SKETCH'NDO: A framework for the creation of task-based serious games

Sergio Moya, Dani Tost[1,*], Sergi Grau, Ariel von Barnekow, Eloy Felix

Computer Graphics Division, CREB, Technical University of Catalonia
smoya—dani—sgrau—efelix@lsi.upc.edu
ariel.von.barnekow@upc.edu

[*]Corresponding author
[1]Avda Diagonal 247, off. 8-53, Barcelona 08028 SPAIN

Preprint submitted to The Journal of Visual languages and Computing *July 25, 2016*

SKETCH'NDO: A framework for the creation of task-based serious games

Sergio Moya, Dani Tost[1,*], Sergi Grau, Ariel von Barnekow, Eloy Felix

Computer Graphics Division, CREB, Technical University of Catalonia
smoya—dani—sgrau—efelix@lsi.upc.edu
ariel.von.barnekow@upc.edu

Abstract

We present SKETCH'NDO, a framework for the interactive design and creation of task-based serious games in 3D virtual environments. The games are single-user in 3D environment. They are dimensionally congruent: inherently 2D tasks such as reading and writing are done in 2D, while manipulation tasks are 3D. The architecture of the system allows educators to design the tasks with a graphical editor that creates the game automatically. This editor does not require gaming expertise. It only needs educators to specify the correct ways of doing the task, without having to consider all possible erroneous learner's decisions. SKETCH'NDO provides a complete mechanism of monitoring and evaluation of the learner's performance that allows a precise assessment of the learning process. It offers a gradation of levels of assistance that can be fixed by educators or automatically adjusted to the trainee's skills. This way, the same task can be trained from a strictly conductist strategy to a fully constructivist one.

Keywords: Serious Games, Task training, Task evaluation, Authoring tools, 3D Virtual Environments

1. Introduction

Computer Games are playing a major role in children, youngster and even older adults leisure. *Serious* Games (SGs) capitalise on the motivation that persons show towards games in order to help them learn, train or rehabilitate [Fre06]. The so-called "digital game based learning" [Pre03] is emerging as

*Corresponding author
[1]Avda Diagonal 247, off. 8-53, Barcelona 08028 SPAIN

an alternative to traditional learning methods, because, in addition to motivation and fun, games bring repetition, adaptive levels of difficulty and the possibility of automatically reporting the learners' behaviour and extracting deeper and broader data about the learning process.

A field in which SGs seem particularly promising is teaching procedural knowledge, i.e. training the cognitive abilities needed to perform a procedural task: to remember the steps, to apply protocols, to identify critical cases, to evaluate risks and, in general, to solve problems. For this type of training, the learning approach is typically based on the constructivist theory [Jon94]: learners build up their knowledge by themselves through their experiences, and instructors are "facilitators" of the process rather than teachers. SGs are suitable for this learning approach, because they can present a Virtual Environment (VE) as similar as possible to a real one where users can experience in a safe and cheap way.

Various SGs for task training have been described in the bibliography with very diverse applications in medicine [SCK*10] [CSK*11], disaster emergencies [FM13] and industry [RJ97] [GMG*08]. However, in spite of their potential, the use SGs for task training is still limited. Several authors argue that this is probably due to unclear standards and guidelines [YCGW09] and to an incorrect balance between the contribution of educational-domain experts and game experts in the game design. As mentioned by Marne et al. [MWHKBL12], depending on the relative weight of one or the other profile in the game development, games are either educationally efficient but little engaging, or on the contrary, entertaining but invalid for knowledge acquisition. Different methodologies for game design have been proposed [vSdF11], all recommending a strong interrelation between the two teams. Still, it is difficult for educational-domain experts to be aware of the potential and limitations of gaming technology and, for game designers, to truly understand the pedagogical goals [TB08]. A promising solution is to provide tools with which non-game experts are able to design games by their own. This strategy allows not only educators to design games, but also learners to modify them. Game design is thus integrated in the learning process, an educational approach called game-development-based learning (GDBL) [WI12]. This methodology has been successfully used to learn programming and software engineering [SHA*11], but there are very few examples of GDBL for task training and with unclear capabilities [GMG*08] or restricted to 2D point-and-click games [TBEF*13].

Finally, a key-aspect of SGs is assessment through the game and at its end [BKL*13]. Assessment during the game is necessary to adapt on-the-fly its level of difficulty to the capabilities that the user has shown until the moment. It is needed at the end in order to measure the acquired skills

and knowledge. Therefore, SGs must provide mechanisms to evaluate every user's action and determine if it is correct within the context of execution and according to what educators consider to be the protocolary way of doing the task. These mechanisms are difficult to implement when the procedures can be done in equally valid ways and when the spectrum of user's actions at each stage of the game is wide, in particular, for SGs in rich 3D virtual environments.

We present SKETCHN'DO, a framework for the interactive design and assessment of task-training games in 3D environments. SKETCH'NDO provides a simple and fast way of defining task-based games through a visual editor, without need of technical skills. Thus, it can be used by educators and even by trainees themselves, to create or modify games within a reactive environment. In addition, it provides a powerful task engine parser that evaluates all user's actions in comparison to what they are expected to do as expressed by domain experts in the visual editor. Thus, SKETCH'NDO allows a wide range of pedagogical approaches: from a purely conductist orientation with the interactions restricted to valid actions only, to a constructivist perspective leaving users free to interact in the environment, while keeping control on the feasibility of the task.

In the next sections of the paper, after reviewing the related work, we describe the pedagogical requirements towards the definition of pedagogical contents, and we explain the solutions that our system provides to match these requirements. We illustrate the features of the system with three different case studies.

2. Related work

Serious Game Content Edition

The development of games is generally based on a three-level structure (see Figure 1): on the bottom level the graphical libraries for rendering and simulation, for instance OpenGL and PhysX; at an intermediate level, a Game Engine such as Unity, Unreal and Blender GE, that provide an integrated environment with a suite of tools that ease the creation of games and the reusing of parts of them [PDdFP10]; and, at the top level the game scripting. Although many game engines, for instance Game Maker and Unity, integrate a visual editor to define parts of the game, they all rely on programmer's scripting to implement the logics of the game, specially if it is complex, thus they are not readily usable by non-game experts.

Many game studios have developed scripting languages on top of the game engines APIs in order to define at a high level the behaviour of objects

and characters [WCKJG09]. These scripting languages facilitate the modification of games and the re-use of components from one game to the next. However, they are primarily intended at programmers. Some games provide virtual interfaces to their scripting languages that allow *modding*, i.e. that provide users restricted means of modifying the game contents. Nevertheless, although some experiences of Leisure Games (LGs) modifications to create SGs have been described [AML*09], these scripting language interfaces are generally insufficient for domain experts to express their pedagogical contents. Moreover, in general, LGs do not integrate an evaluation mechanism suitable for the required per-action contextualised user assessment.

Pedagogical research has shown that there is a need for tools allowing to express visually the learning processes [MWHKBL12]. The models proposed by [PLLC*06] include interesting concepts such as the use of visual languages and state diagrams of the learning process. However, these models are too abstract to be straightforwardly transferred into games creation. Thus, an intermediate layer between game scripting and pedagogical models is needed: the domain expert editor (see Figure 1). Visual Programming Languages [Pap80] provide this intermediate layer for computing learning, for instance Alice [PBCea95], Squeak Etoys [Kay05] and, later, Scratch [RMea09]. For SGs development, a few game development architectures have been described. Nadolski et al. [NHvdBea08] have proposed the EMERGO methodology to design web-based learning environments based on combinations of images and videos, but not on 3D games. Other authors describe very field-specific expert-domain editor: for industrial training simulation [GMG*08], neurorehabilitation tasks [MTG12], medical training [CC09] and 2D point-and-click games for healthcare procedural knowledge teaching context [TBEF*13].

Evaluation

Task-based games require a control on the activities carried on in the VE in order to evaluate if learners fulfil correctly the task, give them instructions and demonstrate how to perform the task. The HCSM framework [CKP95] defines a hierarchical communication model based on concurrent state machines for autonomous agents simulations that provides this control. This model was successfully applied to traffic control and driving simulations. However, the low level of abstraction of this model makes it difficult to integrate in a re-usable game scripting architecture.

The VET system [RJ97] introduces the figure of virtual teacher for professional training. The system is based on a model of causal links between task steps and goals. A virtual agent monitors continuously the state of the system and keeps track of the goals already fulfilled, thus, it is able to give new instructions on the next goals. The specification of constraints and goals may not be simple in complex tasks, indeed VET lacks from a visual editing

tools to define them.

Ponder et al. [PHM*03] describe an immersive VR decision training system based on a scenario graph of all possible user's interactions. The main drawback of this approach is that when users are allowed to interact freely in the environment, and all their actions need to be controlled, the scenario graph can be huge, and even impossible to specify if the task is complex. The Q language [Ish02] was defined to describe the behaviour of interactive agents in driving simulators and free walking of virtual agents. It presents the same drawback of requiring a complete description of all the potential events in the scenario.

The GVT platform [GMG*08] for industrial task training is composed by a 3D reactive and behavioural environment [MGA07], a scenario engine and a pedagogy engine. The scenario engine is a multi-agent system. Its input is a description of what can be done at each moment of the training session, which can also be very extensive, because the possibilities of interactions in reactive VEs can be very numerous. This description can be specified through a graphical editor. It is encoded with the LORA language [MA06] based on a hierarchical parallel transition model composed of step machines implemented as agents. The pedagogy engine reacts to trainees actions, and adapts the system reaction to each trainee's particularities. The system is designed for a specific training environment, and its generalisation to other scenarios is unclear. In addition, it does not bring the possibility of mixing 2D and 3D interfaces.

Cabas and Chittaro [CC09] use the CTT formalism to describe tasks in form of a hierarchical tree-like structure with temporal relations among subtasks defined by a series of temporal operators. The structure allows to control at each step of the game if the user's actions are correct according to what expected. The main drawback of the system is that the formalism is difficult to understand for a non-expert user, and even if the hierarchical structure makes the specification easier, it still requires to describe all possible transitions, which is very complex in 3D interactive scenarios.

Finally, Torrente et al. [TBEF*13] have recently proposed an educational tool for the development of 2D game-like simulations in a healthcare procedural knowledge teaching context. The description of the story-flow is based on a transition diagram. In order to reduce the complexity of the graph-like representation, the system uses hierarchical representations of the state nodes [MBJMO11]. This solution is sufficient for the 2D point-and-click games, but it is not suitable for games in rich 3D virtual scenarios where the domain of user actions is very large, and thus specifying all possible user actions is impossible.

3. SKETCHN'DO

The analysis of the previous work has shown that in order to facilitate and shorten the design of SGs, it is necessary to provide SGs development platforms with visual editing tools adapted to domain experts allowing them to specify educational contents and assessment criteria. With this goal in mind, based on our previous experience of SGs development for neurorehabilitation tasks [MTG12], we have designed SKETCH'NDO a platform for SGs design in 3D environments that offers a domain expert visual editor usable for very diverse training domains, and that provides an effective mechanism of trainers assessment.

The games that can be designed with SKETCH'NDO are ambiented in 3D VEs (see top images of Figure 2, for instance), but they can accommodate 2D interaction contexts (see the examples bottom right image of Figure 2). They are single-user and based on a first-person perspective. Users are trainees that perform virtually the tasks by interacting with the reactive objects of the VE. The tasks are dimensionally congruent: inherently 2D actions such as reading and writing are done in 2D, while manipulation tasks are done in 3D.

In the games, there is a system-driven character called *task supervisor* that represents the trainer. This character provides instructions and feedback, forbids some actions, realises others by her own, and scores the trainee's performance. The games can be played with different levels of difficulty, and the task supervisor can adjust the difficulty dynamically during the game. Neither the trainee character nor the supervisor character are embodied. The trainee avatar is symbolically represented by the cursor. When the trainee drags an object through the VE, a miniature of this object is attached to the cursor. The task supervisor manifests indirectly, through messages and actions.

3.1. System overview

Domain-expert trainers use a visual editor to define the correct way of doing a task and the expected reaction of the task supervisor to the trainee actions. This definition is called *reference task*. The editor is based on a description of the VE contents and behaviour called *reactive scenario model* (see Figure 3). Once the reference task has been designed, the editor automatically creates the logical engine of the game, called *task engine*. The game is then ready to be played.

Figure 5 shows the process of a game created with SKETCH'NDO. All trainee's interactions are processed by the *interaction manager* which translates them into action queries. The interaction manager enqueues the actions

in a pending actions queue. The logical engine of the game or *task engine* analyses the correctness of the queried actions in comparison to the *reference task*. Depending on the level of difficulty of the game, the *task engine* executes all the actions or only the correct ones. It informs the supervisor who emits feedback and instruction messages, programs new actions or modifies the level of difficulty of the game. The different components of the game and the visual editor are described in the following sections.

3.2. Reactive scenario model and graphical model

The *reactive scenario model* defines the reactive objects of the scenario and the actions in which these objects can participate. The reactive objects have different states. They are represented by a Finite State Automaton (FSA), in which nodes are states of the objects and edges are actions. Edges between nodes are actions that produce change of states and self-loop edges are actions defined at specified states of the objects.

The reactive scenario model is an abstract model; the concrete implementation of the objects and actions is done is the graphical model. Every reactive object state is associated to a graphical object, either 2D or 3D. The trainee's interactions are done at the graphical model level. They are translated as actions of the reactive scenario and are executed as graphical actions at the graphical model level. Actions correspond to verbs of grammatical valence three, i.e. verbs with three arguments: *role*, *receiver* and *emitter*. The first argument *role* is the subject of the action. In trainee actions, it is the trainee's avatar and in system actions, it is the supervisor. The second argument *(receiver)* is the object whose graphical representation has received the interaction. It represents the direct object of transitive action verbs (e.g. supervisor picks the apple) or the complement of intransitive verbs (e.g. trainee goes to the table). Finally, the *emitter* is the object carried by the *role* when interacting with a graphical object. It represents the instrument of the action (e.g. trainee cuts the apple with a knife).

Figure 4 illustrates the relationship between the reactive scenario model and the graphical model. The trainee (role) clicks on the iodine cup graphical model (receiver) holding a clean gauze (emitter). The corresponding action causes a change of state of the object gauze from cleaned to stained. The graphical action is a change of texture of the model.

The graphical model is implemented on top of the Blender Game Engine [Kul12]. It is built by 3D designers using any digital content creation system and then imported to blender. SKETCHEN'DO provides an add-on toolkit on top of blender that allows to define the relationship between the *reactive scenario model* and the graphical model. This is a specialised task aimed at

graphical designers. However, once a scenario is created, trainers can design as many training tasks in them.

3.3. Edition of the reference task

The *reference task* is defined in terms of blocks of three types: structural, contextual and atomic (see Figure 7. Structural blocks hierarchically nest other blocks. They represent the order in which these nested blocks must be performed, either sequentially or in parallel. In a sequential block, a nested block must be done after the former block in the sequence to be considered correct. In a parallel block, instead, the order is not relevant. Parallel blocks can be parameterised to require only a subset of the nested blocks to be completed. This way, the *reference task* is not a unique, rigid procedure. It can define various valid ways of performing specific steps while preserving order constraints on other steps.

Contextual blocks are used to specify the graphical context in which the nested blocks are performed: either the 3D environment or a 2D panel. For instance, Figure 2 bottom, shows the 3D kitchen context used for 3D interactions and a 2D context used at the end of the cooking exercise to help trainers to remember the ingredients they have used. Contextual blocks are also hierarchical. Thus, trainers are allowed to mix 2D and 3D interactions in the same task, and to select the interaction space dimension more suitable for each step of the training activity. For each task, there is only one 3D interaction space: the training scenario. However, the number of 2D interaction spaces is not limited. For instance, a task may require a 2D telephone where trainees can only tip numbers, and an address book where they can select names. When defining a 2D contextual block, trainers must specify in which 2D space the interactions will occur. Thus, the only actions available in that contextual block are those allowed in the corresponding 2D spaces.

Finally, atomic blocks represent the task action triplet *(emitter, verb, receiver)*. Emitters as well as receivers can be defined as specific instances of the reactive scenario entities, any instance of a particular category of entities or a list of entities. Moreover, they can be a reference to an entity used in a previous block. Figure 6 shows the parameters of an atomic block aimed at opening a brick of milk. The action block is the third of the task. The receiver is defined as a reference. It represents the brick of milk that was the receiver of the action block 2.

Atomic blocks allow to edit textual information that the supervisor uses to post instructions and feedback messages related to the action. In addition, they allow to edit a set of parameters defining the context of validity of the action, specifically:

- Initial time: the action is considered valid only if it starts afters this time. It requires trainees to pause and think, a frequent educator's query.

- Maximum time: the action must be accomplished at most in this lapse of time. This parameter allows to check if trainees have internalised the task and are able to perform it fast. The maximum time of all the actions can be multiplied by a slowing factor in early training sessions without having to re-edit the task.

- Automatic: it indicates whether the supervisor must do or not the action if the trainee has failed to do it within the allotted time interval

- Permanent: it indicates that the action cannot be undone. If the trainee undoes the action, the supervisor will ask him/her to do it again. For example, the action is to turn on the television, the trainee does it, but after, he/she turns it off. The action of turning on the television will be queried again, and the supervisor will only consider valid this action, until done. The same process will be applied any time the trainee tries to undo it.

- Dependencies: it indicates which other blocks depend on the action. If an action has dependencies, it automatically becomes a permanent action until all the dependent blocks end. In the former example, if turning on the television is not permanent, but has the dependent task to check the volume of the television, it will become permanent until the checking has been done.

- Force: It may happen that an action has been freely accomplished by the trainee before being asked to. This parameter indicates if the supervisor will force the trainee do the action again at the right time, or if it can be skipped. Sometimes, it is not feasible to re-do an action. In this case, the game will end. As an example, the trainee is asked to cut a wooden plank, but he/she has already cut it before. The task may either proceed, force the trainee to cut another plank or finish.

3.4. The task engine

The *task engine* parses the trainee actions and compares them to the *reference task*. Thus, at any step, it knows which actions must be performed. In addition, analysing the FSA of the reactive objects, it knows which state transitions are needed to bring an object to a state in which an action is feasible. For instance, to fill a cup with water in the sink, it is necessary first

to turn on the tap. Moreover, the *task engine* can retrieve information on the location of the objects in the VE. Thus, it can also determine which actions must be performed to make the objects reachable to perform an action. For instance, to pick a knife in a drawer, it is necessary before to navigate towards the kitchen cabinet and open the corresponding drawer.

As a consequence, the description of the *reference task* does not need to be fully detailed, but only to focus on the important actions, because the *task engine* is able to reconstruct the required auxiliary actions. This simplifies a lot the trainer's task definition. In Figure 8, we show a simple task edited with SKETCHN'DO and we compare it with what it would be necessary to define in a editor that needs to deploy all possible user moves and all auxiliary actions. The *Task Engine* can inform the supervisor to send precise and ordered instructions and warnings. In addition, it can detect how to correct wrong actions, for instance it is necessary to drop a wrong object in order to be able to pick the correct one, or an object should be put back at its previous location if it has been erroneously moved. With this information, the supervisor can warn the trainee and even perform himself the corrective action. Finally, the *task engine* can detect when an action or the whole task is no longer feasible, to make the supervisor finish the game.

The *task engine* is the dynamically programmable logic component of the system. It consists of a hierarchical State Machine (SM) that controls the task execution. It is generated from the *reference task* description created with the task editor. For each block of the *reference task*, a template program is used to generate the SM that controls the behaviour of this block. The generated *Task Engine* is the hierarchical composition of the block's SM. The child SMs are evaluated only if they fulfil a *constraints function* that ensures a correct synchronisation between the different child SMs.

The SMs corresponding to structural blocks are defined as follows:

Sequential The SM of a sequential block is generated by connecting the state of acceptance (SA) of each child SMs with the child SM of the next nested block of the sequence. The SA state of the last child is the SA state of the sequential block SM.

Parallel The SA state of each child SM increases a counter. The Parallel block SM accepts when the counter reaches the number of nested blocks required to be completed. For instance, if the required task is to select four vegetables in any order and there are more than four vegetables in the scenario, the correct approach to model the task is to model a parallel block nesting selections for all vegetables in the scenario and require only four to be completed.

The contextual block does not change the state of the task execution. Therefore the contextual block SM only changes the context of execution (from 2D to 3D or vice-versa) and passes the transition to its child.

Finally, the atomic block SM is represented in Figure 9 in a simplified version. The transitions are the interactions performed by the trainee and the supervisor. The actions described in the *reference task* as the triplet *(emitter, verb, receiver)* cause the transitions between states. The atomic block SM accepts if the conditions that make the action realisable are fulfilled. Specifically, it requires to have picked the correct emitter (if the emitter is not the trainee) and to have the receiver accessible (state $S5$). The transition between $S5$ and SA executes the action.

The collection of states $[S1 - S5]$ includes the possible cases considered in the simplified version. The states are evaluated in order so that the current state corresponds to the first state that accomplishes its condition. For instance, if the receiver is accessible but a wrong object is picked, the current state is $S1$. This consideration simplifies the scheme and avoids to evaluate all the possible combinations. The effect of this simplification is that the trainee must first drop the wrong emitter, then pick the correct emitter, and then select the receiver, in this order necessarily. In addition, if the interaction has been done before the state evaluation, the SM passes to the SA state, except if the *force* parameter of the corresponding atomic block is enabled.

In Figure 9 the black transitions represent the interactions that proceed to reach the final goal, and the red ones represent the incorrect interactions that can be considered as trainee errors. We next describe the different state transitions with the sample task of picking a brick of milk from the fridge and filling a glass with it.

S1: Wrong emitter The trainee has picked an object that is not contained in the emitter list of the atomic block, the brick of milk in the example. This object must be dropped anywhere in order to be able to pick the brick.

S2: Inaccessible emitter The trainee has not picked any emitter yet, and the correct emitters are enclosed in closed objects (containers) or nested containers (a box in a closed cupboard, for instance) making them inaccessible. The trainee must first open the containers to make the emitter objects accessible. It is the case when the milk is inside the closed fridge.

S3: Accessible emitter The trainee has not picked any emitter yet, and one or more emitter objects are accessible. The trainee must pick one

of the emitter objects. In the example, the milk is in the opened fridge.

S4: Inaccessible receiver The trainee has picked a correct emitter or he/she is the emitter of the action. There are not accessible receivers, so the trainee must open a receiver container to make one or more receivers accessible. For instance, the trainee has just picked the brick of milk, but the glass is in a cupboard.

S5: Accessible receiver The trainee has picked a correct emitter or he/she is the emitter of the action, and one or more receiver objects are accessible. The trainee must perform the interaction. In the example, the trainee has picked the milk, and the glass is accessible on the kitchen marble.

SA: Accepting The trainee has performed the interaction and the block is accepted. The trainee must perform the next interactions.

A transition is considered wrong if it does not yield to a state further than the current one. There are only four possibilities of doing a wrong interaction:

1. To pick a wrong object in any state causes a transition to $S1$.
2. To drop a correct emitter object in $S4$ or $S5$ causes a transition to state $S3$.
3. To close the emitter or receiver containers causes a back transition from $S3, S5$ to $S2, S4$ respectively.
4. To do any action with no impact in the emitter or receiver objects does not change the state of the block (loop transitions).

As can be seen, the SM only considers the states and transitions necessary to complete the specified task and ignores the actions that do not yield to the goal ("Do anything else" transitions in Figure 9). This approach simplifies the scheme and guarantees the achievement of the goal whenever possible, i.e. when the task design is terminable and the trainee errors don't break the task requirements.

The simplified scheme exposed represents only the transitions caused by trainee interactions. In addition to these, some supervisor actions can alter the state of the execution. Specifically, when the maximum time for the interaction is exceeded in any state, the supervisor makes a transition that forces the accepting state SA. If the supervisor is configured to do the interaction, it does it, including the opening of the emitter and receiver containers if necessary. The supervisor also forces the SM of permanent

action blocks or actions having dependencies to exit the accepting state if the action is undone.

The *constraints function* imposes restrictions on the SM machines to be evaluated. Specifically, it delays the evaluation of an atomic block SM in the following cases:

- if the initial time of the corresponding block has not been reached;
- if the action block has emitters or receivers that reference objects of other action blocks, the SM corresponding to these blocks must be accepted to evaluate the current block;
- if the atomic block is considered permanent or has dependencies, and the action is undone, all the other SM are delayed until the atomic block is accepted.

3.5. Assistance

This architecture provides a trainee customised level of challenge and a continuum of degrees of assistance from a fully driven interaction mode to a completely free one (see Figure 10). The key elements to provide this gradation are the set of trainee's interactions allowed at each step of the game, the instructions given, the feedback provided on trainees' performance, the time left to do each action and the number of allowed errors. The higher level of assistance is provided when only the "correct" interactions are allowed, i.e. those that follow the *reference task*. In this case, trainees can try to interact with other objects, but the corresponding actions will not be performed. Therefore, learning is free-of-error. At this higher level of assistance, detailed instructions on each next step are given as well as reward and correction feedback on the last interaction. Moreover, if the trainee exceeds the time allowed to perform a particular action or makes a number of trials higher than a given threshold, the system will perform the action automatically to let the trainee proceed to the next step. The opposite side of the continuum allows all interactions and does not give instructions and feedback. In this mode, the task finishes because the trainee has completed it, when the total task time is exhausted or when the task is no longer feasible due to the trainee's errors. Intermediate levels of assistance guide some actions and let trainees perform freely others, but warn them and guide them on how they should be corrected.

3.6. Assessment

Since the *task engine* controls all trainees interactions in their context, it is able to deliver an evaluation report of the trainee's performance. The

assessment takes into account four different parameters: trainee interactions, trainee actions, supervisor actions, times and trainee avatar's travelled distances (see Table 1).

Trainee interactions are trainee input intended at yielding game actions. They can be of different types: click on 2D widgets, 2D and 3D objects, virtual keyboard or camera orientation. Trainee interactions are classified into two categories: *fruitful* and *unfruitful*. Only the former ones yield to the execution of actions in the VE. *Unfruitful* interactions are done on objects that are not reactive at that stage of the game. *Unfruitful* interactions are relevant in the trainee assessment, because, during the realisation of a concrete procedure, they indicate potential errors and, in a free exploration, instead, they may indicate curiosity.

Trainee actions are classified into two groups: *correct* and *incorrect*. The former ones match with the expected actions of the *reference task* or are considered as necessary to perform the expected actions. Each correct action is evaluated according to a score defined in the editor. This way, it is possible to define different valid paths to fulfil an action and scoring higher those considered to be better.

Supervisor actions also split in two types: *expected* actions, already included in the reference task and *assistance* actions carried out by the supervisor because the trainee has failed in doing an action and, according to the level of difficulty of the training, this failure yields an intervention of the supervisor.

The times needed to perform each individual action, a task and the whole game are compared to the maximum allotted corresponding times to provide indications of the fastness of the trainee. Finally, in 3D tasks the distances travelled are also compared to the expected distances to measure to which extent trainees have get lost in the VE.

On the basis of all these indicators, in the visual editor, trainers define a global scoring formula. During the game, this overall score is used to dynamically raise or lower the level of difficulty of the game through different parameters. First, trainers can restrict the number of allowed actions at each step of the game flow in order to prevent trainees from getting lost in an irrecoverable situation, if they launch incorrect actions. When only correct actions are available at each step the learning process is error-free. Next, they can define the factor that modulates the maximum time of each atomic block. Finally, they can define the frequency of the instructions and feedback messages posted by the supervisor. Therefore, the same *reference task* can be used to train with different levels of assistance.

4. Results - Case Study

To illustrate the system's features we describe three simple tasks in three different scenarios: a virtual kitchen, an office and a surgery room (Figure 2). The three tasks can be performed with different levels of assistance. These are simple tasks designed just to analyse how trainers designed them. However, SKETCH'NDO can be used to design much more complex tasks. In fact, we have used SKETCH'NDO in a variety of scenarios for various types of training tasks, from complex surgery procedures to hotel cleaning protocols.

Trained educators create these tasks in less than fifteen minutes. The time needed to train educators depends on their previous skills in using computers and on their degree of involvement in the educational project. It can last from 2 to 10 hours before they are able to design simple tasks as the next described. The difficulties faced by trainers in using the framework were not so related with application interface, but with the fact that they were not used to describe the procedures in a non-verbal, rigorous way. Using the editor requires a reflexion that they were not used to in that way. However, it is precisely this reflexion that is needed to specify tasks. In fact, trainers report that this description of procedures was useful also for other applications than serious games, for instance to design storyboards of simulation sessions in the surgery training case.

4.1. Cooking training

The scenario represents a kitchen filled with a large diversity of objects. The objective of the training is to learn cooking by following recipes. The sample task consists of dressing a dish of macaroni. It all happens in a 3D context. The trainee must put tomato sauce and cheese on top of the macaroni no matter in which order. In order to put the tomato, he/she must first open the tomato can with a can opener. The plate with the macaroni must stay on the kitchen counter while it is prepared, and at the end it must to brought to the table. This simple task illustrates the need of parallel and sequential blocks and of the dependencies between blocks. It can be performed with various levels of assistance.

4.2. Office work

The scenario model shows an office for administrative work training. The sample task consists of looking for the phone number of a given person in a phone book, learning the number within a limited time interval, and dialling it. The task involves the 3D environment plus two 2D interaction spaces: the phone book and the telephone. It illustrates fluent integration of the interaction spaces.

4.3. Surgery

The 3D virtual environment is an operating room to train all clinical roles involved in a surgical intervention. The sample task is addressed to instrument nurses. Its goal is help them to find the most suitable distribution of instrument on the Mayo table. The trainees have a limited time to freely order the instruments on the table. After, the supervisor asks them to select a sequence of instruments as fast as possible. This task shows that the system allows completely free interactions, as well as controlled ones. Specifically, in the first part of the task, the trainee can do any action, without any validation, but in the second part, he/she is required to pick the correct objects in the correct order. With a high level of assistance, in this second part, the trainee will only be able to pick the correct ones, and with a lower level, he/she will be able to pick other objects but will be warned to drop them and pick the valid ones.

5. Acknowledgements

This work has been partially funded by project TIN2011-24220 from the Spanish Government.

6. Conclusions

Designing serious games for professional training requires a strong involvement of trainers. We have presented SKETCH'NDO, a framework for the fast creation of task-based serious games. The system brings trainers the possibility of editing games by themselves in pre-designed 2D and 3D reactive environment. The created games allow a gradation of levels of assistance, and an exhaustive control of trainees behaviour that can used for their evaluation. SKETCH'NDO is currently being used to design a variety of games in different scenarios. Its main advantage in comparison to other frameworks is that it requires trainers to define only the expected procedure, at a high level and without having to deploy all situations that can happen depending on how the trainee acts.

Currently, the creation of the reactive scenario is done by game designers following general requirements of trainers. Thus, trainer define tasks on predefined scenarios. The next step in the development of the system is to create a graphical scene editor that could be used by educators to customise the environments using pre-designed objects and actions. This would provide even more flexibility in designing tasks.

The current implementation of SKETCH'NDO is for single-user training. However, the architecture of the *task engine* can easily be extended to support

multiple trainees working collaboratively on the same task. In the future, we plan to implement this extension and modify the task definition to restrict some steps of the task to specific trainees.

References

[AA07] Adj C., Allegretto F.: Virtual pads: decoupling motor space and visual space for flexible manipulation of 2D windows within XEs. In *ESE Symposium on 3D User Interfaces* (2007), pp. 99–105.

[AML*09] Anderson E., McLoughlin L., Liarokapis F., Peters C., Petridis P., de Freitas S.: Serious games in cultural heritage. star. In *The 10th International Symposium on Virtual Reality, Archaeology and Cultural Heritage* (2009), pp. 1–20.

[BKL*13] Belloti F., Kapralos B., Lee K., Moreno-Ger P., Berta R.: Assessment in and of serious games: an overview. *Advances in Human-Computer Interaction 2013*, 136864 (2013), 1–11.

[CC09] Cabas A., Chittaro L.: Using a task modeling formalism in the design of serious games for emergency medical procedures. In *IEEE Conference on Games and Virtual Worlds for Serious Applications* (2009), pp. 95–102.

[CKP95] Cremer J., Kearney J., Papelis Y.: Hcsm: A framework for behavior and scenario control in virtual environments. *ACM Trans. on Modeling and Computer Simulation 5* (1995), 242–267.

[CSK*11] Cowan B., Sabri H., Kapralos B., Cristancho S., Moussa F., Dubrowski A.: SCETF: Serious game surgical cognitive education and training framework. In *GIC'2011* (2011), pp. 130–133.

[DD05] Darken R., Durost R.: Mixed-dimension interaction in virtual environments. In *VRST '05* (2005), pp. 38–45.

[FM13] Farra S., Miller E.: Integrative review: Virtual disaster training. *Nursing Education and Practice 3*, 3 (2013), 93–10.

[Fre06] Freitas S. D.: *Learning in immersive worlds.* Tech. rep., JISC e-learning programme, 2006.

[GMG*08] Gerbaud S., Mollet N., Ganier F., Arnaldi B., Tisseau J.: GVT: a platform to create virtual environments for procedural training. In *IEEE Virtual Reality'08* (2008), pp. 225–232.

[Ish02] Ishida T.: Q: A scenario description language for interactive agents. *IEEE Computer 35* (2002), 54–59.

[Jon94] Jonassen D.: Thinking technology: Toward a constructivist design model. *Educational technology 34*, 4 (1994), 34–37.

[Kay05] Kay A.: Squeak Etoys, Children, and Learning. *Squeakland publications* (2005).

[Kul12] Kuller V.: *Blender Game Engine.* PACKT publishing, 2012.

[MA06] Mollet N., Arnaldi B.: Storytelling in virtual reality for training. In *LNCS 3942*. Springer-Verlag Berlin, 2006, pp. 334–347.

[MBJMO11] Marchiori E., Blanco A., J.Torrente, Martínez-Ortiz I.: A visual language for the creation of narrative educational games. *Journal of Visual languages and Computing 22* (2011), 443–452.

[MGA07] Mollet N., Gerbaud S., Arnaldi B.: Storm: a generic interaction and behavioral model for 3D objects and humanoids in a virtual environment. In *IPT-EGVE Symposium* (2007), Fröhlich B., Blach R., van Liere R., (Eds.), pp. 95–100.

[MTG12] Moya S., Tost D., Grau S.: Interactive graphical design of 3D serious neurorehabilitation games. *Presence: Teleoperators and Virtual Environments 21*, 1 (2012), 58–68.

[MWHKBL12] Marne B., Wisdom J., Huynh-Kim-Bang B., Labat J.-M.: The six facets of serious game design: a methodology enhanced by our design pattern library. In *EC-TEL'12* (2012), pp. 208–221.

[NHvdBea08] Nadolski R., Hummel H., van den Brink H., et al. J. H.: Emergo: A methodology and toolkit for developing serious games in higher education. *Simulation and Gaming 39*, 3 (2008), 338–352.

[Pap80] Papert S.: *Minstorms. Children, computers and powerful ideas*. Basic Book Publishers, 1980.

[PBCea95] Pausch R., Burnette T., Capeheart A., et al.: Alice: Rapid prototyping sypstem for virtual reality. *IEEE Computer Graphics and Application* (1995), 195–203.

[PDdFP10] Petridis P., Dunwell I., de Freitas S., Panzoli D.: An engine selection methodology for high fidelity serious games. In *VS-GAMES '10* (2010), pp. 27–34.

[PHM*03] Ponder M., Herbelin B., Molet T., Schertenleib S., Ulicny B., Papagiannakis G., Magnenat-Thalmann N., Thalmann D.: Immersive VR decision training: Telling interactive stories featuring advanced virtual human simulation technologies. In *9th EG Workshop on Virtual Environments* (2003), pp. 97–106.

[PLLC*06] Paquette G., Léonard M., Lundgren-Cayrol K., Mihail S., Gareau D.: Learning design based on graphical knowledge-modelling. *Education Technology and Society 9* (2006), 97–112.

[Pre03] Prensky M.: Digital game-based learning. *Computers in Entertainment 1*, 1 (Oct. 2003), 1–21.

[RJ97] Rickel J., Johnson W.: Steve: An animated pedagogical agent for procedural training in virtual environments (extended abstract). *SIGART Bulletin 8* (1997), 16–21.

[RMea09] Resnick M., Maloney J., et al. A. M.-H.: Scratch: programming for all. *Communications of the ACM 52*, 11 (2009), 60–67.

[SCK*10] Sabria H., Cowana B., Kapralosa B., Ported M., Backstein D., Dubrowskie A.: Serious games for knee replacement surgery procedure education and training. *Procedia - Social and Behavioral Sciences 2*, 2 (2010), 3483 – 3488.

[SHA*11] Sung K., Hillyard C., Angotti R., Panitz M., Goldstein D., Nordlinger J.: Game-themed programming assignment modules: A pathway for gradual integration of gaming context into existing introductory programming courses. *IEEE Trans. on Education 54*, 3 (2011), 416–427.

[TB08] Tran M., Biddle R.: Collaboration in serious games development: a case study. In *ACM FuturePlay* (2008), pp. 49–56.

[TBEF*13] Torrente J., Borro-Escribano, Freire M., del Blanco A., Marchiori E., Martinez-Ortiz I., Moreno-Ger P., Fernandez-Manjon B.: Development of game-like simulations for procedular knowledge in healthcare education. *IEEE Trans. on Learning Technologies 99*, 1939-1382 (2013), 1–14.

[vSdF11] van Staalduinen J.-P., de Freitas S.: A game-based learning framework: Linking game design and learning. *Learning to Play: Exploring the Future of Education With Video Games 53* (2011), 29.

[WCKJG09] White W., C. Kochand J. Gehrke A. D.: Better scripts, better games. *Communications of the ACM 52*, 3 (2009), 42–47.

[WI12] Wu B., IngeWang A.: A guideline for game development-based learning: A literature review. *Computer Games Technology 2012* (2012), 1–20.

[YCGW09] Yusoff A., Crowder R., Gilbert L., Wills G.: A conceptual framework for serious games. In *ICALT '09* (2009), pp. 21–23.

Table captions

- Table 1: Registered indicators for user assessment

Figure captions

- Figure 1: Program layers of a SG and the role of domain experts and game developers in game creation.
- Figure 2: Three different training scenarios for training games design with SKETCH'NDO. Top left: surgery training; top right: office work; bottom: cooking, at left a 3D view at right a 2D context view.
- Figure 3: Overview of the system: educators use the visual game editor to create the the game.
- Figure 5: Structure of the system: the interaction manager enqueues the trainee's action query; the *task engine* evaluates them and informs the supervisor, who enqueues new actions and emits messages addressed to the trainee.
- Figure 4: A trainee interaction on the iodine cup graphical model is interpreted as the action *trainee stain the gauze* that produces a state transition of the reactive object gauze. The action is implemented as a change of texture of the graphical model.
- Figure 6: A panel for the edition of atomic blocks. The reciver of the action is a reference to the receiver of a prvious action
- Figure 7: A view of the SKETCH'NDO editor. Users select the different types of blocks from the upper menu bar and distribute them on the graphical area. The blocks can be nested. The parameters of a block are defined in the panel at right. The blocks of the reference task shown in the figure consist of opening the fridge and either picking the cheese or picking the milk and looking its expiracy date. All the block, but ticking the expiracy date are 3D contexts.
- Figure 8: Example of a task specified with the SKETCH'NDO editor (left) and using a graph description technique (right). The task consists first of putting the salad on a plate and then add salt, olive oil and vinegar in any order. As it can be seen the expression of the task in editor is simpler, faster to edit and more readable. The graph description has been simplified avoiding to put states where the user grabs other objects that are not related to the task.

- Figure 9: Simplified state machine of the Atomic action block. It controls the context to perform a user interaction.

- Figure 10: The level of assistance continuum

Interaction	Navigation and camera orientation		
	Virtual keyboard		
	Click on	2D widgets 2D objects 3D objects	Fruitful Unfruitful
User actions	Per type	Correct Incorrect	
Supervisor actions	Per type	Expected Assistance	
Times	Per action	Per task	Per game
Traveled distance	Per action	Per task	Per game

Table 1:

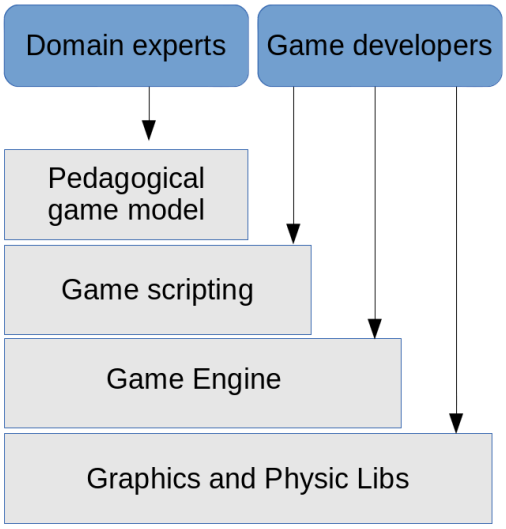

Figure 1:

Figure 2:

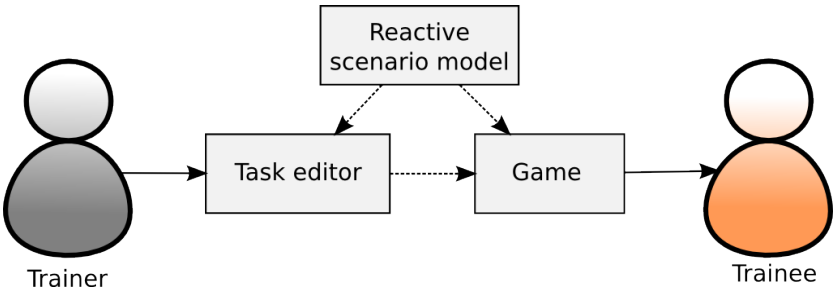

Figure 3:

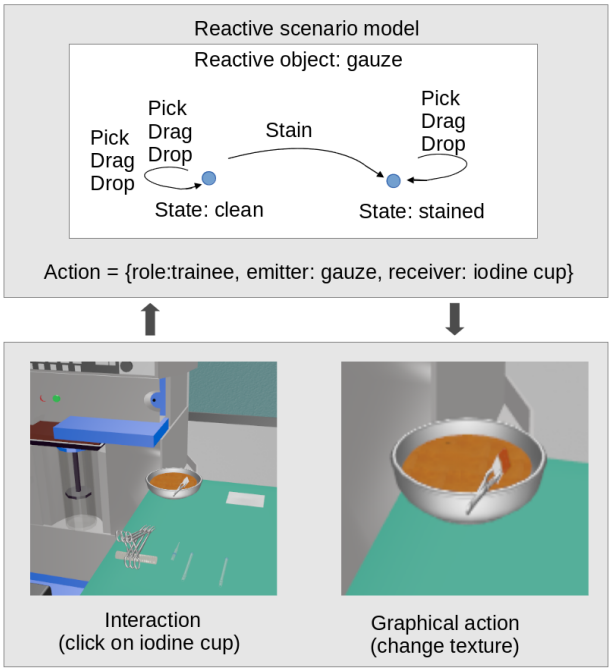

Figure 4:

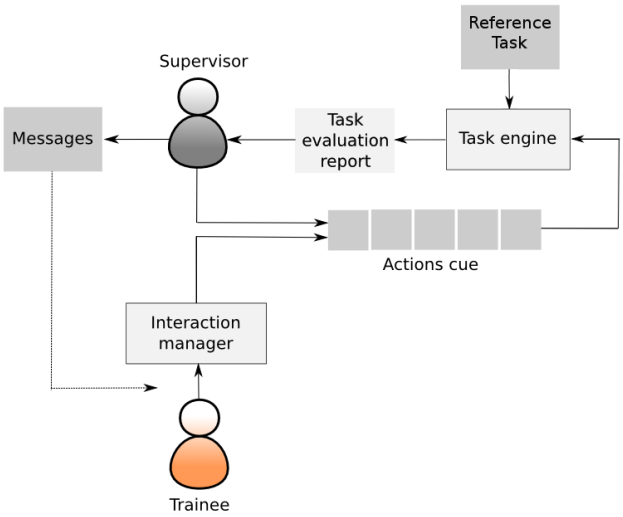

Figure 5:

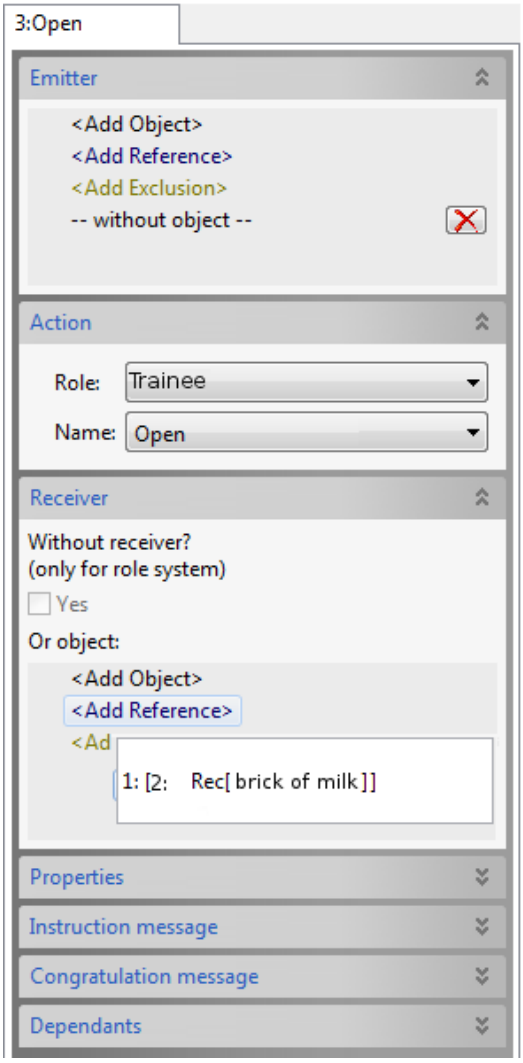
3:Open
Emitter
<Add Object>
<Add Reference>
<Add Exclusion>
-- without object --
Action
Role:
Trainee
Name:
Open
Receiver
Without receiver?
(only for role system)
Yes
Or object:
<Add Object>
<Add Reference>
<Ad
1: [2: Rec[brick of milk]]
Properties
Instruction message
Congratulation message
Dependants

Figure 6:

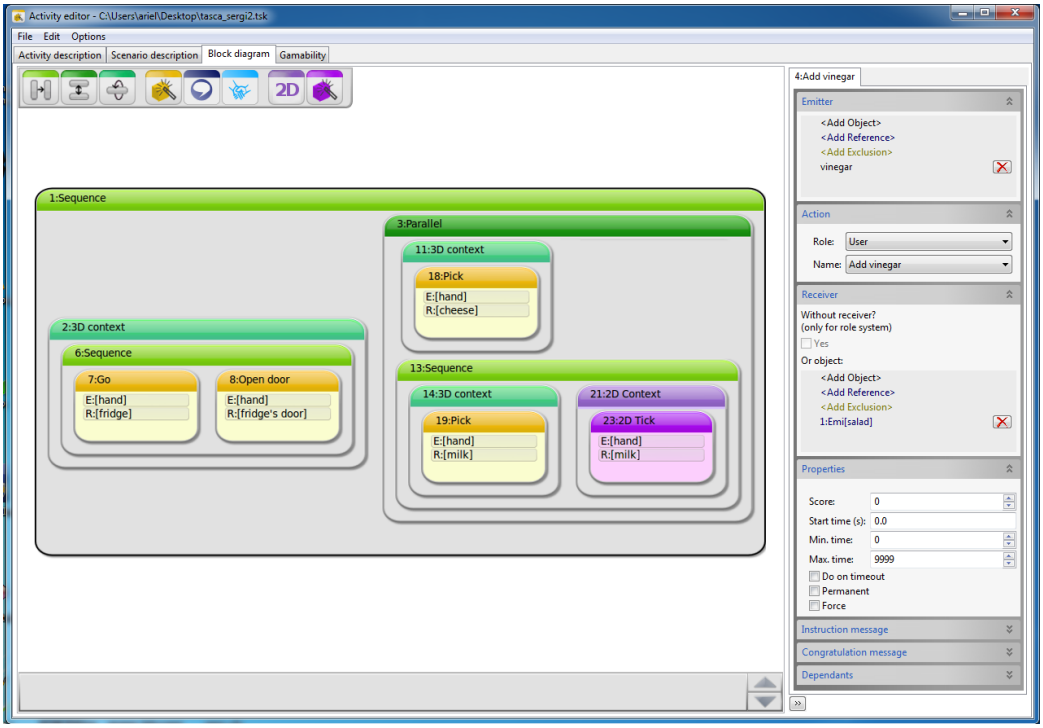

Figure 7:

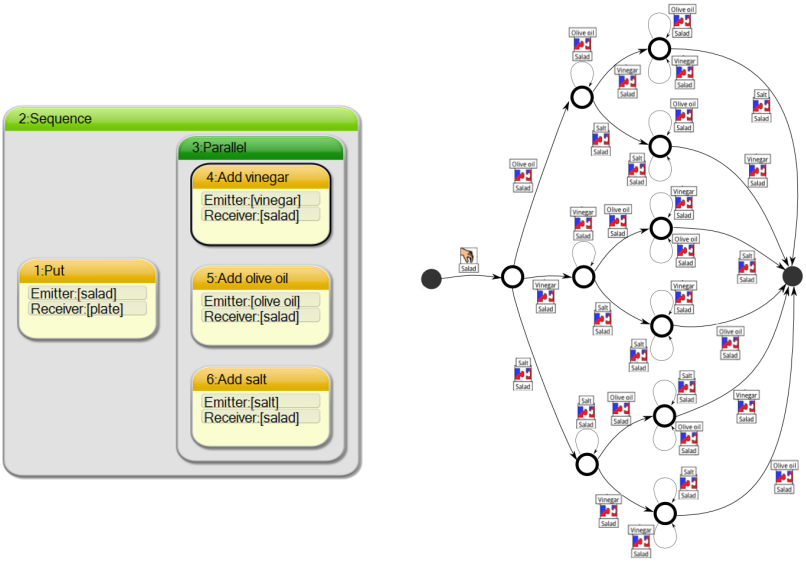

Figure 8:

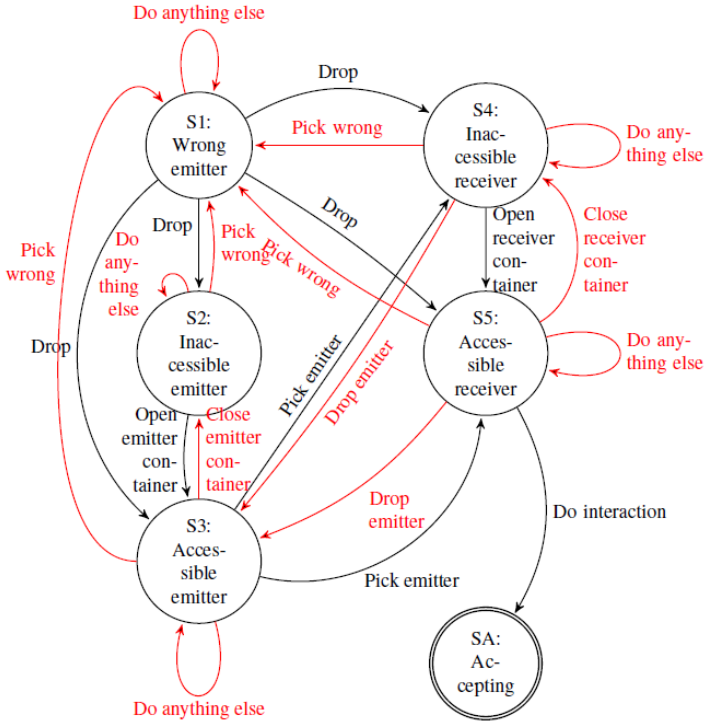

Figure 9:

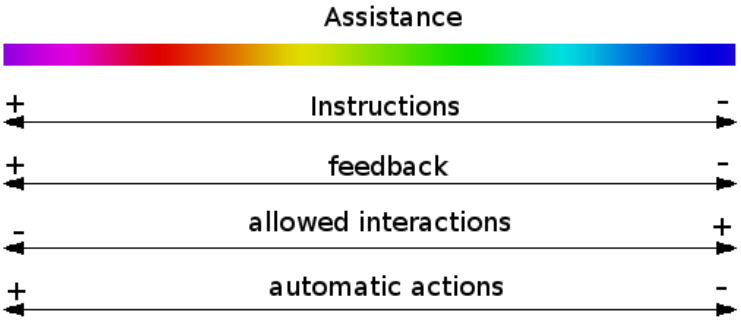

Figure 10: