

# Agent Behavior Alignment: a Mechanism to Overcome Problems in Agent Interactions During Runtime

Gerben G. Meyer, Nick B. Szirbik

Department of Business & ICT, Faculty of Economics, Management and Organization, University of Groningen, Landleven 5, P.O. Box 800, 9700 AV Groningen, The Netherlands, +31 50 363 {7194 / 8125}  
{g.g.meyer,n.b.szirbik}@rug.nl

**Abstract.** When two or more agents interacting, their behaviors are not necessarily matching. Automated ways to overcome conflicts in the behavior of agents can make the execution of interactions more reliable. Such an alignment mechanism will reduce the necessary human intervention. This paper shows how to describe a policy for alignment, which an agent can apply when its behavior is in conflict with other agents. An extension of Petri Nets is used to capture the intended interaction of an agent in a formal way. Furthermore, a mechanism based on machine learning is implemented, to enable an agent to choose an appropriate alignment policy with collected problem information. Human intervention can reinforce certain successful policies in a given context, and can also contribute by adding completely new policies. Experiments have been conducted to test the applicability of the alignment mechanism and the main results are presented here.

## 1 Introduction

This paper presents a method for agents to mutually adjust their behaviors in an automatic way. Behavioral alignment is implemented as a mechanism that allows the agents to successfully complete an interaction. The method was implemented and experiments have been conducted. These experiments have shown that automated procedures to overcome problems in agent-to-agent interactions can make the whole multi-agent system execution more reliable, and have resulted in less human intervention.

When two or more agents (human or software) interact as part of a business process instance (BPI) [17], they can either follow an established protocol, or use their own experience and judgment. In the first case, the agents should learn or be instructed about the protocol beforehand. In the second case, there either is no protocol, or it does not cover special circumstances that occur in this particular instance. Furthermore, the agents have to use their own experience, acquired in previous similar interaction. Before the interaction, the agents will build an *intended behavior* (their own course of action) and they will also assume

what the other agents are doing (*expected behavior*). Together these beliefs form a description of an *intended interaction*, which is also called an *interaction belief* [18].

When two agents are going to perform an interaction, and they have inconsistent behaviors, the resulting interaction will not achieve its goal, especially when the agents do not have mechanisms to change their behaviors. If both agents are able to realize, after the interaction has started, that there is a conflict in their beliefs about the interaction, they need to be able to change their behavior in a collaborative manner that promises success in ending the whole interaction. In successive steps of trial and error, they will be able to manage to finish the interaction. If exactly the same agents are interacting again in the future, they can use the final behavior (which they memorize) that led to success.

The process of collaboratively changing behaviors will be called *alignment*. Two or more behaviors that have been aligned successfully are named *matching behaviors*. If the agents do not manage to align their behavior on the fly, their complete behaviors have to be compared externally. Because agents do not know each other's behaviors, a superior agent who can access both behaviors is the most suitable to align these behaviors. This agent can be a software agent, but can also be human. This procedure, if one of the agents is calling for a higher level agent to align the behaviors, is called *escape* mode. A higher level agent can also *intervene* in the interaction process a priori or during the interaction, to align the agents' intended behaviors. In both cases, the higher level agent adapts the behavior of the agents [14]. A third possibility, which can often be found in real life, is to align their behavior a priori of the interaction, by exchanging each intended behavior and analyze and discuss beforehand. Analysis can reveal potential deadlock and conflicts. The interacting agents can align their behaviors before the interaction starts.

This paper only focuses on how agents can individually align their behavior *on-the-fly*. A discussion about the other types of alignment can be found in [11]. The paper is structured as follows. In the next section related work is discussed. Section 3 explains how the behavior of agents are modeled and executed. Section 4 discusses how primitive and advanced operations allow for behavior modifications. In section 5 a new mechanism for automatic behavior alignment is presented. Section 6 unveils some of the conducted experiments, with their result. Section 7 ends the paper with discussion and conclusions.

## 2 Related work

There are two long-established agent-research areas that contributed to the architecture of the communication and coordination mechanisms in MAS. These are the agent communication language (ACL) research, and the interaction protocol (IP) research. However, there are scarce results in terms of how IPs and ACLs specifications could be coherently linked together. Most approaches concentrate on one of these two sides, either limiting via IPs the agents' autonomy (agents obey central protocols), either leaving open the consequences of

communication, leading to loss of coherence in the overall interactional process executed by the agents. One of the promising approaches that tries to bridge the communication specifications to the way agents adhere to previously created committed commitments is proposed in [15], by using *Conversation Patterns*. These are symbolic constructs that regulate agent-to-agent interaction, but they are modified via machine learning. The question remains if these frames-based models are appropriate for complex business processes, where the execution semantics derived from the distributed structure of the frames that are derived from the conversation patterns can become combinatorial complex.

Dignum [5] emphasizes the importance of the link between Petri net based analysis techniques in the context of protocolized agent communication, and their link to the description of dynamic properties of agent conversations. A non-agent approach that tackles similar problems is the modeling and execution of inter-organizational workflows [2]. However, this approach is based on a central perspective, similar to the IP approach. The main advantage of the approach is that it is possible to determine when a workflow is sound [3]. Another related problem is how to dynamically change workflows, and it has been addressed by Ellis et al. [6]. Ellis and Keddara [7] describe how a workflow can be dynamically changed using another workflow, which defines the change. Recently, distributed workflows expressed as agent behaviors have been investigated by Meyer and Szirbik [11] who have developed a formal way to model agent behavior as executed in agent to agent interaction as Behavior Nets. Because these Behavior Nets are an extension of Petri Nets, workflow techniques as proposed by Van der Aalst and Ellis can also be applied to these Behavior Nets. Behavior Nets are part of the agent modeling language TALL [17], which is currently under development. As for the overlapping research themes in workflows and agents, a thorough review of is presented in [19].

### 3 Modeling and executing agent behavior

As mentioned before, agents' behaviors as exhibited in agent to agent interactions are modeled by Behavior Nets, as formally defined by Meyer and Szirbik in [11], which is an extension of Petri Nets. Petri Nets are a class of modeling tools, which originate from the work of Petri [13]. The advantage of Petri Nets is that they have a well defined mathematical foundation, but also a clear graphical notation [16]. Because of the graphical notation, Petri Nets are powerful design tools, which can be used for communication between the people who are engaged in the design process. On the other hand, because of the mathematical foundation, mathematical models of the behavior of the system can be set up. The mathematical formalism also allows for various analysis techniques of the Petri Net. Moreover, as Petri Nets are commonly used for modeling business processes (see for example [1]), applying Petri Nets for modeling agent behavior makes the approach suitable for simulating and supporting business processes with agents.

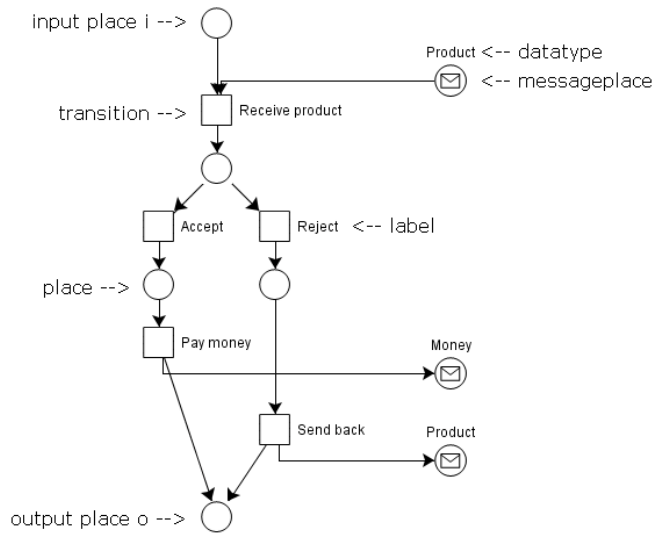


Fig. 1: Example of a Behavior Net

An example of a Behavior Net can be seen in figure 1. The definition of Behavior Nets is partially based on (and an extension of) Workflow nets [3], Self-Adaptive Recovery Nets [8] and Colored Petri Nets [10]. The Behavior Net as shown in figure 1 illustrates most of the Behavior Net constructs. The figure depicts the behavior of a buyer, in a buyer-seller interaction, who will receive a product, and either accepts the product and pays the money, or reject the product and sends it back.

The modeled behavior of the agent can be executed directly, however, to enable on-the-fly problem detection, change, and alignment of behaviors, an extended version of the behavior is used. For this, a two-layered approach is used, based on [9]. These two layers, and the relations between them, are shown in figure 2. The top layer is the original agent behavior, where the bottom layer is the extended behavior, for enabling the alignment mechanism. On both layers, the behavioral descriptions are based on the Petri Net extension as described above. In this way, when the behavior is executed in the compiled layer, it is still possible to visualize the progress of this execution in the original modeled behavior.

During run time, the state of the compiled layer will change, as tokens will flow through the behavior. The state of the other layer will be updated according to the state of the compiled layer, to make it possible to visualize the progress of the execution in the original modeled behaviors. During align time (when the agent is changing its own behavior on-the-fly due to a problem with executing it in this particular interaction) the agent changes the behavior on the behavior layer. The compiled layer has to be recreated by recompiling the behavior layer. The compilation step will be further discussed in section 5.1.

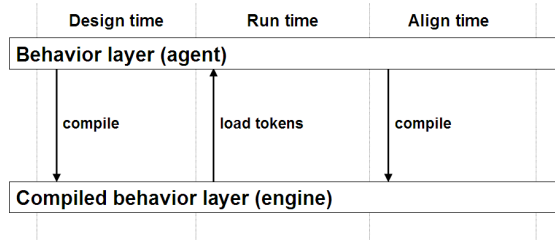


Fig. 2: Two layers to describe behavior

## 4 Modifying agent behavior on-the-fly

Since we want to modify behaviors in a consistent and sound way, the manner in which modifying operations are defined is important. For this purpose, two types of operations on Behavior Nets can be defined. The *primitive* operations allow for simple addition and deletion of nodes and arcs, whereas *advanced* operations allow for more complex changes.

### 4.1 Primitive operations

Based on [8], some primitive operations are identified for modifying Behavior Net structures. In total, 16 primitive operations have been implemented in the system, which are shown in table 1.

<code>createPlace(x)</code>	<code>deletePlace(x)</code>
<code>createMessagePlace(x)</code>	<code>deleteMessagePlace(x)</code>
<code>createTransition(x)</code>	<code>deleteTransition(x)</code>
<code>createArc(x<sub>1</sub>,x<sub>2</sub>)</code>	<code>deleteArc(x<sub>1</sub>,x<sub>2</sub>)</code>
<code>setLabel(x,x')</code>	<code>setDatatype(x,d)</code>
<code>createGuard(x,g)</code>	<code>deleteGuard(x,g)</code>
<code>createBinding(x,b)</code>	<code>deleteBinding(x,b)</code>
<code>createToken(x,t)</code>	<code>deleteToken(x,t)</code>

Table 1: The primitive operations

These primitive operations can be used and are sufficient to build Behavior Nets from scratch, and populate them with tokens. By using the `deleteToken` operation, it is even possible to simulate the flow.

Guards are an addition to the basic Petri Nets. In Behavior Nets, they can be added to and deleted from arcs, and their role is to provide certain conditions that need to be fulfilled before a token can travel along that arc. A guard can put constraints on values which are kept within the token. Guards are used to detect exceptions, that potentially lead to behavior net modifications. Furthermore,

guards enable a straightforward implementation of the typical deterministic OR-split, representing a choice.

A second addition to the classical Petri Nets are bindings, which can be added to transitions. The role of bindings is to modify the values kept by tokens. These values can later on be used by the guards.

Applying a primitive operation does not guarantee preservation of soundness of the behavior. The discussion of soundness is outside the scope of the paper, but interested readers can find the definition of soundness in [3].

## 4.2 Advanced sound operations

In addition to the primitive operations, a set of advanced operations is defined, based on the principles presented in [3,4]. These advanced operation can also be used to modify the structure of a Behavior Net, but contrary to primitive operations, they have the advantage of preserving the soundness of the initial net. All these operations can be constructed by combining a set of primitive operations. The defined and implemented advanced operations are:

- **division** and **aggregation**, which divides one transition into two sequential transitions, or vice versa,
- **parallelization** and **sequentialization**, which puts two sequential transitions in parallel, or vice versa,
- **specialization** and **generalization**, which divides a transition into two mutually exclusive specializations (an OR-split), or vice versa,
- **iteration** and **noIteration**, which replaces a transition with an iteration over a transition, or vice versa,
- **receiveMessage** and **notReceiveMessage**, which adds or deletes an incoming message place,
- **sendMessage** and **notSendMessage**, which adds or deletes an outgoing message place.

For modeling the change schemes the approach of Ellis et al. [7] is used. The migration of the tokens of the old behavior to the new behavior can be exactly defined by modeling the behavior change as a Petri Net. Figure 3 shows how the migration for the operation **parallelization** can be modeled.

## 5 The aligning mechanism

In the previous section, operations have been defined how to alter Behavior Nets. However, a mechanism is required to trigger when a Behavior Net should be altered, and how. In this section, alignment policies will be introduced, as well as a mechanism for agents to select such a policy. For selecting a policy, certain information has to be collected about the conflict and its context.

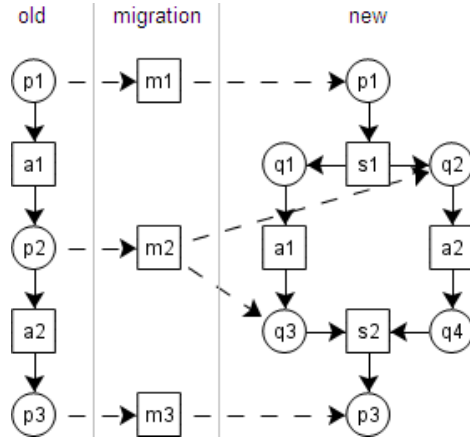


Fig. 3: Migration of old to new behavior

### 5.1 Collecting problem information

As discussed previously, the modeled behavior itself is not executed, but the compiled version of it. By compiling, extra nodes are added to the original behavior, for collecting information when there is a problem in the execution of the behavior. This information can be used for selecting a proper alignment policy (this will be discussed later in this section), which is the best for dealing with the problem. Figure 4 shows what information can be collected by the extra nodes added when the behavior is compiled.

The highlighted nodes are the nodes of the original behavior. The `align` transition is the transition where all collected information is gathered, and (if necessary) an alignment policy is chosen and executed. In this example, the original behavior has only one transition `act` that expects one incoming message `m1`. This transition `act` failed to execute, therefore all the gathered information is about transition `act`, to make the `align` transition able to choose an alignment policy which will successfully align the behavior around the transition `act`. Despite the simplicity of this example, it shows all the ways of how information used for choosing an alignment policy is collected, as for more complex Behavior Nets, the same kind of nodes are added. Problem information is gathered in three different types of occurrences, as discussed next. In all three cases, a timeout mechanism is used to identify the occurrence of non-moving tokens. The information is collected separately for every transition, where problems arise.

*I. When a token from a place is not moving* (Place `start` in this example.) Here, transition `e1` will send information about this occurrence to the `align` transition. It will send the following information:

- `recToken`, short for received token, to inform the `align` transition that there is a non-moving token on that place in the behavior

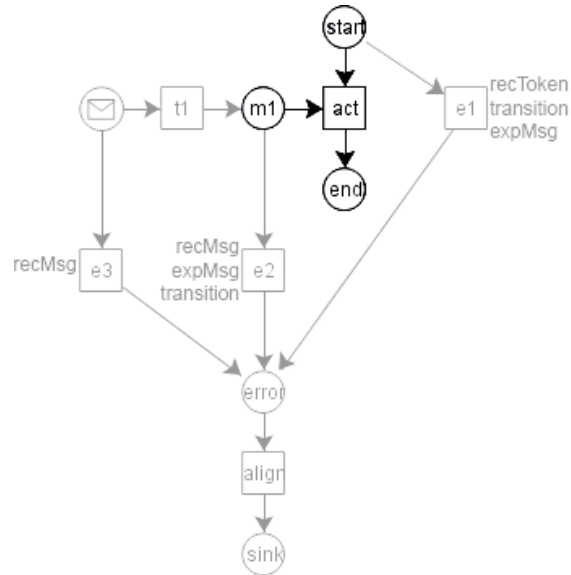


Fig. 4: Collecting problem information

- **transition**, the name of the transition for which the **recToken** is required to enable it (**act** in the example)
- **expMsg**, the data type of the message expected by the transition (in this example the data type the place **m1** may contain)

*II. When a known received message is not moving* (Place **m1** in this example.) In this situation, transition **e2** will send information about this occurrence to the **align** transition. It will send the following information:

- **recMsg**, the data type of the received message (in this example the data type which place **m1** may contain, which of course equals the data type of the message, because otherwise the message would not be in this place)
- **expMsg**, the data type of the expected message (also the data type place **m1** may contain)
- **transition**, the name of the transition which is connected to the message place (**act** in the example)

*III. When an unknown message is received* (The message place in this example.) When this happens, transition **e3** will send information about this occurrence to the **align** transition. It will send the following information:

- **recMsg**, the data type of the received message

All the collected data is combined and used for the selection of an appropriate policy. Furthermore, this data will be used as parameters for the operations within the policy, as will be discussed next.

## 5.2 Alignment policies

An alignment policy is an ordered set of primitive or advanced operations. In this approach, an agent has a set of policies in its knowledge-base from which it can choose when problems arise with the execution of the behavior within an interaction. Table 2 shows two example policies.

Table 2: Examples of alignment policies

Policy #	Operations
1	<code>parallelization(transition,seqTransition)</code>
2	<code>notReceiveMessage(transition,expMsg)</code> <code>specialization(transition,Receive expMsg, Receive recMsg)</code> <code>receiveMessage(Receive expMsg,expMsg)</code> <code>receiveMessage(Receive recMsg,recMsg)</code>

Policy 1 puts two transitions in parallel. Policy 2 makes a specialization of a transition, which expects a certain message, into two transitions, one that expects the original expected message, and the other that expects the actual received message. Instead of giving fixed names as parameters for the operations, variables can be used. Also a combination of fixed names and variables is possible. When the policy is actually executed, these variables will automatically be replaced by the appropriate values for that situation. The advantage is that in this way a more general policy can be defined, that is not only restricted to a given name of the transition or message types. Table 3 shows the list of variables what can be used when defining the parameters, and with what they will be replaced.

Table 3: Variables for defining parameters

Variable	Will be replaced with
<code>transition</code>	the name of the transition
<code>seqTransition</code>	the name of the transition which is sequential placed after <i>transition</i>
<code>parTransition</code>	the name of the transition which is parallel with <i>transitions</i>
<code>specTransition</code>	the name of the transition which is mutual exclusive to <i>transition</i> , and originates from the same specialization
<code>expMsg</code>	the data type of the expected message
<code>recMsg</code>	the data type of the received message

In the current state of research, policies are manually identified by the agent-system designer. However, the agents can learn by repetitive experience when to apply a certain policy, and select the appropriate policy automatically. One of the selection methods that was implemented is presented in the next section. An early observation in the discovery of potentially usable policies has shown that an agent does not need a large number of policies to solve most of the occurred problems in behavior executions. An intuition is that there is rather small and manageable number of alignment patterns.

### 5.3 Implementation

This subsection describes the implementation of the policy selection and alignment mechanism used for experiments. In this system, an agent selects a policy using a base of training examples which contain both the problem information and the policy which was chosen at that time. A machine learning technique is used to generalize over this data, and also to enable the agent to choose a policy based on the previous experience. How an agent will choose an alignment policy (or if it will choose one at all) depends on multiple factors. The factors used in the implementation are: problem information, beliefs about the agents interacting with, and the willingness to change its own behavior.

*Problem information* Most of the time, a problem will occur when the agent is not receiving the message it is expecting. It can also be the case that the agent did not receive a message at all. If it did receive a message, but the wrong one, the type of received message and other information about the problem are used as attributes for selecting the proper alignment policy.

*Beliefs about the agents interacting with* Beliefs about the other agents can be of importance when choosing an alignment policy. For example, when one agent completely trusts the other agents, it might be willing to make more changes in its behavior than when it distrusts the other agents.

*Willingness to change behavior* When an agent has very advanced and fine-tuned behaviors, it is not considered good practice to radically change the behaviors because of a single exceptional interaction. On the other hand, when the behavior of the agent is still very primitive, changing it more often could be a good way to acquire new experience. In this way, when an agent gets “older”, and the behaviors are based on more experience, the willingness to change its behavior will decrease. This approach can be compared with the way humans learn, or with the decrease of the learning rate over time when training a neural network.

The process of alignment is shown in figure 5 as a finite state automaton. The policy selection method (in this particular implementation the decision algorithm based on machine learning) uses the collected problem information and values from the belief base (like trust and willingness) to choose an alignment policy. However, it can be the case that the agent cannot find an appropriate alignment

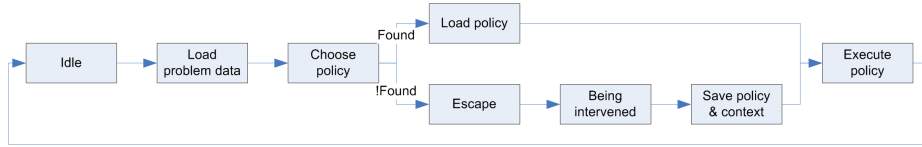


Fig. 5: The process of alignment

policy for this specific problem. In this case, the agent can trigger escape mode, and ask a human to indicate a policy. After this policy has been chosen by the person, the agent needs to memorize the number of the chosen policy as well as the context. In this way, the agent has gained new experience when to apply a certain policy. More information about the concept of escape mode can be found in [14]. After the alignment policy is chosen, either by the decision algorithm, or by a human, this policy will be executed to adapt the Behavior Net.

The machine learning technique used in this implementation is based on artificial neural networks [12]. This is an obvious choice, given the simplicity and robustness of these classification systems, and also given the availability of well proved open-source implementations (in this system, the WEKA3 package is used [20]). However, other learning or classification methods can also be used. With neural networks not only the selected policy is returned, but also the activation levels of all policies. Therefore, the certainty of the choice can be known. The activation level of the selected policy is compared with the activation level of the runner-up (the second-best policy). The activation level of the best policy should be factor  $x$  (with  $x > 1$ ) higher than the runner-up, otherwise, the agent will go into escape mode. In the implementation  $x = 2$  has been used. This ensures that the agent will not execute an alignment policy when suitability for the current problem is questionable. When unsure, it will ask the human for advice. An example of the training data used (two learning patterns) to train the neural network is shown in table 4. The policy # row in the table represents the output of the network, which has one output neuron for each known policy, and the rest of the rows are representing the inputs. The neural network has to be rebuilt and retrained when a new policy is added to the known pool.

## 6 Experiments and results

Experiments have been conducted to test the applicability of the alignment and policy choosing mechanism. This section provides an illustrative example of such an experiment. In this experiment, as shown in figure 6, a buyer and a seller already agreed on the product the buyer wants to buy, but as seen in the figure, they have different views (formalized as behaviors) in what order the delivery and the payment should occur. In this experiment, it is presumed that the behavior of the buyer is very advanced, and thus he is not willing to easily change his behavior. On the other hand, the seller is inexperienced and

Table 4: Example of training data

Training pattern #	1	2	...
<b>transition</b>	Receive money	Receive money	...
<b>seqTransition</b>	Send product		...
<b>parTransition</b>		Send product	...
<b>specTransition</b>			...
<b>recToken</b>	yes	yes	...
<b>recMsg</b>		product	...
<b>expMsg</b>	money	money	...
<b>trust</b>	1	1	...
<b>willingness</b>	1	1	...
<b>policy #</b>	1	2	...

his behavior is still primitive, hence we are looking at the problem how the seller can align its behavior with the buyer, presuming that the seller trusts the buyer.

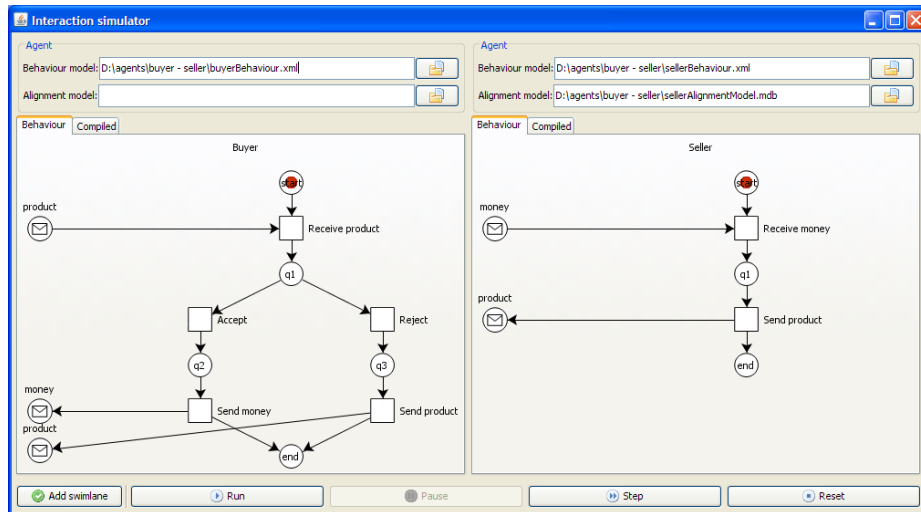


Fig. 6: Behaviors of buyer and seller

Within the experiment, when the interaction started, it immediately deadlocked; the buyer was waiting for the product, and the seller was waiting for the money. The seller collected the information in the way explained in section 5.1. Besides the name of the transition, also **seqTransition**, **parTransition** and **specTransition** (as explained in section 5.2) were looked up. The collected problem information is shown in table 5 (a).

This information was used by the neural network of the seller to select the appropriate alignment policy. Due to the previously trained neural network, via

Table 5: Collected problem information

Information	Value	Information	Value
transition	Receive money	transition	Receive money
seqTransition	Send product	seqTransition	
parTransition		parTransition	Send product
specTransition		specTransition	
recToken	yes	recToken	yes
recMsg		recMsg	Product
expMsg	Money	expMsg	Money

(a) First problem

(b) Second problem

repeated escapes to humans in previous experiments, this led to the execution of policy 1 from table 2. The seller transformed its behavior by placing its two transitions in parallel. The result is shown in figure 7 (a).

Still, the two behaviors were not aligned. The buyer rejected the product, and sent it back. Consequently, the seller did not have the appropriate behavior to handle this, as the seller was expecting the payment. After the buyer sent the product back, the seller detected the problem and collected the data as shown in table 5.1 (b).

The neural network for selecting an alignment policy was used again, and due to previous training policy 2 of table 2 was chosen. The seller divided the transition *receive money* in two mutual exclusive transitions: *receive money* and *receive product*. This changed the behavior as shown in figure 7 (a) into the behavior as shown in figure 7 (b). The behaviors of the buyer (figure 6 (a)) and the behavior of the seller (figure 7 (b)) were now matching. Because the seller already knew two suitable alignment policies, and also when to select them, the interaction completed successfully, despite their initial conflicting behaviors.

The conducted experiments, as the one discussed above, have shown promising results concerning the applicability of the proposed alignment mechanism. The necessary human assistance by the use of the alignment mechanism has significantly decreased as the agents are better able to solve conflicting behaviors during the interaction. Human intervention reinforced certain successful policies in a given context, and also contributed with completely new policies. The net result, revealed during the experiments, is that the simulated business processes become increasingly automated.

## 7 Discussion and Conclusions

Currently, there are several limitations with alignment policies, which need to be studied in future research. Firstly, an alignment policy is limited to a certain scope, only the transitions defined by the problem information can be used as variables for the operations of the alignment policy when defining an alignment policy. Ways to overcome this limitation have to be investigated. Secondly, the problem data used for parameters of alignment policies can only contain one of

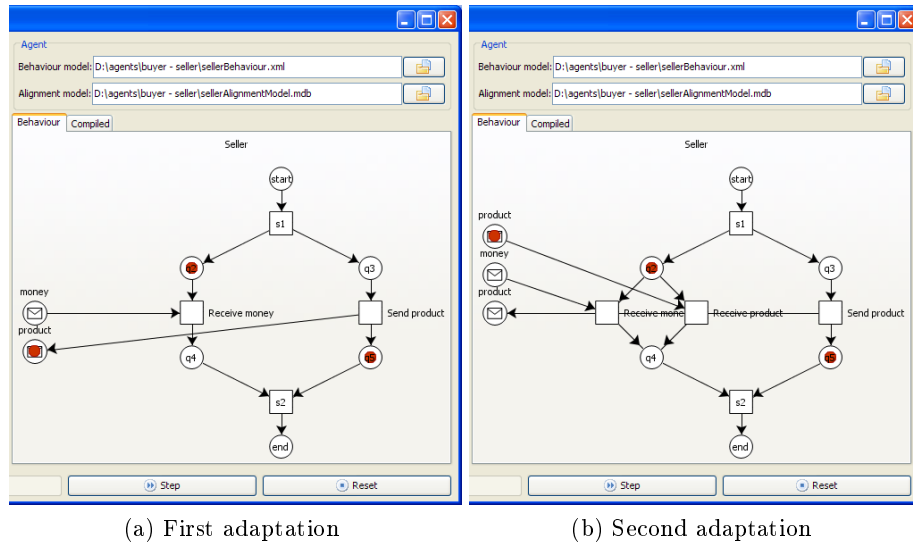


Fig. 7: Adapted behaviors of seller

the transitions which are in parallel with the transition where the problem is. It could however be the case that there are three transitions in parallel. Using the variable `parTransition` in defining the alignment policy would give a non-deterministic execution of the policy. This issue does not only affect the problem data, but also the used neural network, as it is not suitable for having multiple parallel transitions as input. For this reason, other machine learning techniques like decision trees have to be investigated.

In conclusion, it is possible to describe a policy for alignment that can be applied when the interaction beliefs of two or more interacting agents are not matching. An extension of Petri Nets has been used to capture the intended interaction of an agent in a formal way. Furthermore, a mechanism based on a neural network that chooses an appropriate alignment policy with the collected problem information was implemented. When this mechanism fails to choose an alignment policy, a method based on human intervention was described, which can teach the agent new ways of alignment. As the experiments have shown, this approach enables agent behavior execution to be more reliable and necessitates less human intervention in terms of alignment.

## References

1. W. M. P. Aalst, van der. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21-66, 1998.
2. W. M. P. Aalst, van der. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335-367, 1999.

3. W. M. P. Aalst, van der. Workflow verification: Finding control-flow errors using petri-net-based techniques. *BPMt: Models, Techniques, and Empirical Studies*, 1806:161-183, 2000.
4. P. Chrzastowski-Wachtel, B. Benatallah, R. Hamadi, M. O'Dell, and A. Susanto. A top-down petri net-based approach for dynamic workflow modeling. *LNCS*, 2678:335-353, 2003.
5. F. Dignum. Agent communication and cooperative information agents. In *Proc. of CIA '00*, pages 191-207, 2000.
6. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proc. of COCS '95*, pages 10-21, New York, NY, USA, 1995. ACM Press.
7. C. A. Ellis and K. Keddara. A workflow change is a workflow. *LNCS*, 1806:201-217, 2000.
8. R. Hamadi and B. Benatallah. Recovery nets: Towards self-adaptive workflow systems. *LNCS*, 3306:439-453, 2004.
9. D. Harel and R. Marelly. Specifying and executing behavioral requirements: The play-in/play-out approach. Technical Report MSC01-15, The Weizmann Institute of Science, 2001.
10. K. Jensen. An introduction to the theoretical aspects of coloured petri nets. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 230-272, London, UK, 1994. Springer-Verlag.
11. G. G. Meyer and N. B. Szirbik. Anticipatory alignment mechanisms for behavioural learning in multi agent systems. *LNAI*, 4520:(to appear), 2007.
12. T. M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.
13. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik Bonn, 1962.
14. G. B. Roest and N. B. Szirbik. Intervention and escape mode. In *Proc. of AOSE'06*, pages 109-120, 2006.
15. M. Rovatsos, F. Fischer, and G. Weiss. An integrated framework for adaptive reasoning about conversation patterns. In *Proc. of AAMAS '05*, pages 1123-1124, 2005.
16. K. Salimifard and M. Wright. Petri net-based modelling of workflow systems: An overview. *European Journal of Operational Research*, 134(3):664-676, 2001.
17. M. Stuit and N. Szirbik. Modelling and executing complex and dynamic business processes by reification of agent interactions. *LNAI*, 4457:(to appear), 2007.
18. M. Stuit, N. Szirbik, and C. de Snoo. Interaction beliefs: a way to understand emergent organisational behaviour. In *Proc. of ICEIS'07*, 2007.
19. R. Thimm and T. Will. A state of the art analysis of workflow modelling and agent-based information systems. Technical report, University of Trier, 2004.
20. I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition, 2005.