O.C.E.A.N. Project
Negotiation Component Specification

Component's Role in O.C.E.A.N. System Architecture

[This section briefly describes the context of the component; how it relates to the system architecture of the OCEAN system as a whole.]

The Negotiation component is present in each OCEAN node. It primarily makes use of the Auction component, although it might also receive helpful information from the Trader component. Like most node components, it uses the security/communications layer for communications to other nodes on the network. (See figure below.) Like all node components, it supports configuration & maintenance interface. Its network communications are primarily with the Negotiation components of other nodes. The Negotiation components role, as its name implies, is to negotiate the terms of the deal with it's peer nodes before Job Migration takes place. The Negotiation Component might employ a Protocol to decide which node it will do business with based on the information it receives from the auction component.

Functional Needs

[This is a more formal statement of who the component's "customers" are, and what needs of the customer the component is intended to help fulfill.]

Customers.
The Negotiation component customers are as follows:

1. The Job Migration component.
2. The node operator.
3. The Negotiation Components of the other nodes.
4. Mobile tasks and their users.

1. Needs of the Job Migration service.

- **a)** The job migration component needs to figure out fast which node to migrate the job to.
- **b)** A common naming policy should be employed so that the job migration component knows where to migrate the job.
- **c)** Any other information needed regarding the deal should be given to the Job migration component as and when required.

2. Needs of the node operator.

- **a)** The operator needs the Negotiation component not to consume an unreasonably large portion of the local resources controlled by the operator: CPU, memory, disk, network…
- **b)** The operator needs to be able to ask the Negotiation Component to do business with either Best Service nodes or cheap service nodes etc.
- **c)** The operator needs to be able to manually choose which node to do business with.
- **d)** There should also be a default node to do business with (Like the Best Service node etc) so that the node operator need not be involved everytime.
- **e)** The node operator must be able to specify the speed of the negotiation process, ie he may specify that negotiation be completed quickly or that more importance be given to the negotiation process depending on how time consuming the job is or how important he feels the negotiation process to be in the big picture.
- **f)** For ease of setup, the Negotiation component should not require the node operator to do anything special in order for it to start working. (It should just work, in the out-of-box installation.)

3. Needs of Negotiation Components at other nodes.

- **a)** The Negotiation Component should be capable of redefining the requirements if the auction component doesn't give an exact match.
- **b)** The negotiation should take place using standard XML documents(somewhat in the manner of the Auction process).

c) Multiple negotiations can be taking place at the same time in different nodes.
d) The negotiation component should let the winning node know that it has been chosen.
e) The negotiation component should let the losing nodes know that they have lost.

Functional Requirements

[This section takes the broad needs of the customers, and translates them into more detailed and specific requirements on the component.]

Below are the requirements on the Negotiation Component. After each one, we indicate which need (or other requirement) that specific requirement addresses.

1) The Negotiation component should provide the Job Migration service with the information about which node to migrate the job to. (Needs 1a, 1b, and 1c.)
   a) The node address with which the business is being done should be according to the naming conventions (Need 1b.)
2) The Negotiation component should give the node operator power to manually choose the node to do business with. (Requirement 2c.)
   a) Default actions should be provided in case a perfect match is not provided by the auction component. Ie, Should we try again with a different requirement specification or should we choose the peer with the nearest match. (2c)
   b) The matches provided by the auction component should be categorized as Best Value, Best Service etc.(Requirement 2b, 2c)
   c) In case a peer goes down after the job migration has started the negotiation component should ask the user what it should do next. (Need 2d)
   d) In case a peer goes down after the job migration has started the negotiation component should choose the next best option. (Need 2d)
   e) For speed the Best Service nodes and Best Value nodes should be maintained in arrays in memory. (Needs 2b and 2c)
   f) The Best service nodes and Best value nodes should be stored in descending order of preference in the arrays so that the next best node is easily acquired. (Needs 2a 2b and 2d)

3) The Negotiation component should talk with other Negotiation components through standardized XML documents. (Need 3c)
   a) The Negotiation component has to send out messages to the other nodes depending on which node has been chosen. (Needs 3c and 3d)
   b) The Negotiation component needs to ask the auction component to redefine the requirements in case an exact match for the requirements is not found.(Need 3a).

Interface Specification

[In this section, we describe in detail what the external interface to the component is going to look like. This section may not be as formal as an actual Java file containing class & interface declarations, but it should be similar in semantic content.]

The following are the classes and methods used in the Negotiation Component:

Class Buyer
This class encapsulates the buyer part of the Negotiation component. This class contains the following methods to get information about different offers.

Methods in Class Buyer:
void setOptions ()

This is the first method that is called to set up all the options which have to be set up by the node operator (like which offer to choose, best valued or best service, etc) for user intervention free operation.

Offer getBestValuedOffer(OfferList)

This method returns the best-valued Offer from the OfferList class. It also removes the Offer from the OfferList so that it won't be around the next time this function is called.

Offer getBestServiceOffer(OfferList)

This method returns the best-service Offer from the OfferList class. It also removes the Offer from the OfferList so that it won't be around the next time this function is called.

Offer chooseOffer(OfferList)

This method is used by the node operator to give him the control to choose the offer. This method first asks the node operator how many Offers he wants to choose from and depending on that number the method repeatedly calls getBestValuedOffer() and getBestServiceOffer() and shows the Offers to him.

Boolean isContractSigned()

This method is used to check if the contract has been signed by the seller or not.

Void completeNegotiation()

This is the final method called. This waits for the signed contract and then calls the corresponding method in the TSM component to start the job migration.

Class Seller

This class encapsulates the Seller part of the Negotiation component. This class contains the following methods to get information about the contracts it receives.

Boolean isOfferAccepted(int offerId)

Tells the seller if the offer has been accepted or not so that it can call the remaining methods accordingly.

Boolean checkContract(Offer,Contract)

This is called only if isOfferAccepted() is true. This method compares the contract received with the offer it made and checks to see if everything is in order or not.

Void signContract(Contract)

This is called if checkContract() returns true. This method signs the contract and sends it back to the buyer.

Class Contract

This class encapsulates methods to create and send contracts.

ReturnType createContract(Offer)

How this works (Protocol):

The negotiation component of the buyer receives offers from the Auction component. Then, it proceeds to choose a node to do business with. This node is either chosen by the node operator manually or automatically by the component itself.

The method isOfferAccepted(int offerId) listens for acceptance or rejection of a particular offer. If we receive a string with "+ offerId" then it means that Offer with offerId has been accepted and we can proceed to check and sign the contract. Once this has been done the signed contract is sent back to the buyer who proceeds to migrate the job. If there are any discrepancies in the contract the contract is sent back without signing and it is upto the buyer to take care of the mistake.

Note: This is just a barebones protocol which still needs to be refined.