```php
<?php
/**
 * A Micro Framework for the implementation of
 * Interaction Type Approach to Relationships Management
 * by G.Nota & Rossella Aiello
 * JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING. Vol. 8. Pag.1-15
 * ISSN:1868-5137.
 * https://www.academia.edu/55124231/The_interaction_type_approach_to_relationships_management
 *
 * @author rosario.carvello@gmail.com
 */

```

```php
/**
 * Class CommunicationInfrastructure.
 *
 * A basic communication infrastructure used for activating Interactions
 *
 * @note This is a general purpose class realizing a very simple infrastructure
 * used for creating links and message interchange occurring on interactions'
 * activation.
 */
class CommunicationInfrastructure
{
    /**
     * Activate the InteractionType by producing an Interaction.
     *
     * @param InteractionType $interaction
     * @return void
     */
    public function activateInteraction(InteractionType $interaction)
    {
        $relation = $interaction->getRelation();
        $message = $interaction->getMessage();
        $activeEntities = $relation->getActiveEntities();
        $senders = $this->fetchSenders($activeEntities);
        $receivers = $this->fetchReceivers($activeEntities);
        echo "Interaction results:";
        foreach ($senders as $sender) {
            foreach ($receivers as $receiver) {
                if (!empty($sender->getMessage()->getText())) {
                    $textMessage = $sender->getMessage()->getText();
                } else {
                    $textMessage = $message->getText();
                }
                $sender->getRole()->sendMessageToFrom($textMessage,$receiver,$sender);
                echo "<br>";
            }
        }
    }

    /**
     * Fetch all senders from the given array of active entities
     *
     * @param array $activeEntities Array containing all the active entities
     * @return array
     */
    private function fetchSenders($activeEntities)
    {
        $senders = array();
        foreach ($activeEntities as $activeEntity) {
            $role = $activeEntity->getRole();
            if (get_class($role) == "Sender")
                $senders[] = $activeEntity;
        }
        return $senders;
    }
```

```
68
69      /**
70       * Fetch all senders from the given array of active entities
71       *
72       * @param array $activeEntities Array containing all the active entities
73       * @return array
74       */
75      private function fetchReceivers(&$activeEntities)
76      {
77          $receivers = array();
78          foreach ($activeEntities as $activeEntity) {
79              $role = $activeEntity->getRole();
80              if ($role->getName() == "Receiver")
81                  $receivers[] = $activeEntity;
82          }
83          return $receivers;
84      }
85
86  }
87
```

```
88  /**
89   * Class ActiveEntity
90   *
91   * Active entity is an organization, an individual or an auto-mated
92   * component capable of performing a behaviour during the interaction
93   * with other active entities
94   */
95  class ActiveEntity
96  {
97      private $name;
98      private $role;
99      private $message;
100
101     /**
102      * Constructor
103      *
104      * @param string $name The active entity name
105      * @param Role $role   The role active entity regarding the relationship
106      *                     with other active entities
107      */
108     public function __construct($name, Role $role)
109     {
110         $this->setName($name);
111         $this->setRole($role);
112         $this->message = new Message();
113     }
114
115     /**
116      * Get name
117      * @return string
118      */
119     public function getName()
120     {
121         return $this->name;
122     }
123
124     /**
125      * Set name
126      * @param string $name
127      */
128     private function setName($name)
129     {
130         $this->name = $name;
131     }
132
133     /**
134      * Get role
135      * @return Role
136      */
137     public function getRole()
```

```
    {
        return $this->role;
    }

    /**
     * Set role
     * @param Role $role
     */
    public function setRole($role)
    {
        $this->role = $role;
    }

    /**
     * Get the custom message
     * @return Message
     */
    public function getMessage()
    {
        return $this->message;
    }

    /**
     * Set a custom message in interaction
     * @param Message $message
     */
    public function setMessage(Message $message)
    {
            $this->message = $message;
    }

}
```

```
/**
 *  Class Relation.
 *
 *  Represents a logical or physical connection between components of a structure.
 *  Through relation-ships communication becomes possible sustaining the interaction
 *  between active entities
 */
class Relationship
{
    private $name;
    private $activeEntities = array();

    /**
     * Constructor
     *
     * @param $name string A name for the relationship
     */
    public function __construct($name)
    {
        $this->setName($name);
    }

    /**
     * Add the given active entity to the relationship
     *
     * @param ActiveEntity $entity The active entity to add
     *
     * @return void
     */
    public function addEntity(ActiveEntity $entity)
    {
        $this->activeEntities[] = $entity;

    }

    /**
     * Gets the active entities of the relationship
```

```
     *
     * @return array
     */
    public function getActiveEntities()
    {
        return $this->activeEntities;
    }

    /**
     * Get the name of the relationship
     *
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * Set the name of the relationship
     * @param string $name
     */
    private function setName($name): void
    {
        $this->name = $name;
    }


}
```

```
/**
 * Class InteractionType
 *
 * Is the structural element that gives form to one kind of interaction.
 * An instance of InteractionType qualify an Interaction in the sense
 * that it provides external shape or settings to the interaction.
 */
class InteractionType
{
    private $name;
    private $relation;
    private $message;

    /**
     * Constructor
     *
     * @param string $name Name for a semantic description of the Interaction Type
     * @param Relationship $relation The Relationship containing active entities that interact
     * @param Message|null $message The message produced/consumed on Interaction
     */
    public function __construct($name, Relationship $relation, Message $message = null)
    {
        $this->setName($name);
        $this->setRelation($relation);
        $this->setMessage($message);
    }

    /**
     * Get the name of Interaction Type
     *
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * Set the name of Interaction Type
     *
```

```php
     * @param string $name
     */
    private function setName($name): void
    {
        $this->name = $name;
    }

    /**
     * Get the Relationship will be acted by Interaction Type
     *
     * @return Relationship
     */
    public function getRelation()
    {
        return $this->relation;
    }

    /**
     * Set the Relationship will be acted by Interaction Type
     *
     * @param mixed $relation
     */
    private function setRelation($relation)
    {
        $this->relation = $relation;
    }

    /**
     * Get the message produced/consumed on Interaction
     *
     * @return Message
     */
    public function getMessage()
    {
        return $this->message;
    }

    /**
     * Set the message produced/consumed on Interaction
     *
     * @param mixed $message
     */
    public function setMessage($message): void
    {
        $this->message = $message;
    }


}

---

/**
 *  Class Role
 *
 *  Abstraction for the role assumed by an active entity during an interaction
 */
abstract class Role
{
    public function getName(){
        return get_class($this);
    }
}

---

/**
 *  Class Sender
 *
 *  Qualify an active entity to assume the role for sending a message.
 *  Message to send is defined by the Interaction Type or Actives Entity
 *  qualified as Senders
 */
```

```php
class Sender extends Role
{

    /**
     * Send the given text message to the given Receiver from the given Sender.
     * Both Receiver and Sender are object o Active Entity where respectively
     * qualified for receiving or sending message
     *
     *
     * @param string $textMessage  The message to send
     * @param ActiveEntity $toReceiver The active entity qualified as receiver
     * @param ActiveEntity $fromSender The active entity qualified as sender
     * @return void
     */
    public function sendMessageToFrom($textMessage, ActiveEntity $toReceiver,ActiveEntity $fromSender){
       if (!empty($textMessage)) {
           $output = "<p style='color: #1c7430'>The sender <b>{SENDER_NAME}</b> send the message '<b><i>{MESSAGE_TEXT}</i></b>' to the receiver <b>{RECEIVER_NAME}</b></p>";
           $output = str_replace("{SENDER_NAME}", $fromSender->getName(), $output);
           $output = str_replace("{MESSAGE_TEXT}", $textMessage, $output);
           $output = str_replace("{RECEIVER_NAME}", $toReceiver->getName(), $output);
           echo $output;
           $toReceiver->getRole()->receiveMessageFromTo($textMessage, $fromSender,$toReceiver);
       }
   }
}

```

```php
/**
 *  Class Receiver
 *
 *  Qualify an active entity to assume the role for receive a message.
 *  The received message is defined by the Interaction Type or by Actives
 *  Entity qualified as Senders
 */
class Receiver extends Role
{
    /**
     * Receive the given text message from the given Sender in the given Receiver.
     * Both Receiver and Sender are object o Active Entity where respectively
     * qualified for receiving or sending message
     *
     * @param string $textMessage The message to receive
     * @param ActiveEntity $fromSender The active entity qualified as sender
     * @param ActiveEntity $toReceiver The active entity qualified as receiver
     * @return void
     */
    public function receiveMessageFromTo($textMessage, ActiveEntity $fromSender,ActiveEntity $toReceiver ){
        if (!empty($textMessage)) {
            $output = "<p style=\"color: red\">The receiver <b>{RECEIVER_NAME}</b> received the message '<b><i>{MESSAGE_TEXT}</i></b>' from the sender <b>{SENDER_NAME}</b></p>";
            $output = str_replace("{SENDER_NAME}", $fromSender->getName(), $output);
            $output = str_replace("{MESSAGE_TEXT}", $textMessage, $output);
            $output = str_replace("{RECEIVER_NAME}", $toReceiver->getName(), $output);
            echo $output;
        }
    }
}

```

```php
/**
 *  Class Message
 *  A basic class for a text message representation.
 */
class Message
{
    private $text;

    /**
     * @param string $text The text message
     */
    public function __construct($text = null)
    {
```

```php
        $this->setText($text);
    }

    /**
     * Get the text of the message
     *
     * @return string
     */
    public function getText()
    {
        return $this->text;
    }

    /**
     * Set the text of the message
     *
     * @param string $text
     * @return void
     */
    public function setText($text)
    {
        $this->text = $text;
    }


}



// Usage examples

// 1)  Building the Structure
$communication = new CommunicationInfrastructure();
$receiver = new Receiver();
$sender = new Sender();

$omcr = new ActiveEntity("OMCR Supplier", $receiver);
$stamec = new ActiveEntity("STAMEC Manufacturing", $sender);

$message = new Message("Please, provide me an extimation cost for Part Number 01");

$relationStamecAndSuppliers = new Relationship("Manufacturing-Suppliers");
$relationStamecAndSuppliers->addEntity($omcr);
$relationStamecAndSuppliers->addEntity($stamec);

$interactionInterchangeRDA = new InteractionType("Purchase Quotations", $relationStamecAndSuppliers, $message);

// 1) Perform an Interaction by instantiating the InteractionType Purchase Quotations
echo "Instantiate InteractionType <b>{$interactionInterchangeRDA->getName()}</b> From Stamec to OMCR";
printInteractionTypeInfo($interactionInterchangeRDA);
$communication->activateInteraction($interactionInterchangeRDA);
echo "<hr>";

// 2) Re instantiate the previous InteractionType Purchase Quotations by adding DAYTON as new receiver
$dayton = new ActiveEntity("DAYTON Supplier", $receiver);
$relationStamecAndSuppliers->addEntity($dayton);

echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by adding DAYTON as new receiver";
printInteractionTypeInfo($interactionInterchangeRDA);
$communication->activateInteraction($interactionInterchangeRDA);
echo "<hr>";

// 3) Re instantiate the previous InteractionType Purchase Quotations by defining a new message
$message->setText("Please, provide me an extimation cost for Part Number 02");

echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by defining a new message";
printInteractionTypeInfo($interactionInterchangeRDA);
$communication->activateInteraction($interactionInterchangeRDA);
echo "<hr>";

// 4) Re instantiate the previous InteractionType Purchase Quotations by simulating the responses from receivers.
```

```php
$stamec->setRole($receiver);
$omcr->setRole($sender);
$dayton->setRole($sender);
$omcr->getMessage()->setText("The parts number 01 and 02 you previosly required costs, respectly, 1000 and 1020");
$dayton->getMessage()->setText("The parts number 01 and 02 you previosly required costs, respectly, 1200 and 1280. Discount of 20% within the end of current month");

echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by simulating the responses from receivers.<br>";
echo "This can be easly obtaintened by interchanging roles (by setting Stamec as the receiver and DAYTON,OMCR as senders) <br> and by setting the custom message provided by each senders";
printInteractionTypeInfo($interactionInterchangeRDA);
$communication->activateInteraction($interactionInterchangeRDA);
echo "<hr>";


/*
 *  Helper function for printing information about the structure of Interaction Type
 */
function printInteractionTypeInfo(InteractionType $interactionInterchangeRDA)
{
    echo "<br><p style='background-color: #d3d9df'>Structure information: <br>";
    echo "Interaction Type: <b>" . $interactionInterchangeRDA->getName() . "</b><br>";
    $relationShip = $interactionInterchangeRDA->getRelation();
    echo "Relationship: <b>" . $relationShip->getName() . "</b><br>";
    $activeEntities = $relationShip->getActiveEntities();
    echo "Active Entities: ";
    foreach ($activeEntities as $activeEntity) {
        echo "<b>" . $activeEntity->getName() . "</b>";
        $role = $activeEntity->getRole();
        echo "<sup>(" . $role->getName() . ")</sup>  ";
    }
    echo "<br></p><hr>";
}
```