Int. J. Appl. Math. Comput. Sci., 2015, Vol. 25, No. 3, 439–454
DOI: 10.1515/amcs-2015-0033

A SITUATION–BASED MULTI–AGENT ARCHITECTURE FOR HANDLING MISUNDERSTANDINGS IN INTERACTIONS

THAO PHUONG PHAM [a], MOURAD RABAH [a,*], PASCAL ESTRAILLIER [a]

[a]L3i Laboratory
University of La Rochelle, Avenue Michel Crépeau, 17042 La Rochelle Cedex 1, France
e-mail: {phuong-thao.pham,mourad.rabah,pascal.estrailler}@univ-lr.fr

During interactions, system actors may face up misunderstandings when their local states contain inconsistent data about the same fact. Misunderstandings in interactions are likely to reduce interactivity performances (deviation or deadlock) or even affect overall system behavior. In this paper, we characterize misunderstandings in interactions between system actors (that may be human users or system agents) in interactive adaptive systems. To deal with such misunderstandings and ensure state consistency, we present an agent-based architecture and a scenario structuring approach. The system includes several agents devoted to scenario unfolding, plot adaptation and consistency management. Scenario structuring is based on the notion of a *situation* that is an elementary building block dividing the interactions between systems' actors into contextual scenes. This pattern supports not only scenario execution but consistency management as well. In order to organize and control interactions, the *situation* contextualizes interactions and activity of the system's actors. It also includes prevention and tolerance agent-based mechanisms to deal with the misunderstandings and their causes. We validate our consistency management mechanisms using Uppaal simulation and provide some experimental results to show the effectiveness of our approach on an online distance learning case study.

Keywords: interactive adaptive systems, misunderstandings in interactions, situation structuring, consistency management.

1. Introduction

In interactive systems, such as games and simulators, users and internal agents can modify system content and progress in real time through input adjustments. Interactive systems may adapt system execution not only to the users' actions, but also to their profile and behavior, making these systems adaptive. In order to perform the adaptation, the system must capture users' behavior from their interactions. Then, according to the system's and the designer's logics, it adjusts its execution to what it perceives of the user's logic. Due to the unpredictability of the user's actions, the execution process of an interactive system is also unpredictable.

One of the important problems in interactive systems is a potential misunderstanding between the users and the system and, more generally, between the system's actors, virtual system agents or physical human users. If the system does not capture correctly or confuses the user's actions, or if the users do not understand what the system expects, this may lead to an erroneous interpretation of their behavior and an erroneous adaptation of system execution. This misunderstanding may concern user–system interactions, but it may also appear in any kind of interaction between any system's actors. It may be due to the actors' incomplete data or the non-determinism of actors' behavior and cause an interaction deadlock or an application failure.

In our recent work (Pham *et al.*, 2011; Trillaud *et al.*, 2012), we have defined the *misunderstanding in interaction* as follows: *When two or more system's actors have incoherent data in their local visions about the same fact f and these data are used during their interaction, this may cause an interaction deviation from the planned scenario*. An actor may be a human user or a virtual system agent. The local vision is the actor's own knowledge about its external world (a virtual environment, the system's resources, etc.), its relations with others actors (a subset of their states) and its own profile (an internal state). Thus, our work focuses on the management of the consistency between the actors and the

*Corresponding author

system and between the actors' local visions in order to handle potential misunderstandings in interactions.

To handle misunderstandings, we propose to structure application execution into interaction sequences called *situations*, including misunderstanding prevention and tolerance mechanisms. Each *situation* corresponds to a contextual resource-centered sequence of activities and events, and is characterized by preconditions and postconditions. That allows the system to control the execution and to establish the causal links between the *situations*. This model confines the actor's interactions in a given context in order to control their execution and manage the consistency. Consistency handling mechanisms are inspired by techniques from the dependability domain (Laprie *et al*., 2004), since there is an analogy between misunderstandings in interactive systems and errors handled in fault tolerant systems. To use this situation-based scenario structuring, we conceived an agent-based system architecture (Pham *et al*., 2013) that allows the system's scenario to be represented by *situation* combination and uses agent components to control the system's execution, scenario unfolding and consistency management. The architecture includes two kinds of agents: (i) those representing the system's actors that either interact with the user(s) or with each other (as no-player characters in games) and (ii) the system's control agents involved in system execution. Some of these control agents are involved in the consistency management by tracking and handling misunderstandings in interactions between the system's actors.

We evaluated our consistency management approach and mechanisms through simulation. First, we used the Uppaal[1] model to validate the consistency management mechanisms inside situation blocks from the actors' interactions point of view. This allows assessing the structural properties of our overall execution model for consistency management during interactions. Then, we performed experimentation using the GAMA simulation platform[2] on an online distance learning case study in order to show the effectiveness of misunderstanding in interaction handling.

Section 2 presents the related work in the domain of interactive systems architectures for consistency support. In Sections 3 and 4 we formalize misunderstandings in interaction from our point and view and present their handling mechanisms. Sections 5 and 6 describe our system's architecture and situation-based structuring. The consistency management model is validated using Uppaal in Section 7 and its effectiveness in Section 8. Section 9 concludes the paper.

[1]`http://www.uppaal.org/.`

[2]`https://code.google.com/p/gama-platform/.`

2. Related work

In the recent research, we can find several works dealing with the user–system dialogue where the communication is done through a real human language (Karsenty and Botherel, 2005; Lopez-Cozar *et al*., 2010; Rapaport, 2003). According to Rapaport (2003), negotiation is the key to understanding: a cognitive agent understands by negotiating with the interlocutor or by hypothesizing the meaning of an unknown word from the context. This agent can negotiate with itself on something external by comparing its perception and its internal knowledge in order to correct its own misunderstandings. Other works propose to use confidence scores to measure the reliability of each word in a recognized sentence (Jiang, 2005). Besides, Lopez-Cozar *et al*. (2010) proposed to implement a frame correction module, independent of the speech recognizer. This module corrects misunderstandings in a sentence, caused by errors in speech recognition, by replacing the incorrect frame with an adequate one. Karsenty and Botherel (2005) applied adaptable and adaptive transparency strategies in TRAVELS to help the users to understand and react appropriately to system rejections and misunderstandings. The ability of making the system's interpretations explicit and informing the users on how to correct misunderstandings are two ways to help the users handle them. This strategy is very effective in misunderstanding detection and raises the rate of appropriate user responses after system rejections. All of these works deal with the problem in the speech dialogue where misunderstandings are more frequent. But misunderstanding can be found in other forms of interactions like actions, gestures, etc.

Our purpose is to define how we can treat misunderstandings between the actors themselves besides user–system misunderstandings. It is not easy to recognize such a class of misunderstandings. In the dependability domain (Laprie *et al*., 2004), we find the inconsistency problem between systems and operators. The *automation surprise* is an inconsistency error occurring when the system behaves differently from its operators' expectations (Combefis and Pecheur, 2009). It may be due to a mismatch between the actual system behavior and the operator's mental model of that behavior (King, 2011), and it can lead to *mode confusion* and even to critical failures. In general, misunderstandings come from the gap between the user's and the designer's logic, all along action planning between the actors.

Many works, particularly in interactive storytelling, have been conducted to solve the mismatch between the user's behaviors and system logic (Magerko and Laird, 2004; Young *et al*., 2004; Barber and Kudenko, 2008; Paul *et al*., 2011; Silva *et al*., 2003) by predicting the user's future actions and detecting the invalid ones that deviate the execution from the planned objectives.

The minemsis architecture (Young *et al.*, 2004) uses a mediator to detect when the player is attempting to execute an action that may threaten the integrity of the story plan. This approach does not aim to alternate the story but to incorporate unplanned actions or avoid them. Besides, IDA (Magerko and Laird, 2004) introduces a specific agent called *Director* to maintain the story line. It also predicts the player's actions to determine if they may endanger the plot and to try to avoid them before they happen. If there is a problem, *Director agent* can alter the world context by changing any accessible parameter in the application world's state.

In PAPOUS (Silva *et al.*, 2003), the story to be told is organized into levels and each level consists of a set of *StoryBits* characterized by different properties, characters and events. PAPOUS manages the inconsistency between a virtual storyteller and the audience decisions in a simple way: the storyteller ignores missing or inadequate inputs and chooses the next *StoryBit* according to a previously narrated one or to story desirability.

In an interactive system, we try to build a less constraining environment for the users' actions. But the higher the degree of freedom the application allows, the more easily the users' actions may deviate from the planned scenario and the more easily misunderstandings may occur. GADIN (Barber and Kudenko, 2008) is an interactive text-based system using story goal regeneration mechanisms in order to change the game goals into new ones when the player's actions move too far away from the planned goal state.

MIST (Paul *et al.*, 2011) has yet another approach to deal with the scenario deviation problem. It attempts to repair stories that are already in progress when they are invalidated by unforeseen events. The preemptive detection of invalid plan steps is done in advance, some close steps before the point of failure. Once an invalid step is detected, the story management tries to substitute the story plan by a consistent one.

In general, prediction approaches are costly. For instance, the short-term player behavior modeling module implanted in by Magerko and Laird (2004) creates an entire copy of the whole world's state. This module simulates the world changes according to the user's actions. This kind of approach does not seem well suited to real-time interactive systems, nor to systems where the user's behavior cannot be easily modeled by a set of rules.

Our approach focuses on software and a component design model integrating misunderstanding prevention and handling mechanisms. It relies on three points: the system observes and analyses the users' and the system's states, detects the misunderstandings or their consequences and acts to keep the consistency between the actors before and at the end of interaction sequences. These mechanisms do not try to predict users behavior but take into account the users' states to adapt the system's execution in order to avoid misunderstandings between the system's actors.

3. Misunderstandings in interactions

We define a *context* as *a set of pieces of information that can be used to characterize the situation of an entity* (Dey, 2001), where an *entity* is an actor involved in interactions and may be represented by a system agent. A *situation* is an interaction sequence between several actors or agents in a shared fixed context. Thus, an interaction during system execution is carried out between at least two actors within a common context.

3.1. Context and the actor's local vision. The context is related to the actor's activities (Hommel *et al.*, 2000; Dourish, 2004; Picard and Estraillier, 2010). There is interdependence between the common context and the actors located in this context. An actor performs its activities depending on the current situation and the available contextual information. Each actor has to observe and to perceive the world, to interpret it with its own logic, to combine the new information with its existing knowledge in order to obtain its own contextual vision and to update its current knowledge as shown in Fig. 1. This knowledge is called the actor's *local vision*.

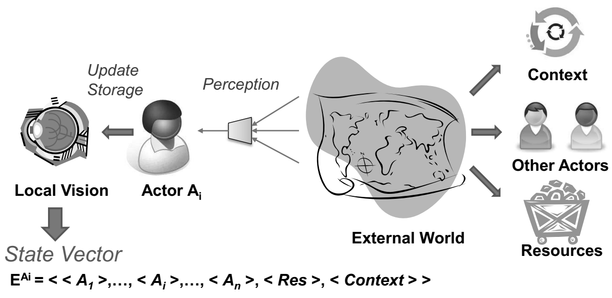

Fig. 1. From the external world to the actor's local vision.

The *local vision* is the actor's own knowledge about the external world (the environment, resources etc.), the relations with other actors (other actors' states), and its own profile (internal state). For an actor A_i, the local vision is a state vector E^{A_i} (as depicted in Fig. 2). This state vector is hierarchical and divided into sub-vectors corresponding to states of every system actor $A_1, A_2, \ldots, A_n$ (with n the number of the system's actors), including A_i's own state vector, the resource state vector Res and the context state vector $Context$. Each sub-vector can be divided again into lower level sub-vectors related to the actors' component entities. For instance, the element A_1 may be a sub-vector composed of m elements that can be other smaller sub-vectors or a final data value corresponding to an end-level attribute: $\langle A_1\rangle = \langle A_1^1, A_2^1, \dots, A_m^1\rangle$. The division into sub-vectors is done until the chosen granularity is reached.

$$\mathbf{E^{A_i}} = \langle\langle \mathbf{A_1}\rangle \dots \langle \mathbf{A_i}\rangle \dots \langle \mathbf{A_n}\rangle, \langle \mathbf{Res}\rangle, \langle \mathbf{Context}\rangle\rangle$$

$$\begin{aligned}
&\langle A_1\rangle = \langle A_1^1, A_2^1, \dots, A_m^1\rangle\\
&\qquad\langle A_1^1\rangle = \langle A_1^{11}, A_2^{11}, \dots, A_m^{11}\rangle\\
&\qquad\vdots\\
&\qquad\langle A_m^1\rangle = \langle A_1^{1m}, A_2^{1m}, \dots, A_m^{1m}\rangle\\
&\langle A_i\rangle = \langle A_1^i, A_2^i, \dots, A_m^i\rangle\\
&\vdots\\
&\langle A_n\rangle = \langle A_1^n, A_2^n, \dots, A_m^n\rangle\\
&\qquad\langle A_1^n\rangle = \langle A_1^{n1}, A_2^{n1}, \dots, A_m^{n1}\rangle\\
&\qquad\vdots\\
&\qquad\langle A_m^n\rangle = \langle A_1^{nm}, A_2^{nm}, \dots, A_m^{nm}\rangle\\
&\langle Res\rangle = \langle R_1, R_2, \dots, R_p\rangle\\
&\langle Context\rangle = \langle C_1, C_2, \dots, C_q\rangle
\end{aligned}$$

Fig. 2. Local vision for an actor A_i as a state vector.

Each state vector contains data elements representing the knowledge perceived by the corresponding actor and characterizing the involved entities. Moreover, it must be stressed that all sub-state vectors $\langle A_1\rangle, \langle A_2\rangle, \dots, \langle A_n\rangle$ except $\langle A_i\rangle$ are the results of A_i's perception, so they are partial. This means that these vectors do not contain the whole real state of the corresponding actors $A_1, A_2, \dots, A_n$. However, the vector $\langle A_i\rangle$ contains its own state that is supposed to be complete.

3.2. Definition of the misunderstanding. The local vision allows an actor to be able to interact with the others with a strategy and coordination. But the local vision is not static: it evolves during interaction sequences. The perceived data are not always identical between different actors due to their differing perceptions and local environments. Hence, their local visions may become inconsistent during interactions. This may lead to a different interpretation of the same fact (a sentence, an action, a state, etc.). If the actors use such inconsistent data in future interactions, a misunderstanding may arise.

We give the following definition from our previous work (Pham *et al.*, 2011): *Two actors are in a misunderstanding state when (i) they are in interaction with each other, (ii) there are incoherent data in their local visions about the same fact, and (iii) these data are used during the interaction. A fact is considered as objective data or an absolute reference to the system's actors or resource states*. If we consider the interactions between two actors as acts of language, a misunderstanding can be observed when two actors think that they are talking about the same thing whereas they actually are talking about different subjects (Rapaport, 2003).

We formalize this definition as follows: Let two actors A and B interact in the presence of the fact f from

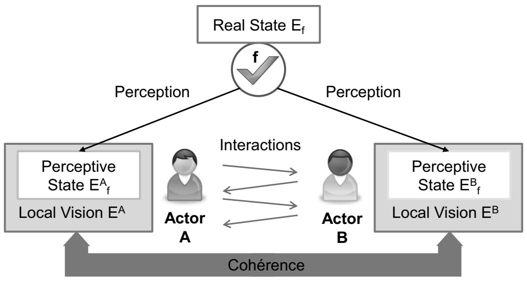

Fig. 3. Actors A and B perceive the fact f in an interaction.

the external world. State vectors E^A and E^B are the local visions of A and B. The atom E_f^{real} is the absolute reference to f. The knowledge perceived by A and B about the fact f is represented by the atoms E_f^A and E_f^B. From the point of view of the state vector, these atoms can be a sub-state vector or a data value element of E^A and E^B: $E_f^A \subset E^A$ and $E_f^B \subset E^B$

The perception can be seen as an internal action that cannot be observed by other actors and corresponding to the local vision updates after having performed or observed an action. The perception of A and B about f in Fig. 3 is represented by the following formulas:

$$A : E_f^{real} \longrightarrow E_f^A,$$

$$B : E_f^{real} \longrightarrow E_f^B.$$

Misunderstanding in interactions appears when E_f^A is different from E_f^B: $E_f^A \neq E_f^B$. Hence, the local visions of A and B are incoherent. The distance D_f^{AB} measures the difference level between E_f^A and E_f^B: $D_f^{AB} = |E_f^A, E_f^B|$.

The ideal misunderstanding-free situation is when the perception of A and B of a fact f is identical, i.e., $E_f^A = E_f^B$, and the distance $D_f^{AB} = \emptyset$.

3.3. Elements that cause misunderstandings. Misunderstandings in interaction have various causes:

Different references. This happens when interacting actors have different contexts. The interactions between the actors are carried out under a concrete context that influences their behavior. The actors located in different reference worlds will consider different things. For instance, the word "bug" is a kind of insect but in the computer world it refers to an error or a fault that produces an incorrect program execution. If the interaction context differs, actors' local visions will not be synchronized and misunderstanding conditions may be established.

Different logics. The actions of an actor depend on its own logic and deduction rules. For example, two actors interact about the identity of some "old person". For A, an old person means a person over 60 years: old$(x) \Rightarrow$ age$(x)>$60 year. For B, "old person" means the oldest known person: old$(x) \Rightarrow \forall y$ age$(x) \geq$ age(y). If B asks A for an old person, B will expect the oldest person, whereas A will just provide someone old but not especially the oldest one. If B asks again, A may provide a different answer, and A and B will be in a misunderstanding since each actor has its own logic.

Semantic ambiguity. A wrong interpretation during interactions may cause a different perception. Semantic is internal (Rapaport, 2003): the external world is reflected subjectively in the actor's "mind" that creates its own narrow knowledge. It is obvious that an actor can interpret a fact as correct or wrong because of the lack of information or the imperfection of the observation. For instance, in an e-learning application, a camera has to check the presence of a student. Due to a limited camera scope, a student may be warned because of his/her absence, whereas he/she is still there but out of the camera scope.

3.4. Misunderstanding consequences.

Misunderstandings in interactions have also various outcomes:

Interaction deviation. The interaction chain between two actors diverges from the planned scenario. An actor may estimate incorrectly the state of its interaction partners because of misunderstandings. As result, the actor will make a wrong decision based on the wrong observed state of its partners. Instead of an appropriate action according to the plan, the actor's behavior will diverge from its normal logic and from the logic of other interacting actors.

Interaction deadlock. This problem arises when a misunderstanding is revealed and the actors get stuck in the middle of an interaction sequence. In this case, an actor receives an answer or a demand that it does not expect, because it is expecting some other reaction. The interaction sequence will be broken. Both actors (or at least one of them) do not know what to do any more.

Propagation. If misunderstanding is not detected or revealed, it can be propagated along the scenario and the execution of the application. Inconsistent data are not treated and are kept for future interactions. Furthermore, misunderstanding severity may increase.

4. How to manage misunderstandings

The aim of misunderstanding management is to avoid misunderstanding occurrence as much as possible. If it happens anyway, it should be eliminated. Moreover, before a misunderstanding is detected, the interactions between two actors may have already deviated from the planned scenario. We must intervene to synchronize their data and behavior.

4.1. Necessary occurrence conditions.

A misunderstanding may occur during an interaction sequence if the following conditions are fulfilled:

C1 *Actors' presence*: at least two actors participate in the interaction sequence. The misunderstanding occurs only when the actors interact with each other.

C2 *Inconsistency of local data*: the knowledge about the same fact f is totally different or contains a part of different data. There is a data inconsistency in the actor's local visions.

C3 *Data sharing*: inconsistent data are used as shared information or common contents between the actors during their interactions.

If the three conditions are met, a misunderstanding will occur. Otherwise, it is not possible. For instance, if two actors have inconsistent data but they never interact with each other (*C2* is satisfied but not *C1*), a misunderstanding will not arise. Moreover, the actors may have different data about the same fact, but if they do not use it as shared data during the interaction (*C1* and *C2* but not *C3*), they will not face misunderstanding.

4.2. Approaches.

We classify the misunderstanding management into four classes:

Ignoring. If a misunderstanding is minor, we can just ignore it. This is similar to the ostrich algorithm (Tanenbaum and Woodhull, 2006) in deadlock treatment.

Prevention. Misunderstanding occurrence can be prevented by denying one of the three necessary conditions mentioned previously. If one condition is missing, we remove the possibility of misunderstanding occurrence. The actors' local vision should not contain inconsistent data. Ideally, their knowledge should be identical and coherent all along an interaction. Hence, the actors' data consistency should be checked after each interaction sequence and synchronized if needed. Moreover, shared data should be identified before interactions begin. An explicit declaration of the shared data allows the actors and system control agents to check the consistency of the data before the actors use them. If an inconsistency is detected, either the data are synchronized between the actors concerned or isolated to avoid its use during interactions.

Tolerance. It aims to detect a latent (potential) misunderstanding during an interaction and resolve it when it is revealed (i.e., when it becomes effective). Misunderstandings are similar to the threats (fault, error, failure) affecting system service in the dependability domain (Laprie *et al.*, 2004). For instance, a byzantine or inconsistent failure happens when some or all the system users perceive differently service correctness. *Automation surprise* and *mode confusion* occur when the system behaves differently than its users expect (King, 2011). These examples show the effects of different actors' perceptions. The principles of misunderstanding tolerance are similar to fault tolerance with error detection and system recovery. The implemented mechanisms track down the system service deviation, and put the system into a degraded mode or restoration. We suggest adapting fault tolerant techniques to misunderstanding management in interactive applications by (i) regularly checking the actors' local vision data in order to detect and eliminate both latent (potential) and revealed (effective) misunderstandings, if possible, before interaction deadlock; (ii) resolving interaction deviation or deadlock by appropriate handling mechanism: either the system rollbacks to a misunderstanding-free state in order to retry the last interaction or it goes on but with reinforcement actions synchronizing the actors' threads, in the most transparent manner for the user and with respect to the designer's storyline.

Removal. It refers to misunderstanding detection and elimination. It is mainly achieved using regular coherency control to detect misunderstandings and data synchronization to eliminate them.

4.3. Handling solution. To handle misunderstandings in interactions, we explore two directions: the adaptability structure and fault tolerance. Our solution relies on three points: (i) we build a robust system architecture with specific additional agent components that are in charge of misunderstandings in interaction management; (ii) we organize the scenario and system execution using *situation* blocks that are not only the basic narrative construction elements but also the execution patterns that contextually confine interactions; (iii) we integrate consistency management into situations' dynamic execution, including data synchronization, misunderstanding detection and treatment inspired and adapted from fault tolerance techniques. Hence, we structurally avoid a part of potential misunderstandings before each interaction sequence start and guarantee a misunderstanding-free state at its end.

In the following sections, we present and detail these three contributions of our research.

5. Proposed agent-based architecture

Several architecture models for interactive systems have been proposed according to the specific purpose of each work. We chose the approach of a multi-agent system of Sehaba *et al.* (2005) as a starting point to build our model. The advantage of this approach is that each agent can be organized and work autonomously and strategically. We define four system control agents (Fig. 4 shows the overall architecture):

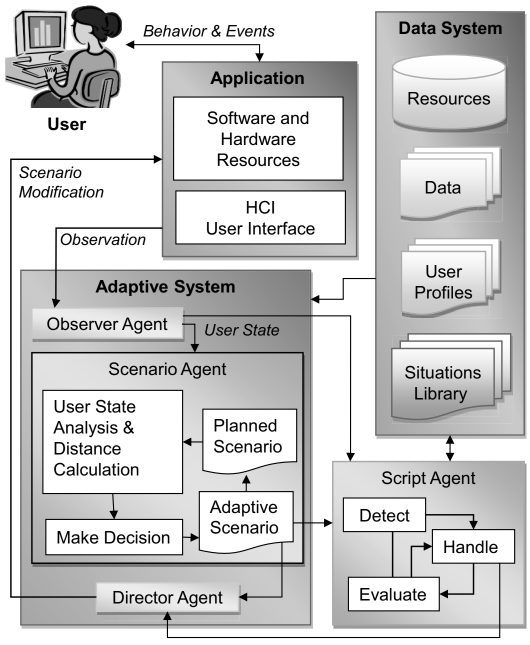

Fig. 4. General agent-based architecture for an interactive system.

- *Observer agent*: It observes the user's behavior and state, formalizes, normalizes and transfers them to *Scenario agent*.

- *Scenario agent*: It makes decisions about the scenario orientation according to the user's state, planned scenario and permanent objective defined by the designer. This agent tries to find the best way to orientate application execution. It takes charge of the library of *situations* planned by the designer. The *situations* (defined in Section 6) represent scenario components and are the interaction and the activity sequences that can take place in the application as, for instance, all possible scenes in a theatre play.

- *Director agent*: This agent receives the decision taken by *Scenario agent.* It takes in charge the production of the adaptive scenario and realizes a modification, an answer or an action adapted to the users.

- *Script agent*: Its task is to track and handle the inconsistency in three steps: (i) *detection*:

detect, confine or partition the inconsistency between situation actors in order to identify the causes of a misunderstanding; (ii) *treatment*: apply the handling mechanism or strategy to remove the inconsistency and to correct the deflected state that causes the incoherence; (iii) *evaluation*: estimate the efficiency of the treatments in order to improve the employed mechanism for the next time.

6. Situation-based scenario

6.1. Interactive storytelling approach. Interactive storytelling is the unfolding of a story that the user's decisions impact (Champagnat *et al.*, 2010; Lebowitz and Klug, 2011). It defines how to generate scenarios which are both interesting and coherent. We assume that the interactions in an interactive application can be organized, strongly or weakly, as a story scenario. That allows us to adapt ideas from the storytelling domain to organize interactions.

The scenario in interactive storytelling is represented by a series of actions/events linked together by cause and effect (Karlsson *et al.*, 2006), by ordered link (Magerko and Laird, 2004; Silva *et al.*, 2003) or by hierarchical task network planning (Paul *et al.*, 2011), where each task is decomposed into subtasks until primitive actions. But these scenario structuring approaches are not suited to build complex interaction sequences where the user's actions are free, non-predictable and depending on a great amount of context data. Hence, we propose the notion of a *situation* that can be seen as a scene encompassing not only interactions execution but also interaction management and resource use. *Situations* are the basic narrative elements that facilitate interaction planning and management by characterizing, contextualizing and confining them.

6.2. Situation model for scenario structuring. Interactions are split into a set of situations. Each situation is a sequence of interactions between two or more actors in a precise context to achieve a predictive objective. It is characterized by (Fig. 5) pre-conditions, post-conditions, a set of participating actors and a set of resources. Since actors' behavior, especially for human actors, is not always precisely modeled, and due to the influence of external events, the progression of a situation can be considered an execution and adaptation black box where the interactions are executed in a non-predictable way. A situation also includes consistency management. It represents a set of mechanisms devoted to prevention, detection and treatment solutions, in order to redress and adjust situation progression in spite of misunderstandings and inconsistency problems. Consistency management is carried out all along situation progression from local context initialization to post-conditions completion.

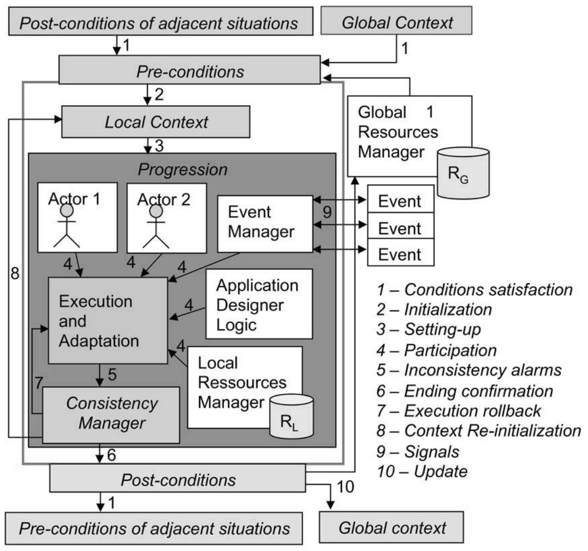

Fig. 5. Elementary situation structure.

6.3. Situation graph and application execution. Situations are considered plot structuring elementary blocks. Each application provides a set of situations defining all the possible interaction sequences that can happen during application execution. They can be grouped and linked together in order to build the overall application scenario. The scenario is then represented by a directed graph of situations. Each node is a situation and each edge is a transition from one situation to another. The situation graph shows the causal relationships between scenario situations. A scenario may have several beginnings and also some possible endings (for instance, Fig. 10 in Section 8 depicts the situation graph of the presented case study).

The situation-based scenario approach improves execution control and interaction adaptation. Application progression becomes a scenario unfolding from one starting node to one final node on the predefined situation graph (taken in charge by *Scenario agent* in the global architecture). If more than one situation is possible, the most pertinent one will be chosen by *Scenario agent*. To increase the adaptability, we can avoid the definition of a predefined graph. In that case, the choice of a situation is made according to the pre-conditions that best satisfy the global state and decision criteria. This method is flexible, adaptive, and applicable in real time during application execution, but it may lead to an uncontrollable situation order or an infinite loop, if the post-conditions and pre-conditions do not contain sufficient data. To avoid this issue, we add a specific situation that handles the absence of post/pre-condition matching when necessary (Pham *et al.*, 2011).

7. Consistency management model

Handling mechanisms. The consistency management that we propose consists of a set of specific methods, techniques and mechanisms that aim to handle the misunderstanding problem and to obtain data consistency all along interactions. They are similar to dependability techniques (Laprie *et al*., 2004).

Prevention mechanisms try to suppress misunderstanding occurrence conditions in order to avoid misunderstandings. To avoid data inconsistency, the proposed technique is an explicit declaration of all involved data before a situation's interaction sequence starts. It aims to identify and share the actors' local visions in order to decrease the possibility of interaction deviation. If inconsistency is detected in the collected data, the actors perform data synchronization. It is also done during the interaction sequence in order to avoid the inconsistency of data newly perceived.

Tolerance mechanisms guarantee interaction continuation despite misunderstanding occurrence.

- *Misunderstanding detection:* regular check of (i) the shared data used during interactions and (ii) the deviation between the actors' logics.
- *Interaction recovery:* once a misunderstanding is detected, the system applies one or several of the following techniques: (i) *rollback* brings the system back to a previous misunderstanding free state to retry interactions; (ii) *rollforward* brings the system to a new misunderstanding free state from which interactions will go on; (iii) *reinforcement* requires from one or several participant actors to do some additional interactions.

Removal mechanisms involve misunderstanding detection and correction, followed by *reinitialisation* of the last interactions, or of the whole interaction sequence. The detected misunderstandings are diagnosed to determine their causes: Which data are inconsistent? Which ambiguities exist in the interaction context? Are there protocol faults? An appropriate correction method is then applied to eliminate the related misunderstanding. Finally, interactions are restarted from the last stable point or from the beginning.

7.1. Inside the situation structure.

Our situation based architecture allows the integration of the previous misunderstanding management mechanisms inside the situation in order to control the misunderstandings and their consequences all along situation execution. We define three phases.

***Prologue* phase.** Before interactions start, the actors' local visions are synchronized through explicit declarations of interaction content and data. If the initial data of the actors involved are identical, the possibility of misunderstanding occurrence will be reduced. If inconsistency exists, a negotiation step is performed between the inconsistent actors. Then, either one or several of them will modify their data, or the divergent data will be isolated/removed and not considered during interactions.

Interaction or the *Dialogue* phase. When interactions are carried out, the actors will update their local data, step by step, as they continuously observe and perceive each other. Despite the initial local vision agreement, misunderstandings may nevertheless occur during interactions. This is why their local shared knowledge is synchronized all along the interaction sequence in order to avoid the divergence of local data about the same facts in the actors' local visions. One or several techniques of *reinforcement, rollback, rollforward* should be used.

***Epilogue* phase.** All the interactions are done in the previous phase. If the post-conditions are fulfilled, we can exit the situation with the expected results. But if, for some reason, we do not reach the expected post-condition, *Script agent* has to detect and settle the existing incoherency in order to avoid the propagation of misunderstandings to other situations. The system may also require that actors perform reinforcing interactions, or, if necessary, make a rollback to a last known stable state, which necessitates a regular state saving mechanism. If it is not possible, a restart of the whole situation should be done. The main goal of this phase is to exit the situation with the appropriate post-conditions and without latent or active misunderstanding. But the rollback or reinforcing interactions may not lead the actors towards the planned post-conditions. Thus, we add in the situation model a special exception exit point that allows the current situation to be stopped without expected post-conditions and that leads to *exception handling situations*.

7.2. Formal validation in Uppaal.

In order to validate the proposed solutions and verify the system's important properties after consistency mechanisms integration, we model our overall proposition using the Uppaal modeling and simulation tool (Behrmann *et al*., 2004). We aim to check what properties are preserved after consistency mechanisms integration. Hence, we model a simple case where the scenario is composed of two situations that have the same behavior and the actors are considered from the consistency management point of view. This is a structural validation of the system's behavior. When the number of situation increases, we shall, additionally, check the situation chaining and scenario validity. Dang *et al*. (2013) show how to use linear logic to achieve this on a entertainment case study. Hereafter, we focus only on structural validation.

7.2.1. Element modelling. The model of our system contains five parts: (i) three models devoted to actor's different aspects (internal behavior, communication channels and its local vision); (ii) the situation block model that integrates the three-phase misunderstanding handling mechanisms; (iii) the scenario model of scenarized application execution presented as a succession of two templated situations. The communication between Uppaal models is done through message exchange (sending/receiving).

Actor models. In order to represent the actor's processes and state, we conceive three automata:

- *ActorLogic* (Fig. 6 (left)) represents the actor's behavior logic by a four-action loop (Observe, Evaluate, Decide, Act). If the actor receives wrong messages, the bad data may disturb its evaluation, decision and activities, and may lead to its blocking. To handle inconsistent data, a synchronization state (**Sync**) is added to check the actor's local vision before the normal activity loop.

- *ActorMessenger* (Fig. 7 (left)) is the communication part devoted to data perception from messages sent by other actors. The perception results may be the states **Expected**, **Lost** or **Unexpected**. It influences the actor's actions.

- *ActorInternalStates* (Fig. 7 (right)) describes the actor's internal state: **Nominal** if nothing goes wrong or **nonNominal** if the actor's action is blocked. According to these two states, different mechanisms of consistency management can be realized: recover, rollforward, exception treatment or restart.

These three automata describe the actors' activity and interaction including misunderstanding occurrence and consistency management.

***Situation* model.** The *Situation* model (Fig. 8) represents a three-phase progress including misunderstanding handling mechanisms as described in Section 7.1. This model is devoted to situation dynamics and presents different global states and transitions. We do not distinguish which transition is carried out by which agent among all system agents and actors of the general architecture (Fig. 4). In particular, the *Dialogue* phase is split into two states: **Execution**, corresponding to interactions between participant actors, and **Consistency**, corresponding to consistency management, where the *Situation* automaton has to supervise the actors' actions states from *ActorLogic* automaton: **CorrectAction**, **DiviantAction** or **Blocked**. Depending on these states, different mechanisms will be selected and applied.

***Scenario* model.** Application execution is in fact a succession of transitions from one situation to another. This process is divided into two steps: the choice of the next situation and the execution of the chosen situation. To model this execution, we built the *Scenario* automaton, where the scenario is composed of two situations S0 and S1, as shown in Fig. 6 (right). The **Selecting** location refers to the decision state, the locations **S0** and **S1** correspond to the execution of the defined situations. In this model, we suppose that the decision mechanism is based only on the satisfaction of situations preconditions. The transition from **Selecting** to **S0** (or **S1**) represents the chosen situation launching. Once this transition is done, the *Scenario* automaton send an authorization message to the *Situation* automaton in order to start situation execution. When the post-conditions of the executed situation are fulfilled, either the next situation is launched or the automaton stops at the **End** location. If the *Scenario* automaton receives the message *except* from the current situation, it will move up to the **ExceptionHandling** state and then finish anyway at the **End** final state.

Communication between models. Figure 9 summarizes the communication between the five automata of our global simulation model. The arrows refers to sending/receiving messages by the automata. The *Scenario* automaton stays at the highest control level and triggers the *Situation* automaton. Interactions between the actor's three models are triggered or modified by the consistency management messages of the *Situation* model.

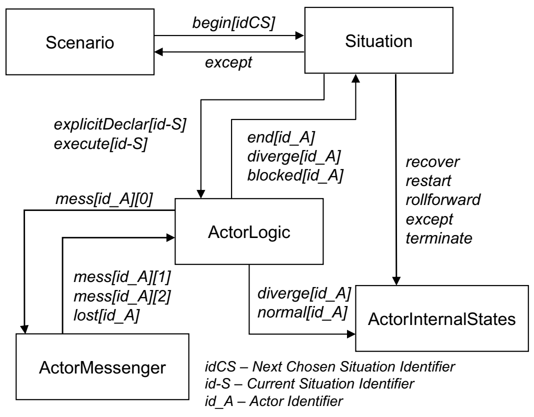

Fig. 9. Exchanged messages between models.

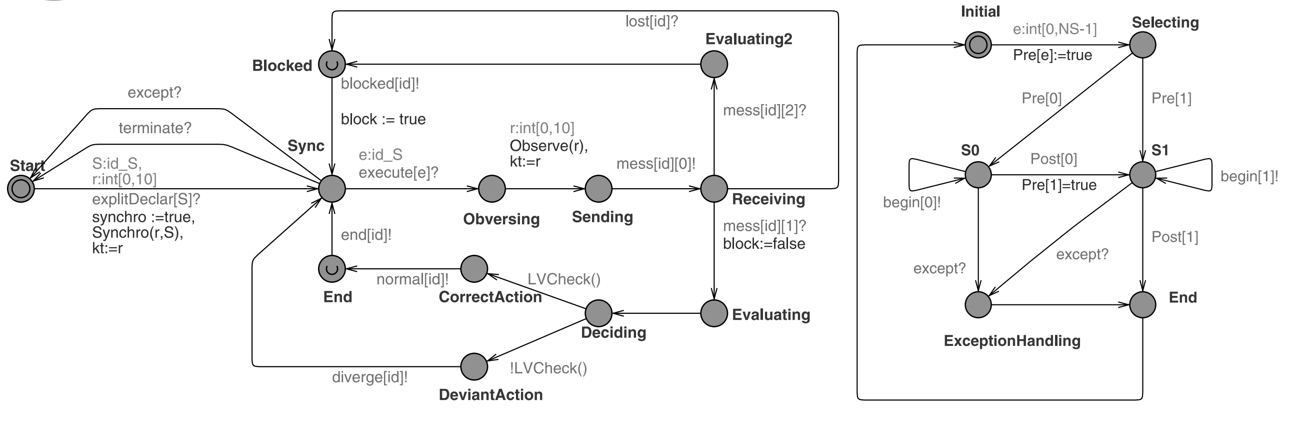

Fig. 6. *Actor* model (left) and *Scenario* model (right).

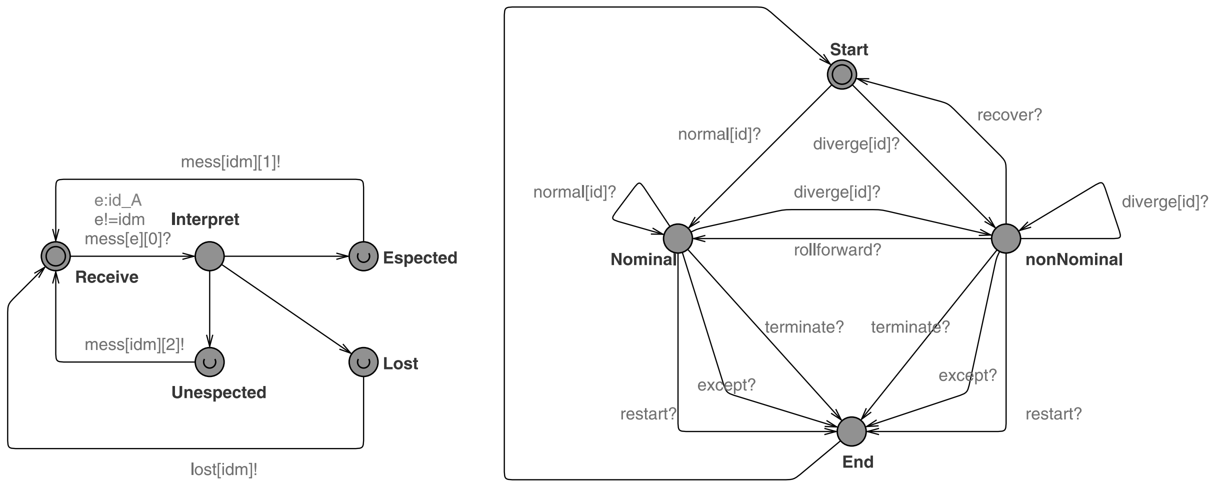

Fig. 7. ActorMessenger (left) and ActorInternalStates automata (right).

7.2.2. Properties validation. The toolkit Uppaal supports not only an automata conceiving editor, but also a simulator to run the system and a verifier to model and check several system properties. We will check our model for three properties: reachability, safety and the absence of deadlock.

Reachability. This property can be understood as follows: Is there a path starting at the initial state, such that a given state formula is eventually satisfied along that path (Behrmann *et al.*, 2004)? We are particularly interested in checking the reachability of all end states: the scenario end, the situation end and the actor's nominal behavior end. These properties do not, by themselves, guarantee the correctness of application execution and actor interactions, but they validate the basic behavior of the model. In Uppaal, we write this property using the syntax $E <> \varphi$:

- **E<>Scenario.End**: the application scenario is executed right to the **End** state,
- **E<>Situation(0).End**: the situation S0 can reach the normal end,
- **E<>Situation(1).End**: the situation S1 can reach the normal end,
- **E<>Situation(0).Exception**: the situation S0 can reach the exception treatment exit,
- **E<>Situation(1).Exception**: the situation S1 can reach the exception treatment exit,
- **E<>ActorLogic(0).End and ActorLogic(1).End**: two actors can realize their nominal actions or preserve their interaction consistency.

These verified properties show that consistency management can preserve system nominal behavior by supervising actor interactions.

Safety. The safety property expresses that, under certain conditions, something bad will never happen (Prigent *et al.*, 2005). It guarantees the respect of structural constraints. It is verified in Uppaal by using path formulas $A[]\varphi$ and $E[]\varphi$ where φ is the state formula:

- **A[] not (Situation(0).nE>NA)**: the number of actors in the situation S0 that reach the **End** location is

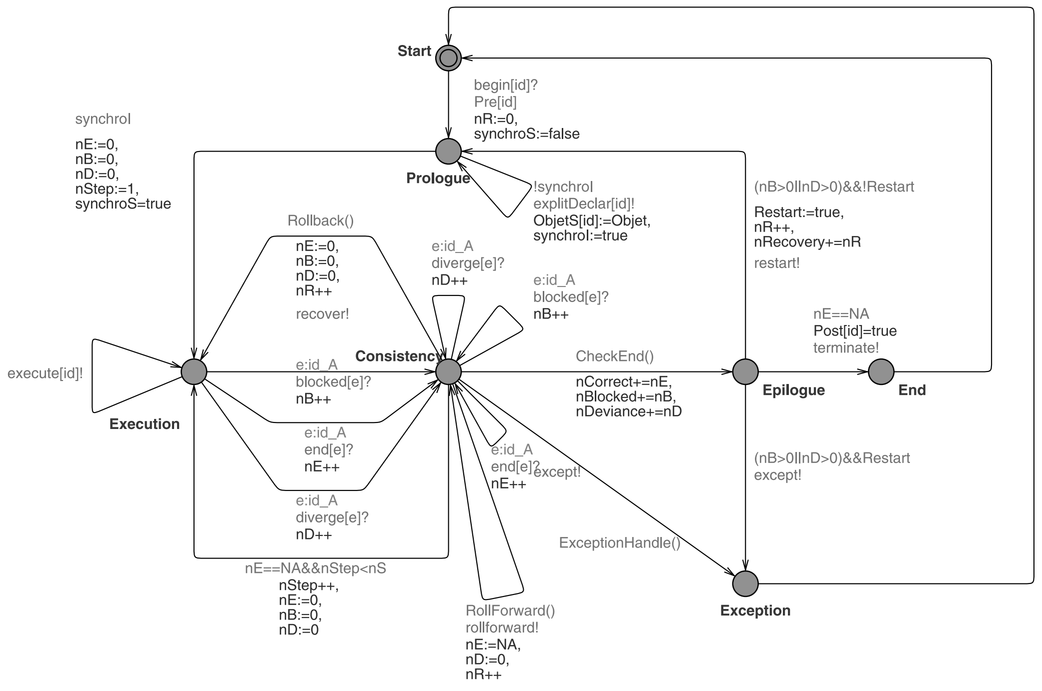

Fig. 8. *Situation* model.

not higher than the total number of actors *NA*,

• **A[] not (Situation(0).nB>NA)**: the number of actors in the situation S0 that are in the **Blocked** location is not higher than the total number of actors *NA*,

• **A[] not (Situation(0).nD>NA)**: the number of actors in the situation S0 whose actions diverge is not higher than the total number of actors *NA*,

• **A[] not (Situation(0).nE + Situation(0).nB + Situation(0).nD > NA)**: the number of state confirmation messages sent by the actors is not higher than the total number of actors *NA*,

• **A[] not (Situation(0).End and Situation(0).nE < NA)**: the situation S0 reaches the normal end with at least all participant actors finishing their interaction correctly.

The same properties for the situation S1 are also verified. This checking shows strict control of consistency management on the choice of appropriate mechanisms according to the actors' states.

Absence of deadlock. The absence of deadlock or a deadlock free system is when the system will never move up to a state where there is no possible progress.

• **A[] not deadlock**: models are deadlock free.

Interactive application execution is an unpredictable process caused by uncontrollable actor actions. The conceived Uppaal models represent an abstract point of view of system components logic and consistency management layer control. Moreover, thanks to the Uppaal simulator and its query language, we can validate, step by step, system execution and verify several important properties concerning integrated misunderstanding handling. All previous properties are verified. That shows structural correctness of our proposed approach.

8. Online distance learning case study

To validate our approach, we applied our situation-based methodology in our current online distance learning (ODL) project (Trillaud *et al.*, 2012). The project is devoted to the development of an online distributed platform that simulates a real classroom: teachers and learners carry out learning sessions as in real life but do so by interacting through a virtual class environment[3]. The platform integrates an interactive numeric board, a camera, a microphone and pedagogic tools (as file sharing system or virtual notebook) to support the courses. Figure 10 shows an example of courses scenario based on 6 situations. The users may face many difficulties: class supervision, course quality assessment, misunderstandings due to weak system interfaces and mechanisms to catch and manage user behavior. The interactions between the actors in ODL contains numerous factors that may lead to misunderstandings: multi-meaning or implicit behavior, supervision tools observation and interpretation imperfection, system component failures, incomplete, missing, implicit or wrong consigns, etc.

[3]`http://foad-l3i.univ-lr.fr/portail/` (in French).

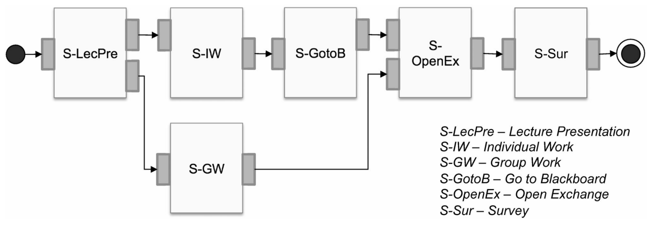

Fig. 10. Situation-based scenario example.

8.1. *Individual work* situation description. To deal with these various misunderstandings, we applied our situation-based solution including consistency management to a particular situation: *Individual work* (SU-IW in Fig. 10). Each learner will work individually and has to do the exercises distributed by the system. The system provides additional exercises each time the learners send the previous exercises report. The expected post-condition is that all the learners reach a required knowledge level *MaxKnowledge*.

Because of the long test duration and development for the real platform prototype, we chose to experiment with our misunderstanding management mechanisms and agent-based architecture through a multi-agent simulation with the GAMA platform[4]. All system actors are modeled and simulated using this platform. We use probabilistic models to represent human actors' behavior. Even though the simulated agents do not behave exactly as real people do, it is sufficient for our purpose because we aim to illustrate and check the benefits of our consistency management mechanisms on a simple case study. Moreover, the simulation experimentation allows parameter tuning and comparison of several experimentation campaigns[5].

8.1.1. Agents. We have four types of agents: *Teacher*, *Learner*, *Observer* and *ODL System*. *Observer's* role is to observe the state of sent exercises in order to evaluate the learners' accumulated knowledge level. The distribution mechanism based on these observations and learners' skill level evaluations is taken in charge by *ODL System agent* that is a combination of the other three agents of our model, introduced in Section 5: *Scenario*, *Script* and *Director* agents (Fig. 4).

Learner agent has to connect to the classroom before receiving the distributed exercises. The factors that influence exercise finishing probability include: exercise difficulty level, the deadline and the learner's smartness level. The learner's knowledge level in a current session is updated after each sent and corrected exercise, and this level will be used by his/her own *Observer agent* to determine if he/she reached the required level, and the exercise distribution will end.

Observer agent's role is that of the go-between for the learner under its responsibility and the ODL system. It observes and estimates the learner's state and transfers it to the system. *Observer* has to check the result of the exercise report (finished or not, accuracy rate), and also supervise the learner's knowledge level in order to be able to notify exercise session ending to the ODL system.

Teacher agent in this situation plays the role of a moderator by supervising all learners' work and checking exercise distribution undertaken by the ODL system.

ODL System agent is a special agent that represents and simulates all the remaining components in our online distance learning platform, including the other system actors, software and pedagogic resources according to our overall architecture in Fig. 4.

8.1.2. Interaction between agents and consistency mechanisms. Figure 11 summarizes the main interactions representing communications between the agents above. To start an exercise session, the teacher and all of the learners have to connect first. The first exercise will be calculated with a random difficulty level and a random deadline. This first decision will be validated or not by the teacher and then sent to the corresponding learner if accepted. Once the learner receives distributed exercise, he/she begins working and sends the report after finishing. This rapport will be analyzed by *Observer agent* in charge, before redirecting it with observing data to *ODL System*. The observing data contain the exercise finishing state, accuracy rate, learner knowledge level and estimated smartness level. *ODL System* uses these observing data for the next distribution: exercises will be adapted to the learner's level so that he/she can finish it with a higher accuracy rate. This strategy suits naturally the learner's desire to be able to terminate exercise session as soon as possible.

However, the observation is never perfect. *Observer agent* can commit an error implying wrong observing data. Potential misunderstandings in this situation may occur when the system distributes exercises that are incoherent given the learners' skills and expectation. They can result from wrong learners exercise state observation or from an inappropriate distributed exercise level. Misunderstanding handling is done inside the situation during its 3-phase progression.

***Prologue* phase.** The system checks each learner's connection status to begin exercise series distribution.

***Dialogue* phase.** In this situation, the interaction content refers to the exercise distribution and reporting. During the learners' work, each *Observer agent* supervises its associated learner's working state and exercise report to collect data: partial or total termination, work duration,

[4] `https://code.google.com/p/gama-platform/`.

[5] We started to perform the same experimentation on our ODL environment prototype mentioned above (work in progress).

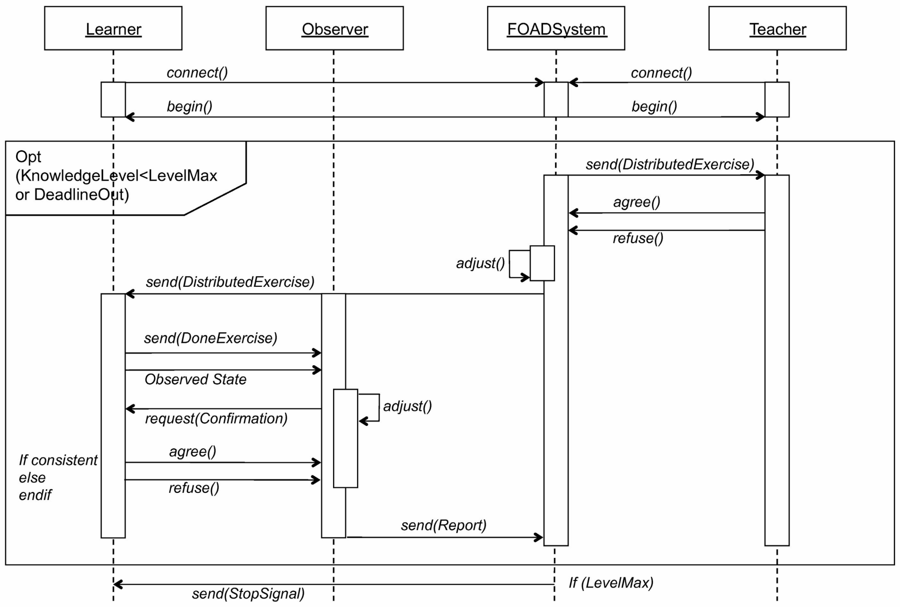

Fig. 11. Agents' main interactions in the simulation.

thee correctness rate. To avoid wrong estimation of the learner's skill and knowledge level, *Observer* synchronizes some of the observed data by asking *Learner* to agree with the collected data before forwarding the report to *ODL System*. This synchronization does not intend to correct the observation but just to check whether or not this observation is accurate for the simulation analysis purpose.

***Epilogue* phase**. To finish the situation, the learners must reach a given skill level after a given number of exercises. If a learner reaches this number without reaching the required skill level, the series will be stopped after a *session deadline* to avoid an abnormal long series. The system sends a *StopSignal* message to all learners to confirm the end of the exercise series after a predefined timeout. It refers to exception treatment.

8.2. Experimentation results. We run the simulation of the *Individual work* situation with the following parameters: 50 learners, one teacher, max knowledge level = 25, max difficulty level = 20, session deadline = 250 steps of simulation. We will measure a set of important factors influenced by potential misunderstandings:

- N_e: total number of distributed exercises,
- N_{notend}: total number of real non-finished exercises,
- N_{bad}: number of bad observation by all observers,
- N_{cor}: number of system observation corrections while detecting wrong observed states (it refers to synchronization times where consistency management is performed to remove incoherent data),
- LI: learners' interest level that increases when the learners succeed and decreases when they fail their exercises,
- T_{total}: total session times (in *steps*) until the last learner has finished his/her series.

The data are recorded and calculated for the average values from 10 simulations launching times in each measure. We compare these data between two cases: "With" and "Without" consistency management. The results are summarized in Table 1. The total distributed exercises number N_e is twice as big in the "Without" case than in the "With" case. The average number of finished exercises in "Without" series is higher than in the "With" series: 747.4 vs 363.4, also depicted in Fig. 12 (left). It is obvious that the session duration in the "Without" case is almost twice longer than in the "With" case.

Figure 12 (right) shows the number of learners that have finished their whole series during situation execution in the "With" and "Without" consistency management cases. The lines shows that the learners work with more exercises and with longer duration T_{total} in the "Without" case. We can make the same observation with the average measure values in Table 1.

Why do we have this difference result? When consistency management is integrated in situation execution to handle the potential misunderstandings, the observers have to adjust their observed data according

Table 1. Statistical data comparison between 2 cases: “With” (Wi) and “Without” (Wo) consistency management.

	N_e		N_{notend}		N_{bad}		N_{obsnon}		N_{cor}		LI		T_{total}	
	Wi	Wo	Wi	Wo	Wi	Wo	Wi	Wo	Wi	Wo	Wi	Wo	Wi	Wo
1	330	735	23	64	83	103	93	114	83	0	78.98	66.1	988	2692
2	363	692	45	28	87	76	110	88	87	0	77.73	76.41	1104	2640
3	361	744	44	55	94	97	114	114	94	0	76.35	69.06	1076	2700
4	383	768	47	73	110	99	129	118	110	0	77.31	65.18	1160	2724
5	347	744	32	60	87	99	109	115	87	0	77.94	67.31	1024	2688
6	360	806	37	65	111	117	122	135	111	0	77.55	64.68	1108	2760
7	392	737	66	59	93	92	118	108	93	0	73.06	66.88	1188	2692
8	379	752	42	66	117	100	131	112	117	0	77.65	65.88	1140	2712
9	353	722	37	69	96	100	111	111	96	0	77.49	67.55	1048	2672
10	361	774	40	62	93	115	117	134	93	0	74.18	65.92	1084	2728
Ave.	363.4	747.4	42.4	60.1	97.1	99.8	115.9	114.9	17.8	0	76.82	67.71	1092	2641

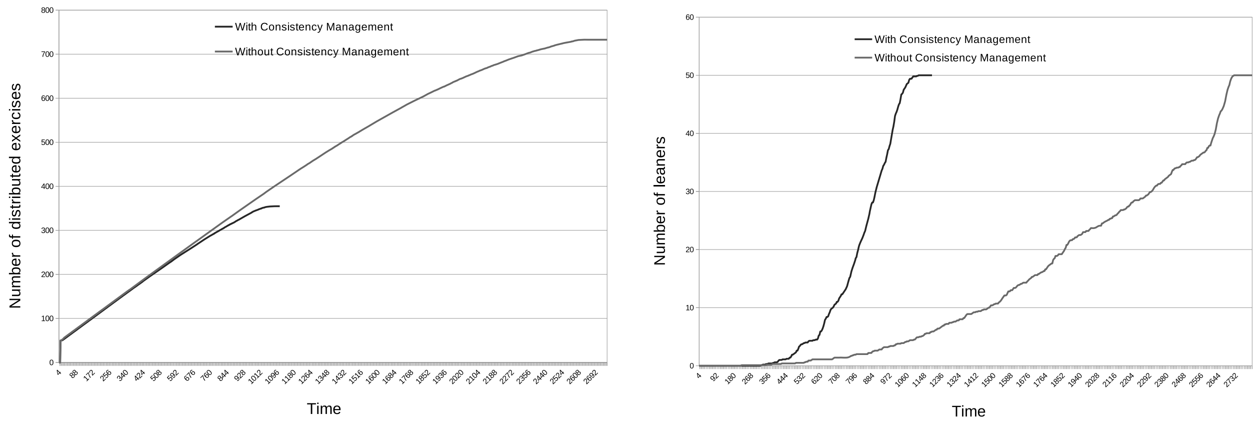

Fig. 12. Comparison between 2 cases: “With” and “Without” consistency management.

to the learners’ *disagree* acknowledgements. Hence, the learner’s skill level estimation will converge faster to the real value, and the difficulty level of the distributed exercises is more appropriate to his/her skills. The result is that learners can finish all the exercises and with a higher correctness rate. In contrast, if no mechanism is added to control the inconsistency between learners and observers, a non-finished exercise can be perceived as finished, and vice versa. Skill estimation is less correct: higher or lower than the real one. There is a higher probability that the ODL system gives to the learners too difficult or too easy exercises. That delays skill level progression, making the learners take more time to terminate the series.

9. Conclusion

In this paper, we presented the situation-based design methodology and consistency management mechanisms to handle misunderstandings in interactions. Our approach is to contextualize the interactions between actors into *situations* and add to these basic narrative blocks consistency management mechanisms split into three steps: *Prologue* (data declaration and consistency verification), *Dialogue* (interaction unfolding, local visions synchronization and misunderstanding treatment), and *Epilogue* (data update and agreement attainment). Our aim is to provide a management pattern that could be systematically used by application designers or developers that allows them to incorporate their own verification, synchronization, prevention and tolerance mechanisms adapted to the specific misunderstandings of their applications.

We formally modeled the proposed approach using the Uppaal tool in order to validate important structural properties. We also applied our methodology to a case study from an online distant learning project. We built a simulation of the *Individual work* situation and integrated into it the proposed solutions to show how consistency management operates on a simulation example. From the experimentation results, we found out that our mechanisms reduce incoherent data between learners and observers and improve the performance of exercise distribution: a shorter session duration, a lower exercise number, faster required level attainment, etc. Even if the simulation is simple and does not cover exhaustively all the possible interactions that can occur in such a situation, it illustrates the benefits of misunderstanding management during interaction progression.

The presented work focuses on the architectural and structural part of our approach. Our current research in progress aims to complete this work from the algorithmic point of view. We are developing post/pre condition matching algorithms and multiple criteria based decision algorithms for situation selection. The first results are presented in the work of Ho *et al*. (2014).

References

Barber, H. and Kudenko, D. (2008). Generation of dilemma-based interactive narratives with a changeable story goal, *International Conference on Intelligent Technologies for Interactive Entertainment, Cancun, Mexico,* pp. 6:1–6:10.

Behrmann, G., David, A. and Larsen, K. (2004). A tutorial on Uppaal, *in* M. Bernardo and F. Corradini (Eds.), *Formal Methods for the Design of Real-Time Systems*, Lecture Notes in Computer Science, Vol. 3185, Springer, Berlin/Heidelberg, pp. 200–236.

Champagnat, R., Delmas, G. and Augeraud, M. (2010). A storytelling model for educational games: Heros interactive journey, *International Journal of Technology Enhanced Learning* **2**(1): 4–20.

Combefis, S., and Pecheur, C. (2009). A bisimulation-based approach to the analysis of human-computer interaction, *ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'09), Pittsburgh, PA, USA*, pp. 101–110.

Dang, K.D., Pham, P.T., Champagnat, R. and Rabah, M. (2013). Linear logic validation and hierarchical modeling for interactive storytelling control, *in* D. Reidsma, H. Katayose and A. Nijholt (Eds.), *Advances in Computer Entertainment*, Lecture Notes in Computer Science, Vol. 8253, Springer, Berlin/Heidelberg, pp. 524–527.

Dey, A.K. (2001). Understanding and using context, *Personal and Ubiquitous Computing* **5**(1): 4–7.

Dourish, P. (2004). What we talk about when we talk about context, *Personal and Ubiquitous Computing* **8**(1): 19–30.

Ho, H.N., Rabah, M., Nowakowski, S. and Estrailler, P. (2014). Trace-based weighting approach for multiple criteria decision making, *Journal of Software* **9**(8): 2180–2187.

Hommel, B., Pösse, B. and Waszak, F. (2000). Contextualization in perception and action, *Psychologica Belgica* **40**(4): 227–245.

Jiang, H. (2005). Confidence measures for speech recognition: A survey, *Speech Communication* **45**(5): 455–470.

Karlsson, B., Ciarlini, A.E.M., Feijo, B. and Furtado, A.L. (2006). Applying a plan-recognition/plan-generation paradigm to interactive storytelling, *Workshop on AI Planning for Computer Games and Synthetic Characters (ICAPS 2006), Cumbria, UK*, pp. 31–40.

Karsenty, L. and Botherel, V. (2005). Transparency strategies to help users handle system errors, *Speech Communication* **45**(3): 305–324.

King, G.G. (2011). General aviation training for automation surprise, *Journal of Professional Aviation Training & Testing Research* **5**(1): 46–51.

Laprie, J. C., Randell, B., Landwehr, C. and Member, S. (2004). Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing* **1**(1): 11–33.

Lebowitz, J. and Klug, C. (2011). *Interactive Storytelling for Video Games: A Player-Centered Approach for Creating Memorable Character and Stories*, Focal Press, Waltham, MA.

Lopez-Cozar, R., Callejas, Z. and Griol, D. (2010). Using knowledge about misunderstandings to increase the robustness of spoken dialogue systems, *Knowledge-Based Systems* **23**(5): 471–485.

Magerko, B. and Laird, J.E. (2004). Mediating the tension between plot and interaction, *AAAI Workshop Series: Challenges in Game Artificial Intelligence, San Jose, CA, USA*, pp. 108–112.

Paul, R., Charles, D., McNeill, M. and McSherry, D. (2011). Adaptive storytelling and story repair in a dynamic environment, *in* M. Si *et al.* (Eds.), *International Conference on Interactive Digital Storytelling (ICIDS)*, Lecture Notes in Computer Science, Vol. 7069, Springer, Berlin/Heidelberg, pp. 128–139.

Pham, P.T., Rabah, M. and Estraillier, P. (2011). Handling the misunderstanding in interactions: Definition and solution, *International Conference on Software Engineering & Applications (SEA 2011), Singapore*, pp. 47–52.

Pham, P.T., Rabah, M. and Estraillier, P. (2013). Agent-based architecture and situation-based scenario for consistency management, *Federated Conference on Computer Science and Information Systems (FedCSIS 2013), Kraków, Poland*, pp. 1065–1070.

Picard, F. and Estraillier, P. (2010). Context-dependent player's movement interpretation application to adaptive game development, *Three-Dimensional Image Processing (3DIP) and Applications, San Jose, CA, USA*.

Prigent, A., Champagnat, R. and Estraillier, P. (2005). Scenario building based on formal methods and adaptative execution, *International Simulation and Gaming Association Conference (ISAGA2005), Atlanta, GA, USA*, pp. 1–19.

Rapaport, W.J. (2003). What did you mean by that? Misunderstanding, negotiation, and syntactic semantics, *Journal Minds and Machines* **13**(3): 397–427.

Sehaba, K., Estraillier, P. and Lambert, D. (2005). Interactive educational games for autistic children with agent-based system, *International Conference on Entertainment Computing (ICEC'05), Sanda, Japan*, pp. 422–432.

Silva, A., Raimundo, G. and Paiva, A. (2003). Tell me that bit again... Bringing interactivity to a virtual storyteller, *in* O. Balet, G. Subsol and P. Torguet (Eds.), *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, pp. 1–10.

Tanenbaum, A.S. and Woodhull, A.S. (2006). *Operating Systems Design and Implementation*, Pearson Education, Upper Saddle River, NJ.

Trillaud, F., Pham, P.T., Rabah, M., Estraillier, P. and Malki, J. (2012). Situation-based scenarios for e-learning, *International Conference on e-Learning (EL 2012), Lisbon, Portugal*, pp. 121–128.

Young, R.M., Riedl, M.O., Branly, M. and Jhala, A. (2004). An architecture for integrating plan-based behavior generation with interactive game environments, *Journal of Game Development* **1**(1): 51–70.

Thao Phuong Pham obtained her Ph.D. in computer science at the L3i Laboratory in La Rochelle (France) in 2013. Her research domain refers to interaction consistency management in interactive systems using situation-based scenarios and fault tolerant techniques. Besides, she explores other domains such as algorithmic, AI, image processing and software engineering.

Mourad Rabah obtained his Ph.D. in computer science at the LAAS-CNRS Laboratory in Toulouse (France) in the dependability domain. Since 2002, he has been an associate professor at the University of La Rochelle within the L3i Laboratory. His current work deals with interactive adaptive systems, where the system adapts its execution to users interactions and behaviors. He explores the application of fault tolerant techniques in order to improve application scenario structuring and adaptation decision-making in interactive systems. He is currently participating in several e-learning projects where the adaptation to learners' progression is partially based on system traces.

Pascal Estraillier is a full professor at the Computer Sciences Department and the Laboratory L3i at the University of La Rochelle. He is also a scientific adviser in the Directorate National Management of Research and Innovation at the French Ministry of Research, in charge of the ICT area. His research concerns the architecture of software components in distributed and cooperative systems. He applies its results to the multi-agent paradigm and uses formal specification theories in order to validate the behavior and interactions between components, and to manage interoperability constraints. The results are mainly applied to games, serious game and e-learning domains in many international research projects.

Received: 30 June 2014
Revised: 2 December 2014