

```

1  <?php
2  /**
3   * A Micro Framework for the implementation of
4   * Interaction Type Approach to Relationships Management
5   * by G.Nota & Rossella Aiello
6   * JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING. Vol. 8. Pag.1-15
7   * ISSN:1868-5137.
8   * https://www.academia.edu/55124231/The_interaction_type_approach_to_relationships_management
9   *
10  * @author rosario.carvello@gmail.com
11  */
12
13 /**
14  * Class CommunicationInfrastructure.
15  *
16  * A basic communication infrastructure used for activating Interactions
17  *
18  * @note This is a general purpose class realizing a very simple infrastructure
19  * used for creating links and message interchange occurring on interactions'
20  * activation.
21  */
22 class CommunicationInfrastructure
23 {
24     /**
25      * Activate the InteractionType by producing an Interaction.
26      *
27      * @param InteractionType $interaction
28      * @return void
29      */
30     public function activateInteraction(InteractionType $interaction)
31     {
32         $relation = $interaction->getRelation();
33         $message = $interaction->getMessage();
34         $activeEntities = $relation->getActiveEntities();
35         $senders = $this->fetchSenders($activeEntities);
36         $receivers = $this->fetchReceivers($activeEntities);
37         echo "Interaction results:";
38         foreach ($senders as $sender) {
39             foreach ($receivers as $receiver) {
40                 if (!empty($sender->getMessage()->getText())) {
41                     $textMessage = $sender->getMessage()->getText();
42                 } else {
43                     $textMessage = $message->getText();
44                 }
45                 $sender->getRole()->sendMessageToFrom($textMessage,$receiver,$sender);
46                 echo "<br>";
47             }
48         }
49     }
50 }
51
52 /**
53  * Fetch all senders from the given array of active entities
54  *
55  * @param array $activeEntities Array containing all the active entities
56  * @return array
57  */
58 private function fetchSenders($activeEntities)
59 {
60     $senders = array();
61     foreach ($activeEntities as $activeEntity) {
62         $role = $activeEntity->getRole();
63         if (get_class($role) == "Sender")
64             $senders[] = $activeEntity;
65     }
66     return $senders;
67 }

```

```

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()

```

```

139     {
140         return $this->role;
141     }
142     /**
143      * Set role
144      * @param Role $role
145      */
146     public function setRole($role)
147     {
148         $this->role = $role;
149     }
150
151     /**
152      * Get the custom message
153      * @return Message
154      */
155     public function getMessage()
156     {
157         return $this->message;
158     }
159
160     /**
161      * Set a custom message in interaction
162      * @param Message $message
163      */
164     public function setMessage(Message $message)
165     {
166         $this->message = $message;
167     }
168 }
169
170

```

---

```

171 /**
172  * Class Relation.
173  *
174  * Represents a logical or physical connection between components of a structure.
175  * Through relation-ships communication becomes possible sustaining the interaction
176  * between active entities
177  */
178 class Relationship
179 {
180     private $name;
181     private $activeEntities = array();
182
183     /**
184      * Constructor
185      *
186      * @param $name string A name for the relationship
187      */
188     public function __construct($name)
189     {
190         $this->setName($name);
191     }
192
193     /**
194      * Add the given active entity to the relationship
195      *
196      * @param ActiveEntity $entity The active entity to add
197      *
198      * @return void
199      */
200     public function addEntity(ActiveEntity $entity)
201     {
202         $this->activeEntities[] = $entity;
203     }
204
205     /**
206      * Gets the active entities of the relationship
207

```

```

208     * @return array
209     */
210     public function getActiveEntities()
211     {
212         return $this->activeEntities;
213     }
214 }
215
216 /**
217  * Get the name of the relationship
218  *
219  * @return string
220  */
221     public function getName()
222     {
223         return $this->name;
224     }
225
226 /**
227  * Set the name of the relationship
228  * @param string $name
229  */
230     private function setName($name): void
231     {
232         $this->name = $name;
233     }
234 }
235 }
236 }
237
238 /**
239  * Class InteractionType
240  *
241  * Is the structural element that gives form to one kind of interaction.
242  * An instance of InteractionType qualify an Interaction in the sense
243  * that it provides external shape or settings to the interaction.
244  */
245 class InteractionType
246 {
247     private $name;
248     private $relation;
249     private $message;
250
251     /**
252      * Constructor
253      *
254      * @param string $name Name for a semantic description of the Interaction Type
255      * @param Relationship $relation The Relationship containing active entities that interact
256      * @param Message|null $message The message produced/consumed on Interaction
257      */
258     public function __construct($name, Relationship $relation, Message $message = null)
259     {
260         $this->setName($name);
261         $this->setRelation($relation);
262         $this->setMessage($message);
263     }
264
265     /**
266      * Get the name of Interaction Type
267      *
268      * @return string
269      */
270     public function getName()
271     {
272         return $this->name;
273     }
274
275     /**
276      * Set the name of Interaction Type
277      *

```

```

278 */
279 */
280 private function setName($name): void
281 {
282     $this->name = $name;
283 }
284
285 /**
286  * Get the Relationship will be acted by Interaction Type
287  *
288  * @return Relationship
289  */
290 public function getRelation()
291 {
292     return $this->relation;
293 }
294
295 /**
296  * Set the Relationship will be acted by Interaction Type
297  *
298  * @param mixed $relation
299  */
300 private function setRelation($relation)
301 {
302     $this->relation = $relation;
303 }
304
305 /**
306  * Get the message produced/consumed on Interaction
307  *
308  * @return Message
309  */
310 public function getMessage()
311 {
312     return $this->message;
313 }
314
315 /**
316  * Set the message produced/consumed on Interaction
317  *
318  * @param mixed $message
319  */
320 public function setMessage($message): void
321 {
322     $this->message = $message;
323 }
324
325 }
326 }
327
328 /**
329  * Class Role
330  *
331  * Abstraction for the role assumed by an active entity during an interaction
332  */
333 abstract class Role
334 {
335     public function getName(){
336         return get_class($this);
337     }
338 }
339
340 /**
341  * Class Sender
342  *
343  * Qualify an active entity to assume the role for sending a message.
344  * Message to send is defined by the Interaction Type or Actives Entity
345  * qualified as Senders
346  */

```

```

348 class Sender extends Role
349 {
350     /**
351      * Send the given text message to the given Receiver from the given Sender.
352      * Both Receiver and Sender are object o Active Entity where respectively
353      * qualified for receiving or sending message
354      *
355      *
356      * @param string $textMessage The message to send
357      * @param ActiveEntity $toReceiver The active entity qualified as receiver
358      * @param ActiveEntity $fromSender The active entity qualified as sender
359      * @return void
360      */
361     public function sendMessageToFrom($textMessage, ActiveEntity $toReceiver, ActiveEntity $fromSender){
362         if (!empty($textMessage)) {
363             $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>";
364             $output = str_replace("{SENDER_NAME}", $fromSender->getName(), $output);
365             $output = str_replace("{MESSAGE_TEXT}", $textMessage, $output);
366             $output = str_replace("{RECEIVER_NAME}", $toReceiver->getName(), $output);
367             echo $output;
368             $toReceiver->getRole()->receiveMessageFromTo($textMessage, $fromSender, $toReceiver);
369         }
370     }
371 }
372
373 /**
374  * Class Receiver
375  *
376  * Qualify an active entity to assume the role for receive a message.
377  * The received message is defined by the Interaction Type or by Actives
378  * Entity qualified as Senders
379  */
380 class Receiver extends Role
381 {
382     /**
383      * Receive the given text message from the given Sender in the given Receiver.
384      * Both Receiver and Sender are object o Active Entity where respectively
385      * qualified for receiving or sending message
386      *
387      * @param string $textMessage The message to receive
388      * @param ActiveEntity $fromSender The active entity qualified as sender
389      * @param ActiveEntity $toReceiver The active entity qualified as receiver
390      * @return void
391      */
392     public function receiveMessageFromTo($textMessage, ActiveEntity $fromSender, ActiveEntity $toReceiver ){
393         if (!empty($textMessage)) {
394             $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>";
395             $output = str_replace("{SENDER_NAME}", $fromSender->getName(), $output);
396             $output = str_replace("{MESSAGE_TEXT}", $textMessage, $output);
397             $output = str_replace("{RECEIVER_NAME}", $toReceiver->getName(), $output);
398             echo $output;
399         }
400     }
401 }
402
403 /**
404  * Class Message
405  * A basic class for a text message representation.
406  */
407 class Message
408 {
409     private $text;
410
411     /**
412      * @param string $text The text message
413      */
414     public function __construct($text = null)
415     {

```

```

417     }
418
419     /**
420      * Get the text of the message
421      *
422      * @return string
423      */
424     public function getText()
425     {
426         return $this->text;
427     }
428
429     /**
430      * Set the text of the message
431      *
432      * @param string $text
433      * @return void
434      */
435     public function setText($text)
436     {
437         $this->text = $text;
438     }
439
440 }
441
442
443
444
445 // Usage examples
446
447 // 1) Building the Structure
448 $communication = new CommunicationInfrastructure();
449 $receiver = new Receiver();
450 $sender = new Sender();
451
452 $omcr = new ActiveEntity("OMCR Supplier", $receiver);
453 $stamec = new ActiveEntity("STAMEC Manufacturing", $sender);
454
455 $message = new Message("Please, provide me an extimation cost for Part Number 01");
456
457 $relationStamecAndSuppliers = new Relationship("Manufacturing-Suppliers");
458 $relationStamecAndSuppliers->addEntity($omcr);
459 $relationStamecAndSuppliers->addEntity($stamec);
460
461 $interactionInterchangeRDA = new InteractionType("Purchase Quotations", $relationStamecAndSuppliers, $message);
462
463 // 1) Perform an Interaction by instantiating the InteractionType Purchase Quotations
464 echo "Instantiate InteractionType <b>{$interactionInterchangeRDA->getName()}</b> From Stamec to OMCR";
465 printInteractionTypeInfo($interactionInterchangeRDA);
466 $communication->activateInteraction($interactionInterchangeRDA);
467 echo "<hr>";
468
469 // 2) Re instantiate the previous InteractionType Purchase Quotations by adding DAYTON as new receiver
470 $dayton = new ActiveEntity("DAYTON Supplier", $receiver);
471 $relationStamecAndSuppliers->addEntity($dayton);
472
473 echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by adding DAYTON as new receiver";
474 printInteractionTypeInfo($interactionInterchangeRDA);
475 $communication->activateInteraction($interactionInterchangeRDA);
476 echo "<hr>";
477
478 // 3) Re instantiate the previous InteractionType Purchase Quotations by defining a new message
479 $message->setText("Please, provide me an extimation cost for Part Number 02");
480
481 echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by defining a new message";
482 printInteractionTypeInfo($interactionInterchangeRDA);
483 $communication->activateInteraction($interactionInterchangeRDA);
484 echo "<hr>";
485
486 // 4) Re instantiate the previous InteractionType Purchase Quotations by simulating the responses from receivers.

```

```

487 $stamec->setRole($receiver);
488 $omcr->setRole($sender);
489 $dayton->setRole($sender);
490 $omcr->getMessage()->setText("The parts number 01 and 02 you previously required costs, respectively, 1000 and 1020");
491 $dayton->getMessage()->setText("The parts number 01 and 02 you previously required costs, respectively, 1200 and 1280. Discount of 20% within the end of current month");
492
493 echo "Re-instantiate the previous InteractionType <b>{$interactionInterchangeRDA->getName()}</b> by simulating the responses from receivers.<br>";
494 echo "This can be easily obtained 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";
495 printInteractionTypeInfo($interactionInterchangeRDA);
496 $communication->activateInteraction($interactionInterchangeRDA);
497 echo "<hr>";
498
499
500 /*
501  * Helper function for printing information about the structure of Interaction Type
502  */
503 function printInteractionTypeInfo(InteractionType $interactionInterchangeRDA)
504 {
505     echo "<br><p style='background-color: #d3d9df'>Structure information: <br>";
506     echo "Interaction Type: <b>" . $interactionInterchangeRDA->getName() . "</b><br>";
507     $relationship = $interactionInterchangeRDA->getRelation();
508     echo "Relationship: <b>" . $relationship->getName() . "</b><br>";
509     $activeEntities = $relationship->getActiveEntities();
510     echo "Active Entities: ";
511     foreach ($activeEntities as $activeEntity) {
512         echo "<b>" . $activeEntity->getName() . "</b>";
513         $role = $activeEntity->getRole();
514         echo "<sup>" . $role->getName() . "</sup> ";
515     }
516     echo "<br></p><hr>";
517 }

```