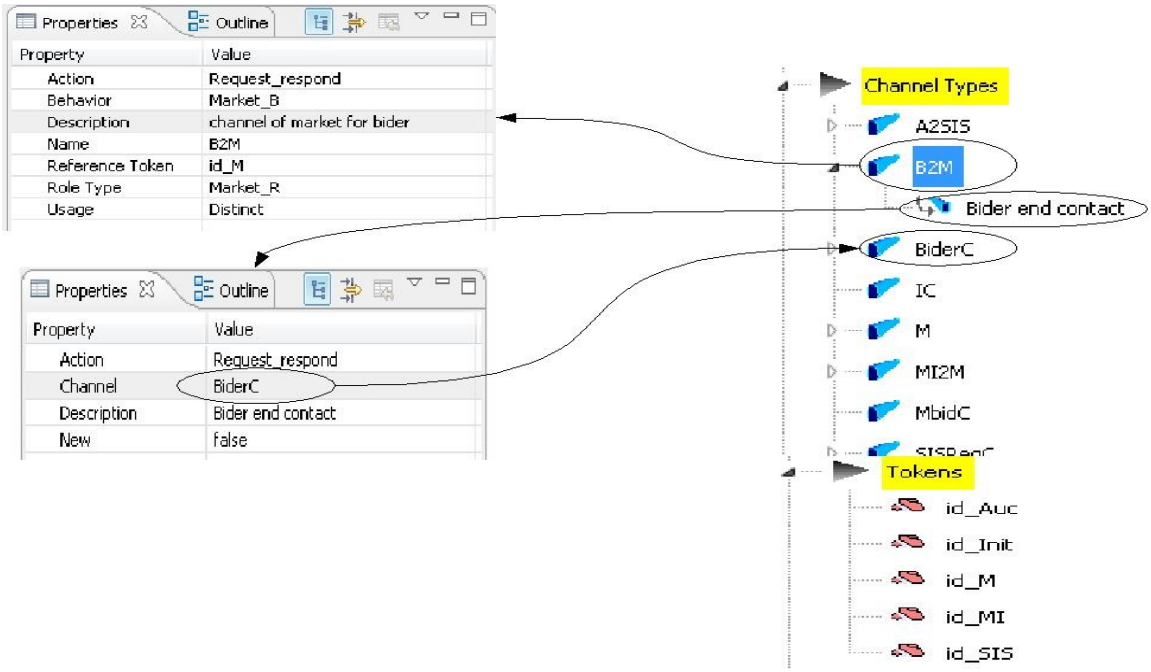

Figure 6: Channel types declaration and passing channel dependency

3. Defining the choreography Flow:

Once the choreography structure (roles-relationships) and needed types are defined we can define the global protocol that defines the different roles interaction (in our case the Single Shot protocol).

The choreography flow is the specification of all allowed interactions (see figure 8) between the different behaviours. We must point out that the choreography flow do not correspond to a possible scenario of interaction but all the allowed scenarios of interactions. Thus, the choreography flow is a set of controls structure operators that restrict a set of basic enclosed interactions. When defining the choreography the designer also define the interface of each behaviour of each role. The interface is defined (see next when we compile WSCDL to BPEL) by the offered operation, the interaction interface, the protocol of accepting partner invocation and consuming partner operations. So we must be careful that by defining the allowed interactions we are defining the services types.

As claimed in the UML description we can distinguish two separate parts of the whole interaction: the initialisation and the registration-biding. This separation appears when defining the whole choreography. It is to the SIS two behaviours ; one for registration and the other for market discovery. So within the global view of the choreography market registration and market discovery interleaves (the SIS can hold may market instances). The whole choreography is an interleaving (parallel composition of the two parts: initialization and market discoverer and play). The figure 7 shows the root structure of the choreography.

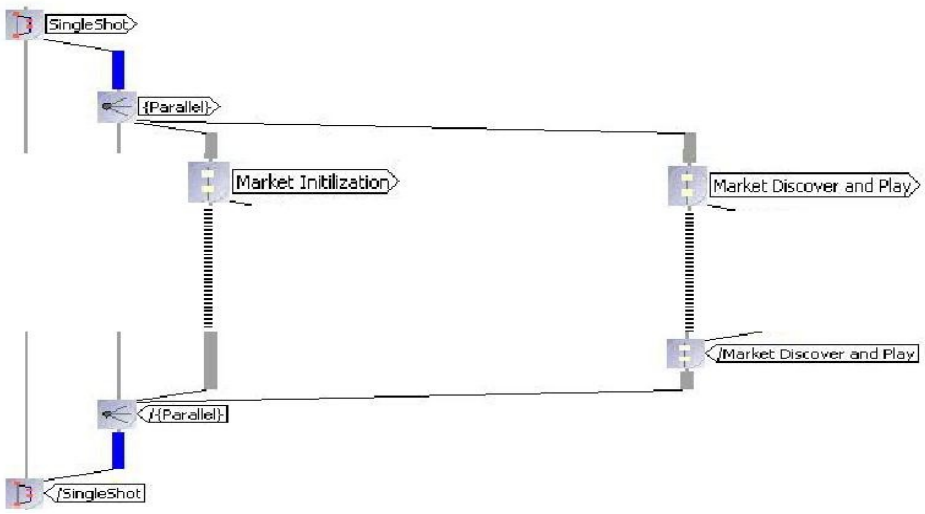

Figure 7: the single Shot choreography is composed by two parallel flow one for initialization and the second for auction

A)The initialisation step

The initialization of market involves the Initiator, the Market and the Registration behaviour (e.g. the SIS role). Their interaction is linear 'see the figure 9):

1. Initiator-->Market : (**ν *varCofMInit [M]***) [2](marketcreatOP(marketInfo)).
2. Initiator-->Market : ***varCofMInit*** (getMarketOwnerRefOP(***InitiatorCannelValue***))[3].
3. Market-->SIS_Reg: (**ν *varRegSIS [SISRegC]***) registerOP(marketInfo).
4. Market-->SIS_Reg: (***varRegSIS***) (getBibsessionRefOP(***varCofMInit***)).

First the initiator initiates an interaction with the market so a fresh instance of market service is generated.
Then the initiator sends its reference (its instance channel to the market instance). The market instance will contact the SIS_Reg (registration behaviour) and then publish the market information. This interaction creates a new instance of the SIS_Reg. That instance will receives the market instance reference.

[2] *A--->B : (ν s[t]) (opName(arguments))* : means that *A* invokes the operation *OpName* of the behaviour *B* and this using the channel variable value *s* of type *t*. That invocation creates a new instance of B and that instance channel is s plus the value of the locators during the exchange.

[3] A--->B: s(OpName(arguments)): means that A invokes the operation OpName using the channel variable s

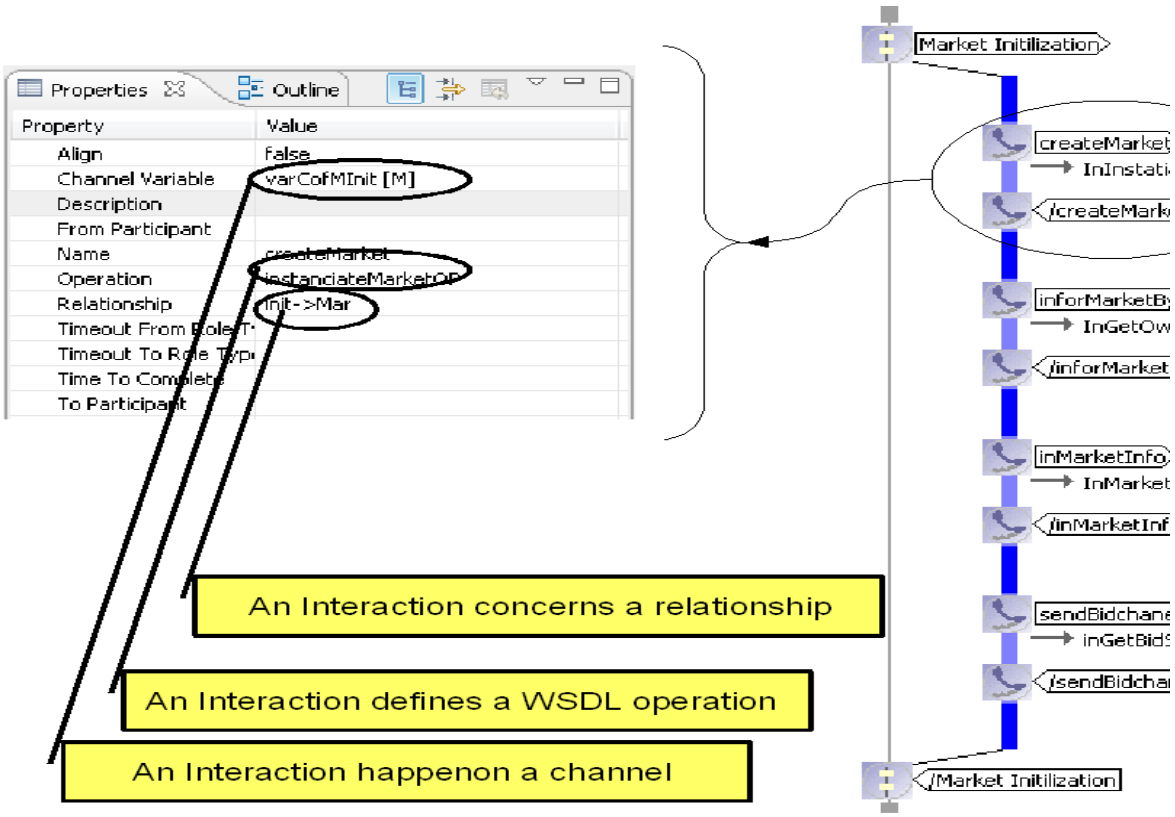

Figure 8: the market creation interaction detail

B)The market discover and auction step

This part is initiated by the bider who checks for a market. First he requests the SIS_Get services for a market, two cases are possible: either the market exists and then the choreography continues or an exception is raised (***NotFoundMarketExp***) end the choreography stops here. In case of a possible market, the response of the request is the market instance reference. When getting the market bid channel reference, the Bider instance can then interact within the market. While a set of possible bidders can interact within the same market instance, we can not sequence the registration the biding. That is why we consider that the registration and the biding can happen in parallel. The side effect of such design may lead to a bider behaviour that can interleave the biding and the registration. This is so permissive while we must restrict the bider that the bidder registration must happen before biding. In order to realise such order constraint we use for each biding session (that concerns a bider) a sub-thread of the market one. This sub-thread reference (here channels) are generated by the market instance for each registration. This means that the bider must get the biding session channel in order to perform it. An other important aspect with this part of the choreography is time constraints. The registration is possible only when registration date is reached, the biding is possible only when the beginning if the bidding session date is reached. And finally the result of the market is only sent when the bidding session is closed. These time constraints are designed using **workunit** constructors (see deliverable 1). For the three cases, registration, biding and result send back, interactions are placed in **workunits** guarded by "***cdl:hasedeadlinepassed(date)***. The dates are supposed to be communicated by the initiator[4] (the XML data types are in the rachives delivered with this document) when initiating

[4]This does not appear in our design because of the difficulties we had to import complex types schemas.

a market. They are global variables. **workunit** are blocked until the respective dates are reached.
We show in figure 9 an annotation of the single shot choreography Flow..

C) Exception block *vs* applicative rules

The tool supports also the reaction of the choreography when exceptions are raised. We can distinguish two kind of exceptions: the first type when the condition of the exception raise is not global (local to one partner). For example, in the choreography when the bider asks the SIS for a market and no instance fits its requirement an exception is raised and the associated exception block is activated (the exception block do nothing but terminate). This provokes the end of the choreography. The other type of exception is when the condition is globally observable. For example, when the bider sends a bid value less then the market value. When condition on observable (or exchanged) data can be expressed, this belongs more to the choreography rules than exception. This is the reason why the bad bid is handled by a conditional behaviour rather than by an exception. We will see how this choice will lead to well-detailed role definition.

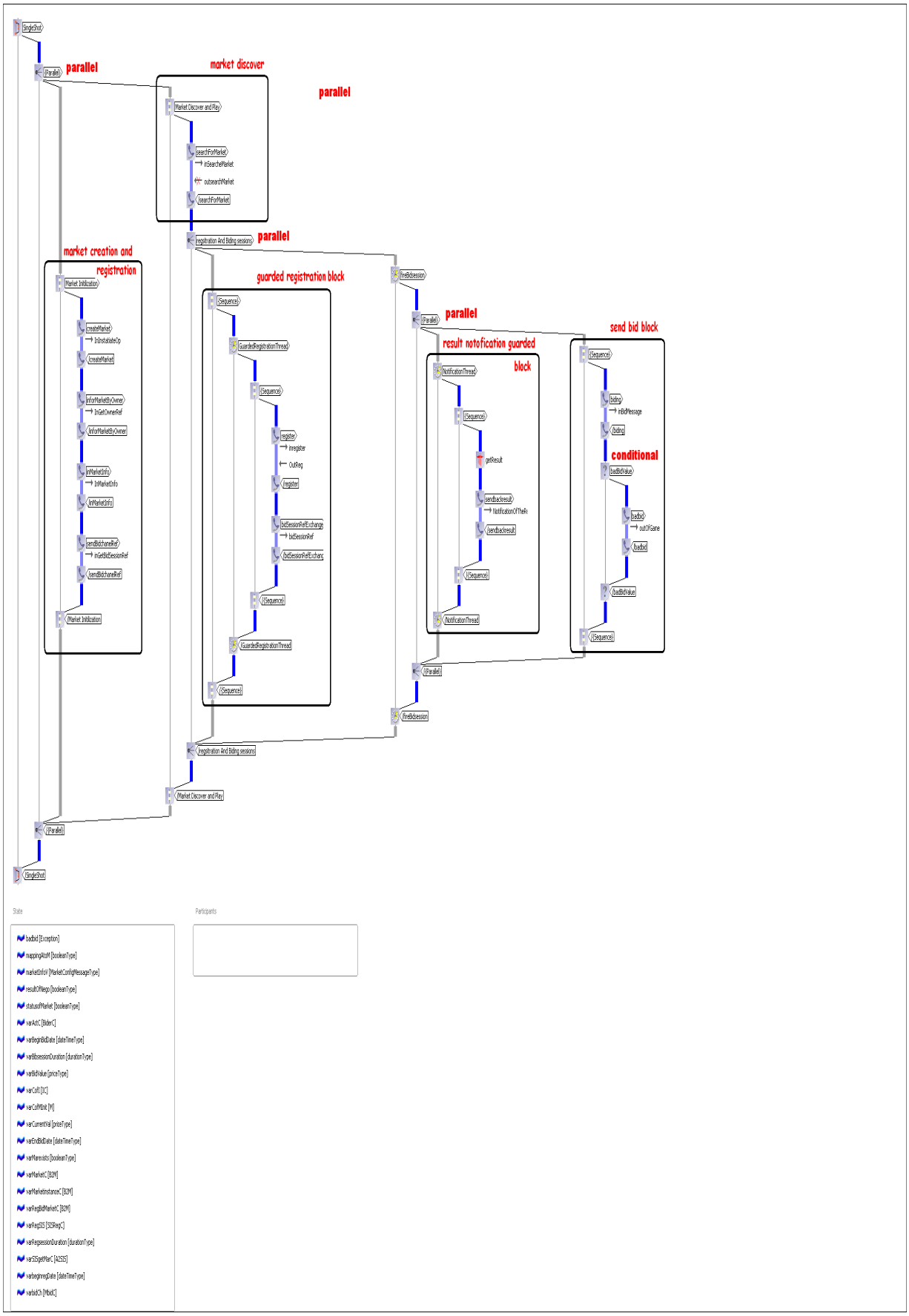

Figure 9: The Single Shot Choreography Flow

5. From WSCDL to activities state model

The resulted file produced by the tool is a ***.cdm*** file (a specific XML file). Once the design finished, we can export such file to a set of other languages: UML, HTML description or to the WSCDL specification.

WS-engineers Is an eclipse plug-in that extends the LTSA (Labelled Transition Systems Analyser), tools to Web services languages such as WSCDL and BPEL. This plug-in takes as input a WSCDL specification and transforms it to an equivalent FSP[4] specification (see[5] for translation details). FSA is a process algebra used as specification language by the tool (see figure 10 for LTSA-WS-engineers perspective). We note here that such translation ignores data and time constraints. Only interactions are modelled..

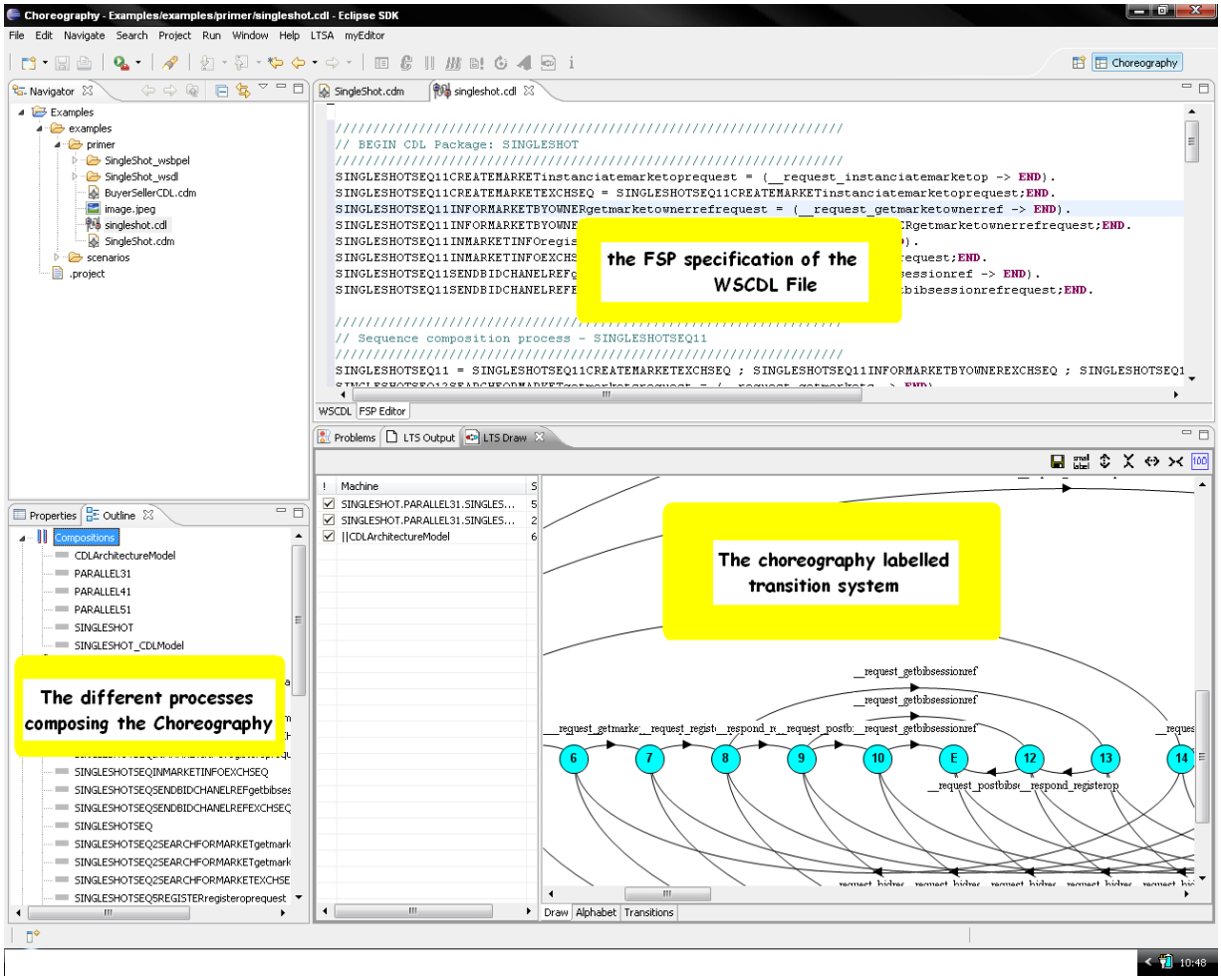

Figure 10: The LTSA We-engineers tools perspective : from WSCDL to FSP

Using the resulted FSP specification we can use the LTSA tool verification and simulation functionalities. This tool allows the checking of deadlock freeness and also for the progress properties. In our case (choreography) deadlock checking is not relevant because deadlock can not appear in global view specification so by construction the FSA specification also will be deadlock free. In contrast, the tool offers the possibilities to check LTL[4] formula specified by the designer this can be used the check applicative properties on the global interaction (for example be sure that if the registration happens the bider will always receive the result, even it was eliminated after a bad bid value.). The simulation is an other functionality that can be useful for a designer not familiar with model checking. The simulation offers a step by step

execution of the choreography interaction. At each step the designer is guided by firing one of the current possible state interaction. The figure 11 shows the simulator of LTA applied on the single shot. The trace shows all possible interactions until the biding. The user have then the choice to simulate a bad bib or a good bid.

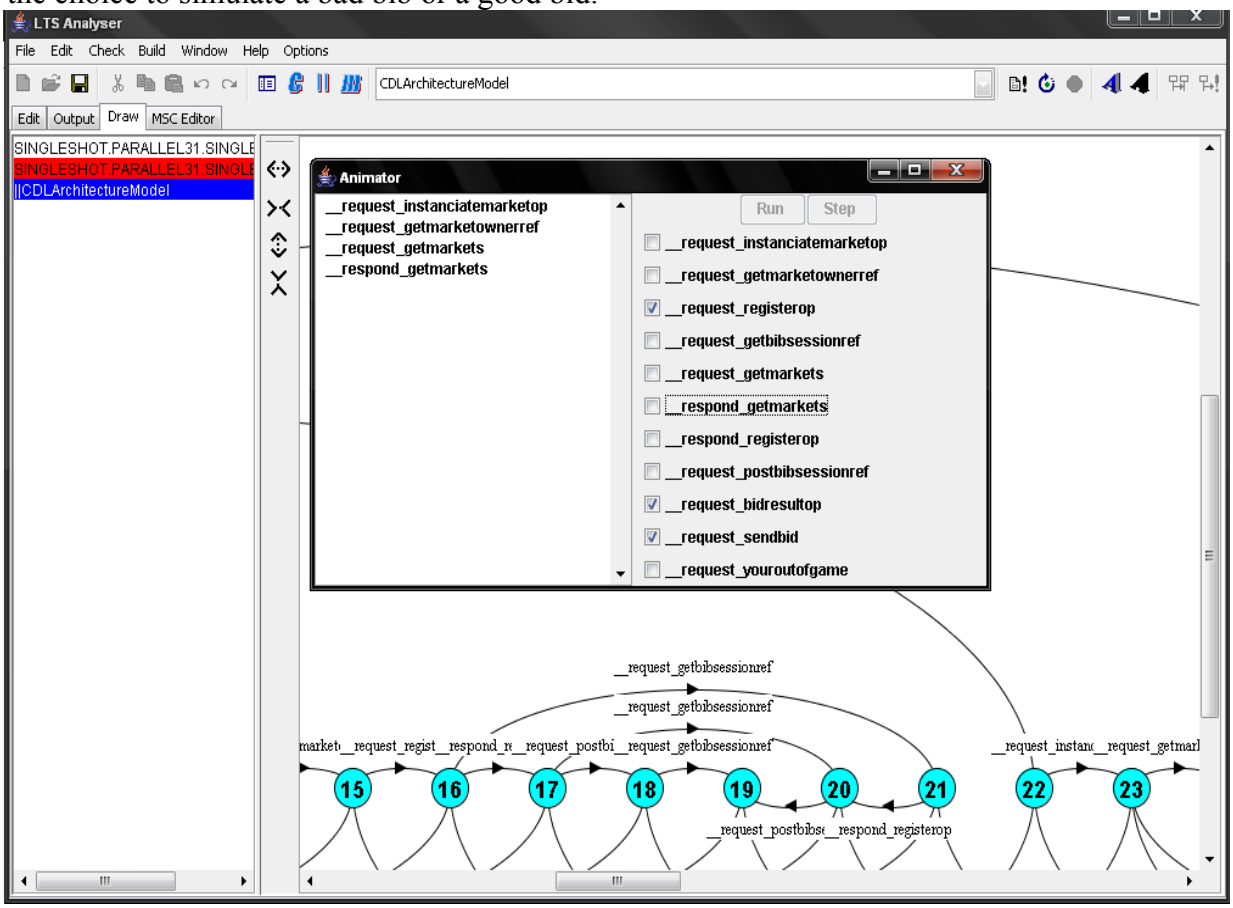

Figure 11: The LTSA WS-engineers simulator.

6. From WSCDL to BPEL

We pointed out in deliverable 1 the theoretical foundations behind global calculus that leads to EPP (End Point Projection). The EPP consists in deriving from the global specification (the choreography) each participant (here the roles) specifications in a sound and complete way (their composition behaves exactly according to the global one). We have explained also the importance of such generation for the proposed functional model ; "the participant behaviours generation allows the market actors to check their conformance to a role". An other advantage (also pointed out in deliverable 1) is that the EPP can be used for code generation (or more precisely a starting point for the implementation). The PI4SAO implements a variant of the EPP method. It uses WSDL as target language for role specification joined to BPEL (Abstract BPEL). Other target languages are also prosed like Java for example.
The only drawback of the projection functionalities of PI4SOA is that project the conditional behaviours on all the involved partner even the condition concerns one role data. *Why such choice*?

- First, we must remember that variables are situated which means that they belong to one or more roles.
- Second, The condition expressions are boolean Xpath expressions (i.e. strings). To situate a conditional behaviour, the tool must extract (by parsing the expression) the involved variables.
- Last, when a condition is involving more than one variable and those variables to a disjoint set of roles, the project leads to a redefinition of local conditions (which is in the most cases impossible to realize without adding explicit state communication).

Those reasons make a more specific projection of conditional behaviours hard to realize, that is why the conditional behaviours is projected on all the roles involved in the condition block.

A) How to generate

To generate participant specification, we must first enable the generation functionalities (right click on the choreography file --> properties). We can then choose to enable the target specification and also the version for example for WSDL. We have the choice between version 1 or version 2. We can also specify if the WSDL file will be just the interface or we generate also the biding details.

B) Generation rules

WSDL generation : PI4SOA allow the generation of the WSDL interface for each role. As we explained before each role divided in more that one behaviour this mean that we will generate a WSDL for each behaviour. The WSDL file defines the set of operation (elementary services offered by the behaviours) this is constructed by analysing the different interaction within a behaviours as target and then extract the union. Each operation have the same name as the operation is the WSCDL specification and the message types are extracted from the enclosed exchange information types. The WSCDL file of all the behaviours are in the archives associated to this deliverable figure X represent the WSDL File of the SIS registration service. It offer two operations ***registerOp*** for registration and ***getbidsessionRef*** that allow the market to send its channel information.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://localh
    <portType name="SIS_Reg_WSDL">
        <operation name="registerOP">
            <input message="xsd:string" name="InMarketInfo"/>
        </operation>
        <operation name="getBibsessionRef">
            <input message="xsd:string" name="inGetBidSessionRef"/>
        </operation>
    </portType>
    <binding name="SIS_Reg_BBinding" type="tns:SIS_Reg_WSDL">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="registerOP">
            <soap:operation soapAction="http://localhost:8080/chor/SingleShot/registerOP"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/chor/Single
            </input>
        </operation>
        <operation name="getBibsessionRef">
            <soap:operation soapAction="http://localhost:8080/chor/SingleShot/getBibsessionRef"/>
            <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/chor/Single
            </input>
        </operation>
    </binding>
    <service name="SIS_Reg_WSDLService">
        <port binding="tns:SIS_Reg_BBinding" name="SIS_Reg_WSDLPort">
            <soap:address location="pi4soa: User defined definitions"/>
        </port>
    </service>
</definitions>
```

Figure 12: The generated WSDL file of SIS registration behaviours

The BPEL generation: the BPEL generation is more complicated than the WSDL one. The BPEL normally will correspond to the local collaboration of a given behaviours. The generation of BPEL for a given behaviour can be summarized in three steps :
partner declaration: partner of a behaviours here are all the choreography behaviours wich are related with an outgoing relationship of the considered one. In other words the are all the behaviours that provides operation to the behaviours.

- **Projection of interactions:** in this step we consider all the interactions that the behaviours participate either as consumer or provider. The local projection of an interaction depends of the role played by the behaviours, consumer or a provider. In first case the interaction is represented locally by an invoke activities and the second case the interaction is represented by a receive is the request exchange element appear and reply if the response exchange appear in the interaction.
- **Projection of the control flow:** The control flow on the previous transformation (interaction to invokes or receive/reply) is derived by project the chronography control flow on only the behaviours concerned interaction. Then a translation of choreography specific constructor to BPEL one that perseve the same operational semantiqc (the constraint order). For example the workunit is transformed to a **scope**[5] : A **workunit** uses the **hasdedlinepassed()** condition is transformed to a **scope** with a BPEL **wait** activity as the first one and then the behaviours of the **workunit** block. On the other hand if the condition is ***hasdurationpassed*()** the **scope** is defined with a ***time-out*** (using the argument as duration) and the **workunit** behaviour block will be the ***time-out*** behaviour.

The figure X represents the BPEL file(for in xml tree format for clarity) of the bider BPEL. we point out here the exception behaviours when a market was'nt found. The rest of different FPEL file of the different behaviours are in the archives.

[5] This is a critical choice and cause a lot of bugs

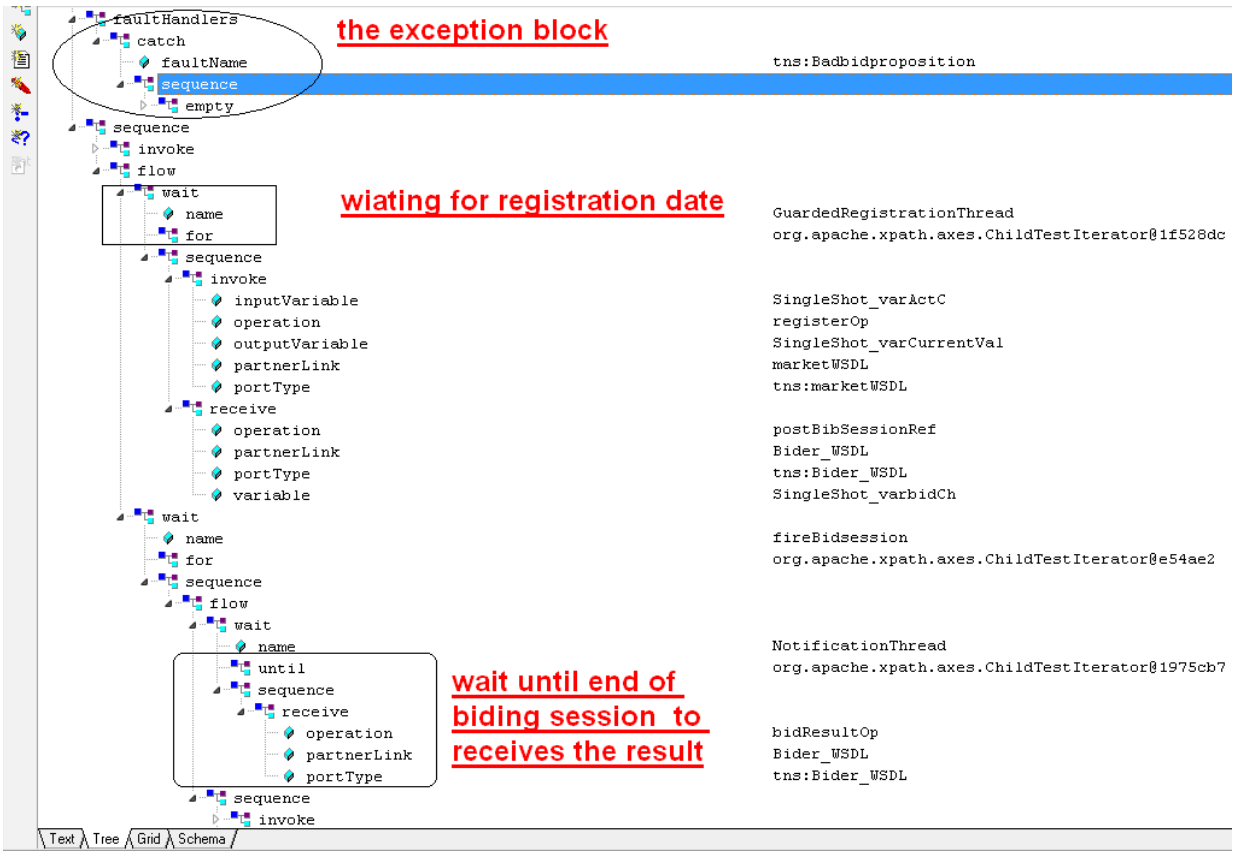

Figure 13: the XML structure of the Bider BPEL behaviour

2. From BPEL to models

Many works aims to give a formal model to BPEL and that for different reasons the first one is that XML format is very fastidious to use we need a intermediary light model to specify the service behaviour before implementing it; The other reason is to check properties of the BPEL services so we need to transform it to model checkable format (For a more detailed description of related work on transforming BPEL see the project proposition). The most existent tools model only behavioural aspect of BPEL using petri-nets, state charts or automata etc. Part of them support time constraint. In this class of tools the data are totally ignored, in the next deliverable when we address compatibility we will sketch the needed feature to compare actors behaviour to abstract roles. the LTSA plug-in WS-engineer presented also support translation of BPEL process to FSP specification then similar functionalities can be done (verification and simulation). In addition we consider here a second tool (a prototype called WSMOD [3]) provided by IBISC partner that transform abstract BPEL to timed automata this tools is working on Abstract BPEL (no data and no conditions) is now being adapted to support additional BPEL feature to fit to PI4SOA generation format. for example the wait is generated by using BPEL 2.0 format while our tool work on BPEL 1.1 . The figure 14 presented the the transition system associated to the bider role.

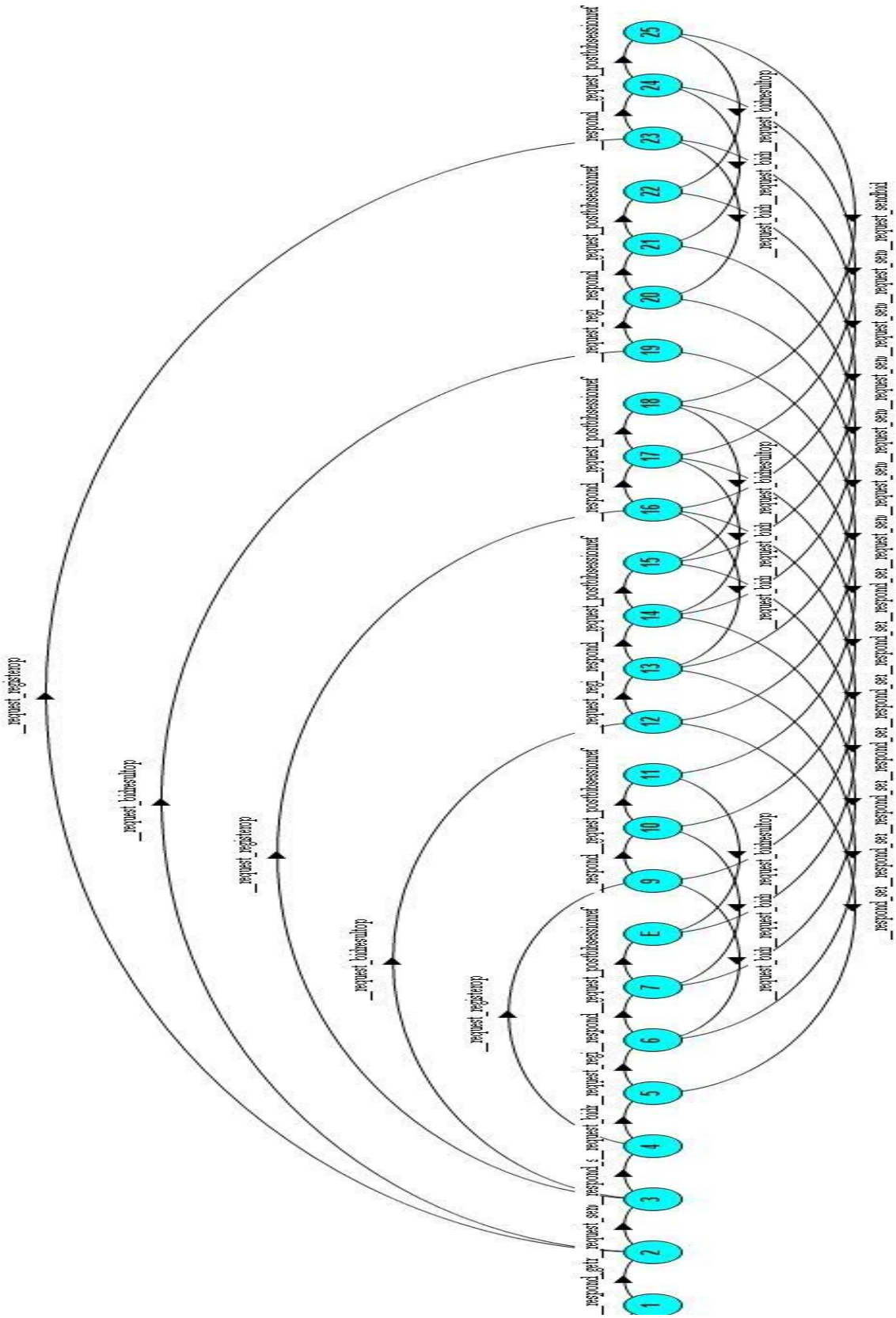

Figure 14: The labelled transition system of the Bider behaviour

7. Conclusion

In this deliverable we illustrated our methods (presented in deliverable 1) by using the Single shot protocol. We also presented a set of tools to design negation mechanism a choreography of Web services, PI4SOA the tools is an open source and offer additional functionalities like roles BPEL specification generation. We presents also an other plug-in that called LTSA WS-engineers that transform a WSCDL specification and BPEL specification to a process algebra called FSP. The transformation concerns only communication action, both time and data are ignored. We are extending a tool provided by out IBISC partner to support time and data oriented model. conditional

The archives

This deliverable is send with an archive that contain the following document:

- singleshot.cmd : is the PI4SOA specification of the single shot protocol
- iterative.cmd : is the PI4SOA specification of the iterative protocol
- singleshot.cdl : is the WSCDL specification of the single shot
- iterative.cdl : is the WSCDL specification of the interactive
- singleshot_wsdl : directory contains the WSDL files generated for the single shot
- singleshot_wsbpel : directory contains the BPEL files generated for the single shot
- iterative_wsdl : directory contains the WSDL files generated for the iterative
- iterative_wsbpel : directory contains the BPEL files generated for the iterative
- singleshot.html : is HTML description of the specification
- iterative.html : is HTML description of the specification
- singleshot.xmi : correspond to an UML export of single shot specification
- iterative.xmi : correspond to an UML export of iterative specification

The tools download addresses and bibliography

[1] PI4SOA : the tool can be download here http://pi4soa.sourceforge.net/ with the eclipse 3.3.X

[2] LTSA WS-engineers : is a plug-in developed by the imperial college of London. Its open source and can be downloaded here http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/

[3] WSMod is works with the BPEL 1.0 is an ibisc provided tool and can be downloaded here www.ibisc.fr/~melliti

[4] Peter Michael Sewell PHD thesis "*The Algebra Of Finite Ftate Processes*" October 1995.

[5] Howard Foster, Sebastian Uchitel, Jeff Magee and Jeff Kramer " *Model-Based Analysis of Obligations in Web Service Choreography* " IAICT-ICIW 2006 page 149.