SIMACT: a 3D Open Source Smart Home Simulator for Activity Recognition with Open Database and Visual Editor

K. Bouchard, A. Ajroud, B. Bouchard and A. Bouzouane

LIARA Laboratory, Universite du Quebec a Chicoutimi (UQAC)
555 boul. Universite, Saguenay (QC), Canada, G7H 2B1
{Kevin.Bouchard, Amir.Ajroud, Bruno.Bouchard, Abdenour.Bouzouane}@uqac.ca

Abstract

Smart home technologies have become, in the last few years, a very active topic of research. However, many scientists working in this field do not possess smart home infrastructures allowing them to conduct satisfactory experiments in a concrete environment with real data. To address this issue, this paper presents a new flexible 3D smart home infrastructure simulator, which is developed in Java specifically to help researchers working in the field of activity recognition. A set of pre-recorded scenarios, made with data extracted from clinical trials, will be included with the simulator in order to give a common foundation to test activity recognition algorithms. The goal is to release the SIMACT simulator with a visual scenario editor as an open source component that will benefit the whole smart home research community.

Keywords: *3D simulation tool, Smart Home, Activity Recognition, Cognitive Assistance, Real case scenarios, Open source*

1. Introduction

The aging of the population combined with a birth rate decline in occidental countries will result in growing needs in healthcare services. A significant increase of diseases causing a cognitive deficit such as Alzheimer's [1] is predicted by both national [2] and international organizations [3]. During the last few years, the smart home has become firmly established as an active topic of research [4] exploring the process by which adapted assistance for the in-house performance of the Activities of Daily Living (ADL) [5] can be provided to a resident suffering from some type of impairment whether due to his age or a disease. A smart home can be defined as an enhanced environment with miniaturized processors, software (mobile) agents communicating between each other, and multi-modal sensors that are embedded in any kind of common everyday objects, making them effectively invisible to the habitant [6]. The first fundamental step in smart home assistance is to be able to identify the on-going inhabitant ADL [7] from observed basic actions and from the events produced by these actions. It is why a growing community of scientists [8, 9, 10], like us [6, 11], are investigating this specific problem of recognizing ADL in a smart home. However, many of these researchers face a major difficulty in their investigation: they have only a limited access or no access at all to experimental smart home infrastructures. The problem is that the construction of a smart home requires space, money and a lot of time. Moreover, even when they have access to these infrastructures, they often are unable to experiment with real case scenarios because of the complexity of gathering real data from conducting clinical trials. Furthermore, real large scale experiments are very expensive, and thus only a limited number can be conducted. Therefore, to adequately investigate this field, one must be able to conduct satisfactory experiments with concrete data at an affordable cost, and one needs a common foundation (platforms, standards, tools, etc.) upon which different recognition methods and algorithms can be validated experimentally and compared.

To address this issue, several projects [12, 13, 14, 15, 16], over the last few years, tried to develop simulation tools for conducting smart home experiments. However, these previous tools suffer from many weaknesses. First they are not offered freely to the researchers as an open source component. Their designs were not made to test activity recognition approaches, and they are not flexible enough to be customize in a way to answer specifics needs. ISS, for instance, presents a static 2D habitat virtualization without any modifiable characteristics or features. More importantly, none of them provide a set of pre-recorded real case scenarios of activities with the tool. In this paper, we propose a new flexible 3D smart home simulator developed in Java. This simulator is specifically designed to help researchers working in the field of activity recognition [11] to conduct experiments with real data. The architecture of the system allows a third party application (such as an intelligent recognition agent) to connect itself easily to a database in order to receive, in real time, the simulated sensors' inputs. A set of pre-recorded scenarios, made with data extracted from clinical trials, will be included with the tool in order to give a common foundation to test activity recognition algorithms. The goal is to release the SIMACT simulator as an open source component [17] that will benefit the whole smart home research community.

The paper is organized as follows. Section 2 presents the new proposed simulator by describing the general functionalities, graphical user interface, advanced parameters, software architecture and Java implementation. Section 3 explains how the parameterization of SIMACT is done in text based files. This part includes .ini file, dynamic object declaration with XML and scripting in text files. Section 4 describes the new visual scenarios' editor that will be included in the SIMACT package. This section presents the overall features of the editor and shows, step by step, how to build a script with the visual editor. Section 5 presents our actual initiative to gather real data that will be incorporated in the simulator as real case pre-recorded scenarios. Section 6 presents an overview of related work describing positive and negative aspects of each one. Finally, section 7 concludes the paper by highlighting important features and by outlining future developments of this work.

2. SIMACT: a 3D Smart Home Simulator

SIMACT was built with the intention of providing a software experimentation tool that will help researchers validate their smart home recognition approaches. It is really a simple and straightforward application, meant to concrete function of testing and validating ideas and algorithms in the specific field of activity recognition. However, as we shall see in this section, nothing prevents its use for other topics about the research on smart homes because of its enormous flexibility. One of the most important advantages of SIMACT for the scientist is that it will be delivered as a freely open source component on the Internet. Open source is well recognized to improve robustness, but it is also specifically effective to allow future enhancements and evolution on software. In addition, it allows someone to customize it in the goal of answering his special needs.

The application consists in reproducing the behavior of a person really living in a smart home. To do so, the simulator has the same basic architecture as a smart home. It is simply composed of different sensors that analyze what happens in the house and write the information in a database. Obviously, these sensors and the concept of a house are virtually defined in the software. However, to help the user, SIMACT incorporates a three dimensional representation of this environment. Furthermore, a smart home is normally barely reconfigurable, but as you will see in subsequent sections, SIMACT's virtual environment is perfectly fitted to be changed at will. Secondly, to simulate triggering and actions on the sensors, a simple method must be provided. In SIMACT, this is done by the implementation of scenarios of the activities of daily living. A scenario is a series of steps, or a plan, including incoherent behavior or not within the process of achieving a concrete goal. To be specific, it aims to reproduce a common activity done by a human being through a specific and well described script. Large sets of scenarios are easily defined with our formalism and our tools. With these scenarios, SIMACT plays the role of an interpreter to execute each of them as a real person would do in a real smart home. Moreover, SIMACT offers many interesting options to control the execution and to see visually the progression of steps. It allows one to replay the same sequence of steps as much as is wanted. Thus it is even more suited to comparison between algorithms than a real smart home would be.

2.1. User Interface

The user interface of SIMACT has been designed to be uncomplicated, and so only the important features are displayed on the main screen. As we can see in Figure 1, the window of the application is divided into three particular zones. The first one is a three dimensional graphic canvas built from one of the most promising game engines in Java: the Java Monkey Engine (JME) [18]. This frame contains all that is necessary to simulate the real environment incorporating virtual sensors. In the 3D zone, one can freely move the camera around the environment to take different points of view and then decide where is the most appropriate position to see what is going on. It is simply done by clicking on the frame and using a keyboard: the arrow keys to rotate and pg up/pg down to zoom. Another quality of the 3D zone is that it was built in a way to abstract code from design. It is not necessary to dive into Java code to modify the environment. SIMACT comes with a ready-to-use 3D kitchen, but one may decide to change it into another kind of room such as a bathroom or a bedroom. These changes are really effortless to achieve, and it is also possible to add sensors and animated objects without touching a line of code.

The second zone is the script control zone, which provides a fast, reliable and easy way to control the execution of a script in real time. To make it simple, it works like a video player; one can watch it play, make it pause, and even go back and forward in the execution. During play time, the application shows some useful information about the current step. The speed of the execution can be adjusted without changing the real time value. For example, if there is one step that normally takes five seconds and the playing speed is multiplied by a factor of five, it will take only a second to execute it, but SIMACT will proceed as if it has taken five seconds to execute the task. During the execution of a script simulation, SIMACT gathers data from virtual sensors and saves it in a database. Consequently, a third party application (i.e. a recognition agent) may be built independently and connected to the database to extract the data from the simulation in real time. SIMACT will come with a predefined set of scripts to get started in a flash, because we really want to provide an easy-to-use tool. Users also have the possibility to write custom scripts. SIMACT's flexibility can be truly seen with this abstraction of the script. To be clear, the scripts define a series of interventions in the simulated environment, and SIMACT interprets these actions and acts accordingly. To illustrate, the script may say: "Tactile mat in front of the oven activated." and the response from SIMACT will be to change the color of the mat, insert an event in the database and show a message to the user.

These messages are shown in the third and last zone of the graphical user interface: the console zone. As of today, this part typically serves to act as a user journal keeping a trace of what happened. The console is, in reality, a multipurpose area where future options and miscellaneous information are to be displayed. Inline scripting and parameterization are to be implemented in following version directly within the console area to give more control to the user.

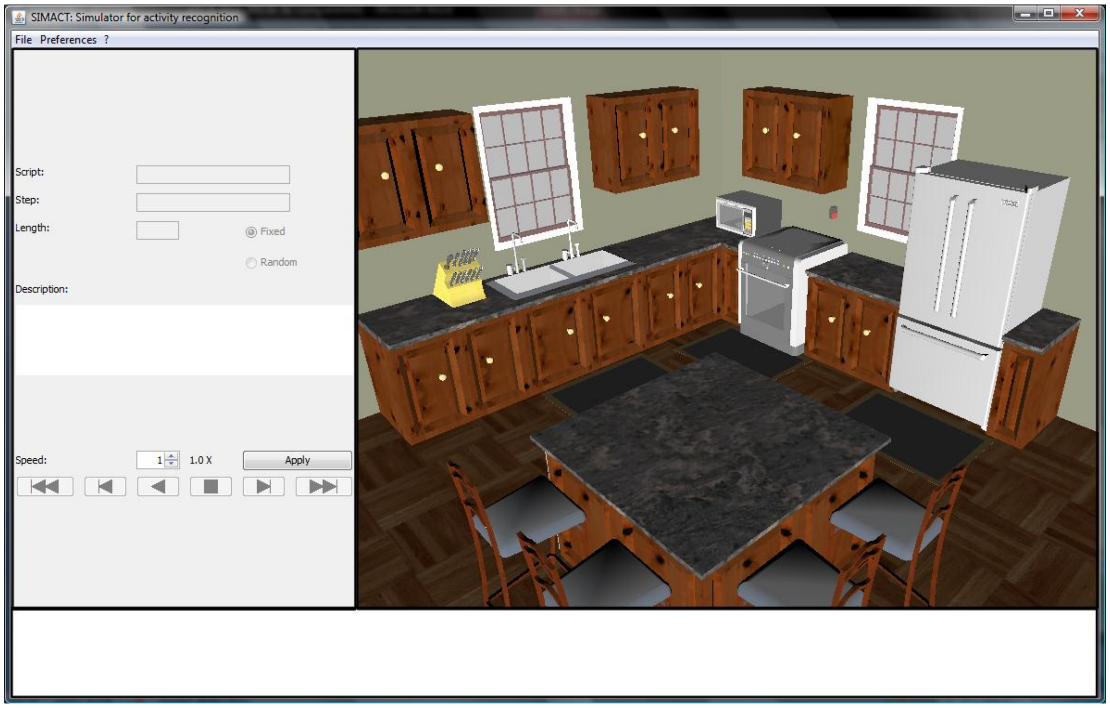
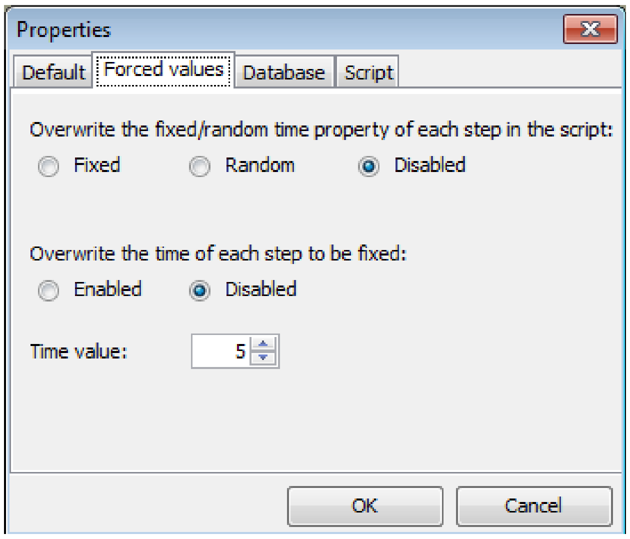


Figure 1. SIMACT Simulation Tool in Action

2.2. Options and Parameters

There are several options that can modify the way SIMACT works or even the way scripts are interpreted. The menus allow the user to change quickly the most important one, but there is also an advanced window in the Properties submenu that is better adapted to precise customization. Figure 2 shows this window while being in the Forced values tab. The various parameters were separated into four categories in order to keep the display lighter. Besides, this allows the user to find easily the options he seeks and so saves his precious time. The following paragraphs will describe some functions already implemented in SIMACT.

The first section, named Default, includes the values used during the script interpretation in the case where the parameters' definitions were missing in the script. For example, the default time for a step can be defined here, and it will be used for each step without a predefined time value. Another option consists in choosing if a step should have a fixed time value or not. The normal time for the step is generally fixed by default but could be switched to be random. This means that every step that does not specify a fixed time in their definition would have a random time execution. To illustrate, if step X takes normally 5 seconds, it may vary from 5 seconds to 5*Y seconds with the random time execution parameter.



Properties

Default | Forced values | Database | Script

Overwrite the fixed/random time property of each step in the script:
Fixed Random [✓ Disabled]

Overwrite the time of each step to be fixed:
Enabled [✓ Disabled]

Time value: 5

Figure 2. SIMACT Properties Box

The second section, called Forced values, classifies options that are used to overwrite parameters defined into the scripts. They are all disabled by default and when activated, they also modify the interpretation of the scripts and the sensors' outputs. The first parameter allows forcing all steps of the scripts to have a fixed or random time value. For a demonstration, it may be useful to test a script by ignoring temporal inconsistencies integrated in it without modifying it by forcing a fixed time value. The second parameter of this tab overwrites the completing time of all steps to the specified value if activated. Of course, random time applies on this static value.

The Database tab obviously includes options that affect the database. For now, there are two parameters but this number will change as SIMACT evolves. First, the database can be disabled and enabled. If disabled, SIMACT will interpret the scenarios and will show the results without saving any events. Second, the path of the database can be modified and as a result, a different database could be used for the interpretation of certain scenarios.

The last tab is named Script, and it includes every parameter influencing the playing of the scenarios that does not fit into previous tabs. An important one is the playing speed. Contrary to the playing speed parameter on the main window of SIMACT, this parameter changes the default value. Consequently, in its next start up SIMACT will make use of this newly defined default speed. The second parameter defines a multiplicative floating point constant that is used to generate a random time value to a step if required. The generated value will be between its base time X up to X*Y where Y is the constant defined here. The last parameter is used to set the default location to search in when opening a script in SIMACT. This parameter habitually does not need to be updated because SIMACT will set it based on the folder most frequently accessed.

2.3. Software Architecture

The software architecture is flexible and adaptable to the specificities of the problems encountered by scientists. It was designed in such a way as to separate code from every dynamic aspect of the software. As we can see in Figure 3, there is a set of classes used to control the 3D canvas. These classes are the only ones with JME code and are strictly used to operate the frame, not to control it. More precisely, there are, in the main system, interpreter classes with the role of understanding what to do and how do it. The first step occurs at the startup of SIMACT; an XML reader module will read an object definition file and ask the 3D control classes to load each of them by searching in the *objects/jme* folder. If it cannot find the objects in *.jme*, it will try several other formats such as *.obj* and *.3ds*. Once the object is found, it is exported as *.jme* for the future launches because the engine performances are a far better with its own type of objects. The second part consists of rendering these objects and keeping a pointer on them to follow references and interactions. A map structure is used for each category of objects (normal, text, etc.). Then, they can be manipulated through the scenarios just by using their unique IDs. Imagine the event "Open the oven door". The command will be read from the script by a XML reader module and sent back to the command interpreter that will finally tell the 3D canvas control classes to open visually the oven door. The interpreter classes will always keep modifications made on the objects in a stack to allow the user to go back in the execution sequence. To define scripts and the objects list, we have used XML because of its readability; look at this simple example: *<action> <object> Oven </object> <open> true </open> </action>*.

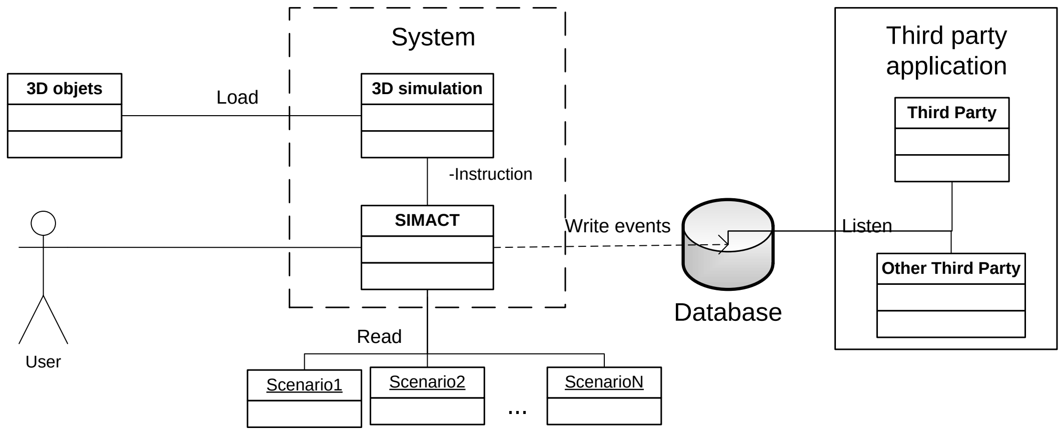


Figure 3. Architecture Representation

The meaning of this simplified example is obvious: it informs the interpreter module that an action will occur, notifies all the affected objects using their unique IDs, and, finally, sets the new disposition. The last part of the sequence consists of accomplishing something useful with these playing scenarios. During playtime, SIMACT saves events in the database but not when moving back and forth in the sequence of steps. It will restart the recording wherever in the sequence the play button is being pressed.

2.2. Technologies for Implementation

The SIMACT simulator was programmed entirely in Java because we wanted to make it open source and portable. The open source community [17] around Java is quite large, and no other languages meet the requirements. Java is also a powerful object-oriented language which is known in computer science to favour code reusability. The swing library was chosen for the GUI (graphical user interface) because it is cross-platform, well documented and easy to use. The 3D part was a little trickier, because Java is infrequently used in game programming. After some research, we found Java Monkey Engine [18], a relatively new promising 3D engine. JME is in a mature state; it is a serious engine that was adopted by the Java community over the last few years. It gave us all the functionalities that we expected for our 3D simulation and more. For the 3D design, we have chosen to use trouble-free software named SketchUp from Google [19]. SketchUp is a modeling tool oriented on house design and indoor accessories. The principal advantage of using SketchUp is certainly the huge 3D object's library maintained on Google’s website by the community around Sketchup. These 3D models are free, and we mostly constructed our environment from that library. In order to give an abstraction layer to the software, we decided to use the popular language XML. It was used for the definition of the three dimensional environment as a simple list of objects (loaded from files on disk) with their basic information such as absolute or relative position in space, unique ID, rotation, scale, etc. XML is also used to define the real case scenarios. Since SIMACT has many options, we provided two useful ways to change the configuration. It can be done by standard text edition in the *.ini* file, or with the GUI menu directly on SIMACT. More details are given about XML and *.ini* in the next section.

3. Parameterization in Text Format

It has been seen previously that SIMACT offers many parameters and options where they can be modified via the graphical user interface. These properties, some more advanced options and many constant values are stored in a simple text file adopting the popular *.ini* file format. In this file, the lines beginning with a semicolon are comment lines, and they are there to give useful advice about each parameter to the user. Word(s) enclosed in square brackets [] are used to define a group of parameters that are used as a unique set by accessors in the program. The constants and variables are found on the lines beginning with a letter and are given a value by using the equality symbol =. The options and parameters in this file are advanced and may modify important features of SIMACT. Therefore, there is a safety copy that allows the user to restore the default parameters in case a mistake is made. Some features in the conf.ini file will be greatly appreciated for users seeking to extract the most efficient customization. It is possible to change the default directories employed in SIMACT. This includes directories for the *.jme* models, *.obj* models, *.3ds* models, the scripts and even for the image files utilized by SIMACT. The default names and locations of files used by the software are parameterizable too, except for the *.ini* file. There are several other constants that can customize the application, but it would take too long to enumerate all of them. They vary from the window size to the font used in the console. The configuration file is not for everyone, but specific adaptation begins here.

3.1. Dynamic Objects Loading

The ability to create new scenarios and to modify the environment without coding in Java constitutes an important feature of SIMACT. The environment is scripted by declaring a list of objects to be dynamically loaded at the startup of the software. Typical object declarations use two markups *<object> [description] </object>*. Critical information, such as unique ID place between *<id> </id>*, must be specified. The ID is the most important tag, and it is used by SIMACT to remember a specific object and act upon it via the scenarios. Another required markup is the name which represents the physical name of the model on the hard drive. The name does not need absolute URI (Uniform Resource Identifier) nor extension because SIMACT will automatically take models placed in the default model’s directory. For example, the markups *<name> oven </name>* will search for a model named *oven.jme* in the directory. If the object cannot be found, the simulator will search in other format directories. It is possible to force the simulator to always take a specific format other than *.jme* by specifying the corresponding extension, but it will dramatically reduce performance as the number of models increases. A relative location could also be used from the default folder, but it is preferable to change the default folder or simply avoid this whenever possible. The name is not necessarily unique: a model could be used for as many object's declarations as wanted. For exemple, if a table has four chairs, each of them should be defined using the same *chair.jme* model. Every other markup describing an object is optional. Initial position can be specified with <*position*> <*x*></*x*> <*y*></*y*> <*z*></*z*> </*position*>. With the position come the markups <*parent*></*parent*> which can be used to specify a relative position from another object. The parent also affects the rotation markup, which specifies a radian rotation of the object: <*rotation*> <*angle*></*angle*> <*axe*></*axe*> </*rotation*>. The scale <*scale*> </*scale*> markup is also influenced by the parent definition and is utilized to adjust the scale in comparison with other objects. The visibility <*visibility*> *true|false* </*visibility*> markup is true by default and can be used to tell SIMACT to not initially render one or many objects. Besides the normal <*object*> markups, a declaration can be enclosed in the <*static*> *[description]* </*static*> markups. The only difference with the normal declaration is that SIMACT will prevent the object from further modifications during the execution. The <*sensor*> *[description]* </*sensor*> type forces the use of a parent plus a <*type*> tag to robotically insert events in the database when an action occurs over his parent. There are the <*text*> *[description]* </*text*> and <*text3d*> *[description]* </*text3d*> that are used to define a label directly on the 3D canvas. The first one does not move with the camera around the environment because it is position in a 2D (Cartesian) plane. The text objects do not have a <*name*> tag but instead have a textual value that can, of course, be changed by the scenarios. Finally, there is a special textual object whose ID is *Time*. Only the visibility attribute can be changed. This object shows a real time clock on the 3D environment that systematically updates during each engine loop.

3.2. XML based Script Creation and Edition

Considering correct declarations in the object file, new scenarios become very simple to define using XML and the unique ID provided with each declaration. A scenario is described between <*script*> </*script*> markups and needs to have a short title written between <*title*> </*title*>. The script is composed of steps (actions) <*step*>, which require a small unique <*name*> </*name*> of 20 characters or less and optionally a short description <*description*> </*description*> of less than 200 characters. If a field is too long, it will just be ignored, and the first part will appear correctly. The step can follow a tree structure having the important steps divided into smaller one and so on. For exemple, the high order step “make a toast” is composed of the atomics steps “take a slice of bread”, “put it in the toaster” and “wait few minute”. The step markup can contain two attributes. First, the *time* attribute represents the time required to simulate a step in seconds with a dot for the milliseconds like this: 0.000. Second, the *fixed (true|false)* attribute tells SIMACT to take always the specified time to execute the step (true) or to add a random value to simulate an imprecise human behavior, a cognitive deficit, or simply a mistake. These attributes are optional, and if they are not specified, they will respectively take the basic default values *1* and *true*. On top of that, a step could raise events that will be defined in <*before*> </*before*> or <*after*> </*after*> markups. It is up to the user to determine which case is the best for his events. For example, pressure mat activation must be set “before” in the execution of the activity “Go to the microwave and open the door” but the visual action of opening the door must take place after. The general rule in the process of designing scenarios is that most events should take place before the step and only those representing consequences of the action are to be placed in <*after*></*after*>. Following this simple rule, the "before" section represents the prerequisites, and the step is only an observation of these events.

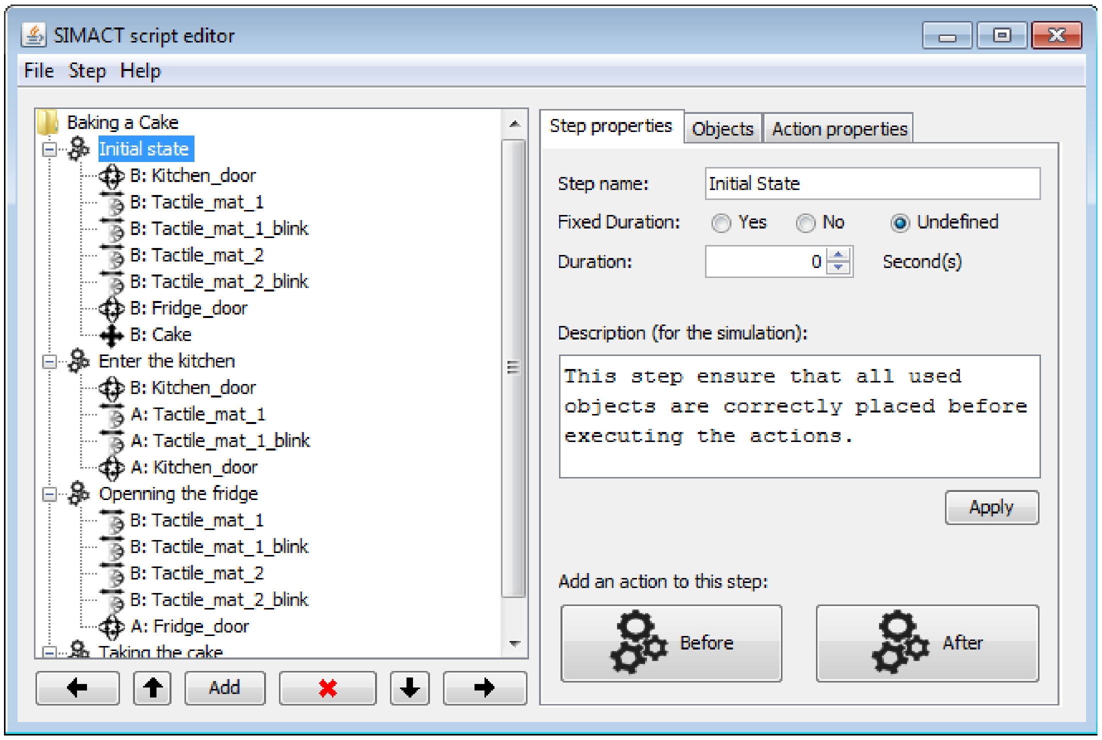
Another important rule is that adequate fragmentation must be done to ensure unambiguous steps. Above all, only the leaf nodes are meant to have action markups. There are a few predefined actions that can be used depending on the object type. Even if they are all distinct, each of them has two markups in common. The first one is the unique ID of the object affected by the event, and the second is a short and optional description for the database insertion. If the <*database*></*database*> markups are omitted, the action will not produce an event in the database. The first action that can be made upon an object is a translation. It is a simple action that moves the object to <*x*></*x*> <*y*></*y*> <*z*></*z*> coordinates (<*text*> type will only use x and y). The second action is the rotation and use <*axis*></*axis*> and <*angle*></*angle*> markups to perform a rotation in radian unit on an object. Thereafter, the markups <*visibility*></*visibility*> and <*scale*></*scale*> are respectively used to alter the visibility and the scale of an object. They use exactly the same syntax as in the object definition file. To conclude, there is a special action for the textual objects that makes possible the dynamic adjustment of its value: <*text*></*text*>. This could be very useful in implementing a textual thermometer or a barometer, thus dropping the barriers as much as possible for the users.

4. Visual Script Editor

We recently began working on a visual script editor to improve the capacities to write good scenarios in a limited amount of time. This section will present an overview of the present and future features of this new editor that will come with SIMACT. We opted to use Java combined with the Swing library to build our editor. This choice was made for similar reasons compared to SIMACT, but also to maintain consistency with the principal application. Even if everything seems represented graphically, the editor follows the same guideline of SIMACT by using the XML language. Therefore, every scenario defined directly in XML could be loaded directly in the editor. The same goes for the scripts built with the editor: they can be edited in text mode. Basically, the scripts from the editor are not different from the others; they are just easier to create and modify. The main screen of the editor, presented in Figure 5, is divided into two important sections. The first section represents the current scenario in editing mode. This section never changes except for its contents of course. The scripts are represented, likewise, by a folder tree where the root is the script name and every other non leaf node is a step. The leaves of the structure are always actions that occur during a certain step (the parent node). These actions are represented by particular icons that will be described later. But, there are two important icons represented in Figure 4 that still need to be understood.

Figure 4. Icons for Steps Representation

The first one is used by the root of the script and by every container step. The second replaces the first when a step has at least one action. The other section is a tabbed pane which is controlled by the editor while the user builds his scenario. The right pane is selected automatically without the intervention of the user. In fact, the user cannot change the tab itself since it is done by the program.



Baking a Cake
- Initial state
  - B: Kitchen_door
  - B: Tactile_mat_1
  - B: Tactile_mat_1_blink
  - B: Tactile_mat_2
  - B: Tactile_mat_2_blink
  - B: Fridge_door
  - B: Cake
- Enter the kitchen
  - B: Kitchen_door
  - A: Tactile_mat_1
  - A: Tactile_mat_1_blink
  - A: Kitchen_door
- Openning the fridge
  - B: Tactile_mat_1
  - B: Tactile_mat_1_blink
  - B: Tactile_mat_2
  - B: Tactile_mat_2_blink
  - A: Fridge_door
- Taking the cake

Step name: Initial State

Fixed Duration: Yes No [✓ Undefined]

Duration: 0 Second(s)

Description (for the simulation):

```
This step ensure that all used
objects are correctly placed before
executing the actions.
```

Figure 5. Visual Script Editor

4.1. Script Design

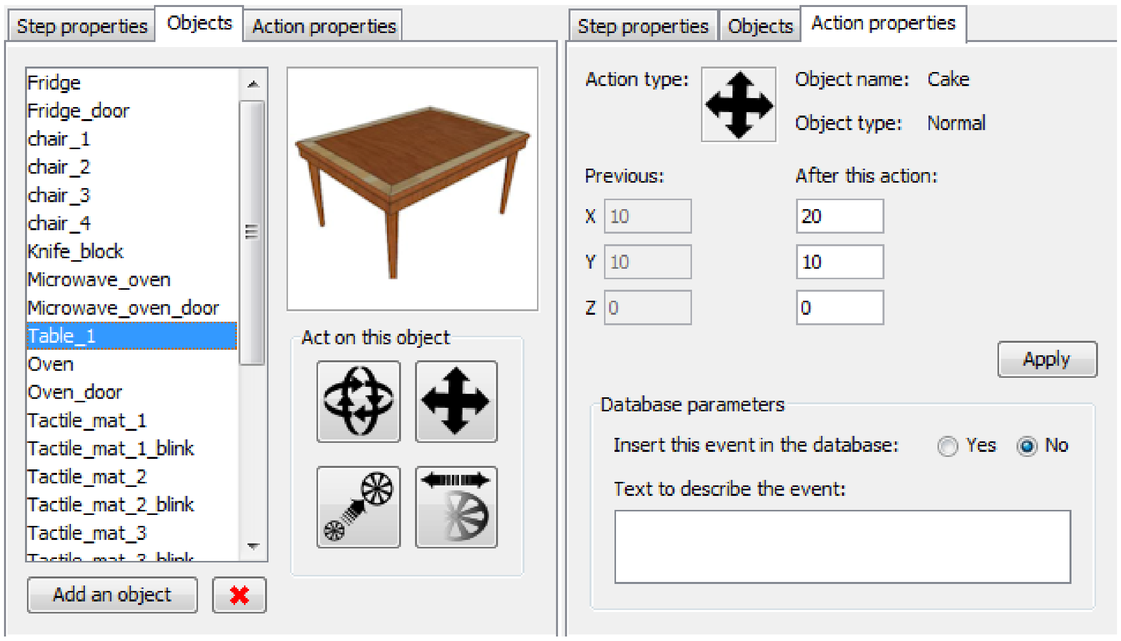
In the previous section, we talked about the features of this brand new visual editor. This section will focus on the design of scripts by explaining everything in detail. The *File* menu is similar to any ordinary file menu in common application. It is used to create, open, close and save the scenarios. When the *New Script* button is pressed, a text box window appears to ask for the name of the script. The name must be given for the creation process to succeed. The new script will be created with the root node right after pressing the ok button. This script is already valid and could be saved and executed right away. Nevertheless, to get something interesting some step should be added first. To do so, six buttons under the tree zone are available. The *Add* button creates new nodes as children of the currently selected node. If no node has been selected, they will be just created as children of the root. These nodes are then deleted by selecting them and pressing on the ✖ button. A confirmation box is always shown when pressing on this button, because suppressing a step also suppresses all its child nodes, even the action nodes, with no way to go back. The up and down arrows are used to change the sequence order of a selected node. The highest nodes are the first to be executed. All the children of a node must be interpreted before passing to the node beneath. The left and right arrows are used to change the level of a step in the tree structure. Since it is a tree, it will be impossible to level up a node at the root level. There can be only one root, and it is also to create this root that the name of the script is asked for during its creation process. The left arrow is in charge of leveling up a node. In contrast, the right arrows make a selected node a child of the first upper node on the same level. If there is no upper node on the level, the command will be ignored. Finally, it is important to remember that it is a good idea to create a step to initialize all the manipulated objects of the scenario to ensure that they will always be in the expected state and position before beginning the script interpretation.

Now, the freshly created nodes will need some information to acquire a real sense in the scenario. Clicking on a step in the tree view will automatically select the "Step properties" tab and will fill the fields with default information. The options are really the same here as when editing in text mode. You can choose a name for the step and write a short description with the only use of bringing some handy tips during the simulation. Fixed or random time duration could be also defined. Negative values are not allowed, but decimals can be use with a dot. After making all these modifications, the *Apply* button must be pressed to erase permanently the old data in order to be able to retrieve the new data next time that the node is selected from the tree view.

4.2. Action Definition

The other part of the process of creating a scenario consists of associating actions to the key steps. The two biggest buttons in Figure 5 are there to do it. Obviously, the first one will set an action before the step and the second after. Clicking on one of these buttons will give access to the *Objects* tab that can be seen on the left side of Figure 6. From this point, the action can be seen without icons in the tree view as a child of the selected step. The label of the action will follow a simple code. The first letter, B or A, indicates if the action is executed before or after, and the name after the colon is the ID of the affected object (null if undefined). The *Objects* tab is, in fact, a little special. It does implement a list of objects that are dynamically loaded from the same object file that SIMACT uses to build its environment. This feature is really important for the visual editor because it permits the user to be sure that his object is correctly defined for SIMACT and for the scripts. Besides improving robustness, there are also two interesting buttons that provide the functionality to manage the objects list graphically. This part is really basic but fully replaces the XML edition. The ✖ button suppresses an object, and pressing it will show a confirmation box to make sure that the user is aware that once an object is deleted it cannot be restored. When an object is selected in the list, a preview is shown in a square in the top right corner. To work, a preview picture needs to exist in the corresponding folder of the object. In a future implementation, we expect SIMACT to autonomatically take a screen shot of new objects when no picture is existing. We decided to use pictures instead of loading the real 3D model for performance purposes, and because it was unnecessary. The textual and sensor objects are also displayed in this list. So, it happens that some of them (actually, all textual objects) don't have a preview picture. In that case, the picture frame shows a default image with the following message in it:"Preview not available". Once the object is chosen, an action can be performed between the four represented by these icons: ✥ rotation, ✚ translation, ⤢ scale change and ◐ visibility change. When textual objects are selected the rotation button is altered by **A** button that is used to change the value of the text label. A click on one of these buttons will send the user directly to the last tab *Action properties* and will add the corresponding icon beside the action in the tree view.

There is not much to say about the *Action properties* tab. It is used to enter the parameters of the chosen action type. These parameters are the same as those previously described in the XML section. The important characteristic here is that this tab change depends on the kind of action chosen. The right side of Figure 6 shows an example of what it may look for a translation action. The top section shows the icon corresponding to the type of action, the affected object name and its type. The middle part will expose all the matching parameters plus the preceding one defined on this object. The previous parameters displayed are the last modification of this type on the object before this action. For exemple, if the script performs a first translation on a cake from the origin to the point (10,10,0), the next translation will show the coordinates (10,10,0) as previous parameters. If no translation has been done before, the initial position defined in the object declaration will be used. The bottom section of this tab does not change with the type of action. It is always there with the same field. It is used to set a recording event which means an event to be saved in the database by SIMACT. The last step would be to press the *Apply* button to save the modifications.



Fridge
Fridge_door
chair_1
chair_2
chair_3
chair_4
Knife_block
Microwave_oven
Microwave_oven_door
Table_1
Oven
Oven_door
Tactile_mat_1
Tactile_mat_1_blink
Tactile_mat_2
Tactile_mat_2_blink
Tactile_mat_3

Action type:
Object name: Cake
Object type: Normal

Previous:		After this action:
X	10	20
Y	10	10
Z	0	0

Insert this event in the database: Yes [✓ No]
Text to describe the event:

Figure 6. Objects and Action Properties Tab

5. Experiment, Trials and Scenarios

In its current beta state, the SIMACT application is working and offers a variety of functionalities and many options. Functionality is surely important, but one of SIMACT's most significant contributions to the community is to come with a well defined set of scripts of activities, based on real case scenarios extracted from clinical trials. To achieve that, we designed a simple experiment based on a well established cognitive test [20] using a few activities that are adapted to the needs of scientists. We then used this experiment to prepare two sets of clinical trials with real subjects.

5.1. Designing an Experiment Meeting Research Needs

To design our experiment, we took into account several basic needs of smart home scientists [21]. First, they need real data about activities that are important in the daily life of an elder, such as meal preparation. Knowing that a lot of examples in the literature imply kitchen activities (preparing tea or coffee, cooking, etc.) [6], it is surely a good lead. Secondly, scientists need models of activities that are representative, but that are also easy to experiment with, in a research context. Therefore, the proposed activities models must be simple and must include a sufficient but limited number of steps. These activities should also be achievable in a reasonable amount of time. Finally, scientists need to possess data about different activities least few, which can cover diverse situations. To meet all these needs, we chose to base our design on a well established cognitive test: the Naturalistic Action Test

(NAT) [5]. This test is conceived to evaluate the performance of the individual while performing common kitchen tasks. The NAT has purposely been designed to assess patients with neurologic afflictions. Originally, the NAT features three activities. Only two of them have been selected for several reasons: (I) the test is composed of one common activity and two less common ones, so we just selected one uncommon; (II) the chosen uncommon activity has been selected for its expected difference in execution depending on the participant’s characteristics; and (III) the patient's evaluation time is reduced.

The first selected activity is “preparing coffee and a toast”. Figure 7 illustrates a step by step model of the possible ways of achieving this activity, designed on the basis of the results of the clinical trials. The purpose of giving this model is to help researchers determine whether the performance of the activity is normal or not (for instance, the order of steps) and to determine the average variation among the control population. The following model can be used as a baseline for both comparison and assistance in the development of cognitive aids. The activity is composed of two sub-tasks that can be interleaved: “prepare a cup of coffee” and “make a toast”. This activity constitutes a really good choice because it is composed of

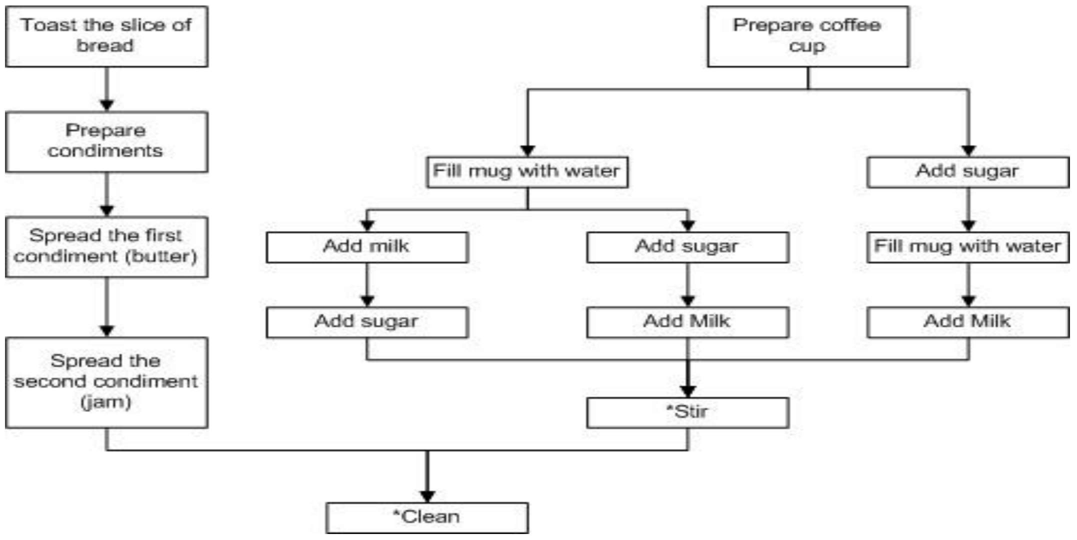


Figure 7. Model of the Activity: ***preparing coffee and making a toast***

two sub-tasks, several interleaved steps, it can be done in less than 10 minutes, and it is similar in nature and size to a lot of smart home literature examples such as preparing tea [21], cooking pasta [6] and washing hands [9].

The second chosen activity is “wrapping a gift”. This activity is less common but brings a different activity context in the collected data. In the original version, all three activity scores were combined to give the complete NAT score; the score is based on the completion of the task and the kind and number of mistakes made. The initial test also scores lateralized errors, but the adapted version we used did not use these scores because they don't reflect the kind of impairment (cognitive deficit) of the clinical population we want to work with.

5.2. Experimental Approach for Gathering Useful Data

Our experimental approach to gather useful data can be divided into two phases. The first phase consists of conducting clinical trials using the designed experiment with an adequate set of normal subjects. The objective of this first phase is to gather relevant data about the average performance of each activity: conventional completion time for each step and for completing the activity, usual step's order, errors committed, hesitations, differences in sex and ages, etc. Scientists will then be able to use the data to calibrate their algorithms and for comparison with abnormal observed behavior. The second phase consists in conducting the same experiment with a similar size group of cognitively impaired people. The goal of this second phase is to gather specific data concerning the erroneous performance in the realization of these activities caused by cognitive troubles. This will allow researchers to better understand the behavior of such patients and to have concrete tools helping them to adapt their assistive technologies to their particular needs for that type of patients. In the subsequent section, we will detail each phase of our approach.

5.3. Phase 1: Experiment with 20 Normal Subjects

For the realization of this phase, we obtained the approval of an ethics committee and recruited a set of subjects on a voluntary basis. Publicity was taken out in regional newspapers, as well as on the campus. The participants had to be aged between 18 and 77 years old. The only inclusion criterion was the absence of any type of cognitive impairment. This impairment could either be a psychiatric or neurologic issue or the use of prescription drugs. The participants were also asked to refrain from any alcoholic or drug use, for the 24 hours preceding the test. A total of 20 participants were selected to perform the two activities.

The experimental protocol went as follows. The participant sat on a chair; all the material needed for the completion of the activity was laid on the table in front of him. The participant was free to use anything he wanted to complete the task and could do so in any order. At first, the participants were asked to prepare a cup of coffee with sugar and milk, and a toast with butter and jelly. Secondly, participants were asked to wrap a gift with the proposed material. There was no time limit. The participants had to quit the room while the assistant was preparing the set for the second activity. Each trial was recorded on video and timed. The videos were framed so as to preserve the anonymity of the participants; the faces are not shown on the video. We developed a scoring sheet to record the order in which the different steps of the task were performed. Following the completion of the test, the order and time of each step were individually processed. Statistical analysis (modal and mean analysis, mean comparison and correlation), was used to determine the order of the steps in the different models and to determine if any sub-group difference was relevant, for example, if there were any differences depending on the gender or the age of the participant.

5.4. Phase 2: Experiment with 20 Alzheimer’s Patients

To accomplish the second phase of our experimental process, we signed a formal collaboration agreement with the regional rehabilitation center Cleophas-Claveau of La Baie (QC), Canada. This center is able to provide us with an adequate group of cognitively impaired people for our experiment, mostly Alzheimer’s patients at moderate stages. We already have been approved by the ethics committee for a clinical experiment with a group of 20 people suffering from Alzheimer’s disease. The multidisciplinary team that will manage this second phase is composed of a neuropsychology researcher, two smart home researchers, and several graduate and undergraduate students in psychology, computer science and engineering. The recruiting was done at the beginning of 2010 and the experimentation with Alzheimer’s patients began in June 2010. Things are going smoothly, and so we are expecting to be able to conclude this phase by September 2010.

5.5. Open Source Database coming with SIMACT

The main purpose of our entire project is to provide adequate experimental tools to researchers that work in the field of smart homes. To accomplish that, we built a database containing the details of all our experimental results. It includes the execution time for each activity, for each patient, and for each step, the errors made, the compiled results and, more importantly, a set of pre-recorded real case scenarios based on our experiment. When the second experimental phase will be completed, we will include the data and scenarios about both normal and impaired subjects. At this point, we will be able to release our tools on the Internet to be freely used by the smart home community. We are convinced that our work will provide good quality real-case scenarios, with the tool for the benefit of the scientists.

5.6. Testing a Recognition Algorithm with SIMACT

In keyhole activity recognition [6], the temporal variables may influence the result and the correctness of an activity completion. Unfortunately, most approaches do not take into account the temporal aspect, or when they do, they only do it in a very limited way. This time issue is fundamental, because some kinds of incorrect behavior can be detected by monitoring this aspect. For instance, all the steps of an activity may have been correctly ordered but if one of them has too short or too long duration, it may be unacceptable. Let us illustrate this by a small example. Suppose that a person, named Peter, has correctly executed all the steps of making a cake in the right order and in the appropriate way. Peter is now baking the cake, and the system can just conclude that the only possible next step is “TakeOutTheCake”. Therefore, even if Peter lets the cake bake for seven hours, no assistance will be provided if the time is not taken into account. This duration is not only erroneous, but it is also a threat to his safety. Another example of the need of taking into account the temporal aspect can be observed when the patient performs several tasks simultaneously. In this situation, the lapse of time between two steps can be the key to understand what he is doing. Knowing these facts, it seems obvious that a good model for recognition shall take into account the time factor to minimize the number of plausible hypotheses and to identify more realization errors. In this context, we have addressed this issue by extending a previous recognition model [6] to integrate non-symbolic temporal constraints [22]. We then used the SIMACT tool to test it.

Our new temporal recognition model is based on the work of Weida [23], which uses Description Logic formalism to encode low level actions that are represented by concept and defined by role. When encoded, these activities are modeled with the help of a temporal graph structure in which actions correspond to nodes and temporal constraints to edges. We have improved this model by introducing a new temporal structure. This structure is a logical evolution from the previous works with an enhanced precision capability. It allows us to take into account the quantitative aspect of the duration and the delay between steps in the execution of an activity. Our objective was to improve the performance of Weida’s model to detect more anomalies and to perform a faster elimination of the inadequate recognition hypotheses.

The experimentation phase for this extended model was designed using SIMACT. The first step was to implement the two versions of the temporal recognition algorithm: the original algorithm of Weida and our new augmented version. Then we played the real case scenarios included in the SIMACT database, which come from results extracted from the clinical trial with human subjects. We added some temporal mistakes (duration and delay) in the scenarios. The objective was to compare the capabilities of the two models in recognizing correct and incorrect plans. Figure 8 shows the general architecture of this implementation. All scenarios were played by SIMACT, which inserts each step or action one after the other in the database until it is completed. At the same time, a module from the testing application reads the events and sends them to an interpreter. The interpreter is the part that implements the Weida algorithm and our own. When it receives a new step, it compares the data with the possible plans in its database by considering the temporal constraints. To compare the performance, we computed the amount of errors detected by each algorithm.

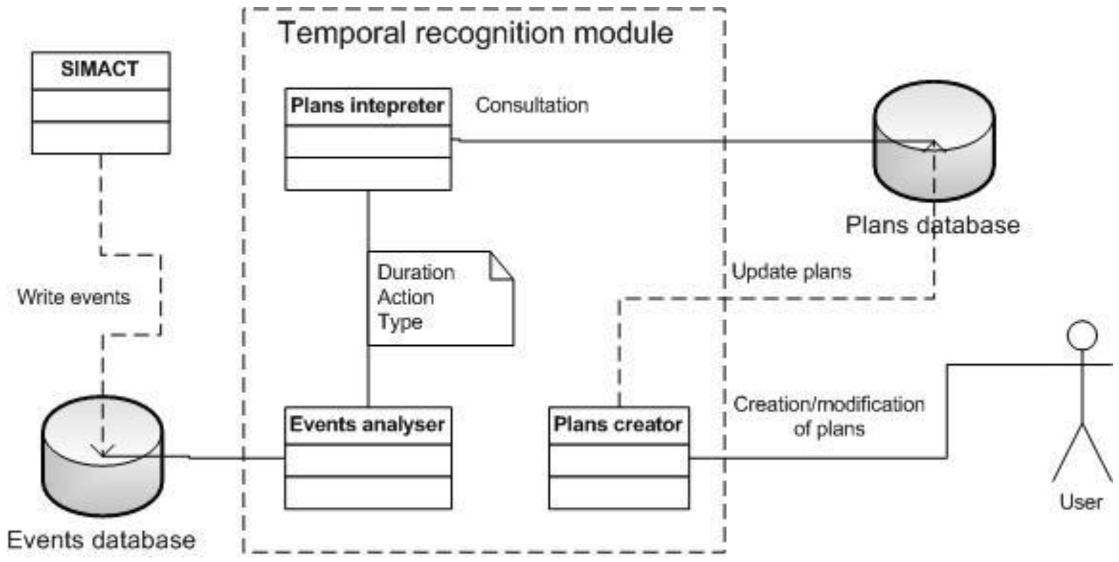


Figure 8. Software Architecture of the Experimentation

In this experimentation, we were able to completely build our algorithm and easily test it using only the simulation tool and its database. SIMACT gave us the opportunity to manage the test of the two algorithms independently. Moreover, we have been able to do the same tests twice with identical errors and actions executed by our virtual patient. This has increased the precision of the results of the comparison between the two versions of the algorithm rather to clinical trial where the patients never execute the activity the same way twice. This experimentation has shown how SIMACT can help researchers needing a cost effective way to try their algorithms and artificial intelligence (AI) models for smart homes. The details about this new recognition model and the complete results of this experiment will be the object of another publication later in 2010.

6. Related Works

Over the last few years, several projects [12][13][14][15][16] tried to address the issue of developing a simulation tool for smart home experimentations. This section will discuss as clearly as possible each of these approaches by presenting their advantages and disadvantages.

6.1. CAST Simulation Toolkit

CAST is an important project in the history of smart home simulation [12]. The project was oriented on the problem of context acquisition. Thus, it supports the acquisition of reusable contexts and can generate users and devices in a virtual home domain. It uses a certification/authorization protocol for the acquisition of contexts so the application devices could safely use it. Moreover, the architecture of the software was built on a scenario-based approach. However, CAST was implemented with proprietary technology Flash MX, regardless of its weakness in supporting free user control of the environment. The CAST project is not adapted to our situation and obviously cannot answer the needs we talked about in this paper.

6.5. ViSi from A. Lazovik & al.

Lazovik proposed in 2009 a new 2D tool [13] with a flexible GUI in order to virtualize a complete smart home environment. His architecture was based on the web services' model, so that it could be used in conjunction with any language or hardware supporting the web service stacks. The objective of this tool was to provide a full featured simulation of any possible domotic scenarios. However, this tool is mainly focused on the planning aspect of activities of daily living and does not take into account important elements of the recognition process, such as gathering all the details about the inputs of sensors and providing a simple and comprehensive interface to interrogate them in real time. The simulator was only slightly documented, and no details were given about the implementation of the scenario. Besides, this simulator was directly implemented within Google SketchUp [19] scripting capabilities in the Ruby language. That does not mean it is bad, but it makes the simulator software dependent on SketchUp. The simulator was not well adapted to test recognition algorithms, because the client part of the software was mainly focused on high level goal definitions. So it was more like a third party application to simulate a user in the environment. The planner was the part which could have been personalized to define a recognition algorithm, but no such functionality was specified in the paper.

6.5. Park's Simulator: CASS

Park recently proposed another simulation tool named CASS (Context-Aware Simulation System for Smart Home) [14]. This tool aims to generate the context information associated with virtual sensors and devices in a smart home domain. By using CASS, the self-adaptive application developer can develop rules based on simple sensor condition and resulting action on a device. For instance, a rule could be "If humidity of a room reaches 60%, activate dehumidifier" and would be defined by the condition 60% on humidity sensors and the resulting action of activating the dehumidifier. One of the main features of CASS is the ability to automatically detect rule conflict in context information and determine optimal sensors and devices in a smart home. A user could also add and delete virtual sensors. As we can see, CASS is oriented to focus on sensors' distribution problems more than providing a means to experiment with recognition algorithms. Moreover, there are no scenario capabilities, and the user's defined rules cannot be very complex. Finally, CASS is very good but is not adapted to our situation.

6.5. ISS System from H. Nguyen & al.

A recent simulator was presented by H. Nguyen & al. [15] at the ICUT'08 conference. This simulator is similar to CASS in terms of the covered area. It is really simple to use, with an uncomplicated user interface which includes a 2 dimensional graphical simulation of the intelligent house. However, it is not flexible and does not allow a lot of customization. It is used to simulate simple rules, as for CASS, such as automatically opening the TV when the patient sat on the sofa. The scenarios' implementation seemed interesting, but no details are given about them. On the other hand, this simulator supports many new ideas such as the importance of the simulation tools and the use of RFID tags in smart homes.

6.5. Smart House Simulation Tool by Krzyska

The closest project to our proposal is the smart home simulator proposed by Krzyska [16]. Her team developed a smart home editor providing the functionalities needed to build a customized smart home from scratch. The user turns himself into an architect by defining walls, doors and sensors. Furthermore, there are scripting functionalities similar to ours, allowing one to simulate activation of the sensors. However, a limitation of her approach is that scripting does not allow simulating complex real case scenarios because it was only designed for very basic experimentations. Moreover, the simulation is done in a 2D frame made with the Swing library for Java, and it lacks clarity. On the contrary, the SIMACT tool specifically focuses on the integration of real case scenarios and is more user-friendly with its animated 3D frame.

In conclusion, none of these previous contributions answered the existing needs of the activity recognition research field, that we emphasized in the present document. All these approaches were developed in a research setting without having their simulation tool ever being proposed freely to the community in an open source environment allowing for further enhancements. They are less flexible and adaptable and not oriented on activity recognition. Their scripting capabilities, if they exist, are limited. Besides, SIMACT displays the virtual habitat within a three dimensional graphic canvas build from dynamic objects' list. Finally, one of the most significant contributions over previous work consists providing a complete set of adequate real case scenarios with the tool.

7. Conclusion and Perspective

People will get older during the 21st century because of a birth rate decrease combined with a longer life expectation [1]. This reality will be challenging, knowing that we already lack qualified caregivers. In addition, cognitive disease will continue to ravage our society, amplifying the problem. Therefore, according to the 2008-2009 report of the Canadian Alzheimer's Society, 500 000 people are currently affected by the disease or a related disease in Canada and that number should reach 700 000 individuals in only 10 years [2]. Smart homes constitute a viable avenue of solution [4] to this growing disease for the sake of giving more autonomy to the victims and to slow down the progress of the disease. To do so, it is important to be able to recognize activities of daily living composed of simple actions, and events output produced by these actions. However, research and development in this field requires complex infrastructures and a lot of money that many scientists cannot afford. Furthermore, it may be very hard to get access to real patients in order to conduct adequate experiments. Simulation tools constitute, therefore, a very promising and affordable alternative [14] that could be reused at any time. In this paper, we demonstrated the huge potential of SIMACT, a new user-friendly 3D animated simulator offering a simple interface. SIMACT will be freely offered as an open source package to the scientific community and to everyone researching in the field of smart homes. The simple, likewise, video player features allow the software to be used by scientists other than computer science experts. We demonstrated how the XML technology was exploited in order to separate the 3D environment from the Java source code. Moreover, XML was used to build an abstraction layer for the scenarios. Thenceforth, the introduction of a new graphical editor to create scenarios and manage 3D models was meticulously detailed. SIMACT could be used by anyone, and it can be easily adapted to individual needs. SIMACT even has a full range of options and parameters to faithfully render most of the simulation. In addition, we talked about our method to provide a large set of pre-defined real case scenarios to allow scientists to test and compare their recognition algorithms with robust, well established scripts.

The next step toward the development of the SIMACT project is to finish our study with Alzheimer patients and compile the results in the database. With the data, we will design the second part of the scenarios to be included within the package. Thereafter, we will push SIMACT on the Internet to make it available to download. This phase will include the tricky part that consists of publicizing SIMACT within the scientific community. As we are very enthusiastic about this project, we have many more ideas to extend it to the next level. We plan to define other three dimensional environments in addition to the kitchen, adding new rooms. There is also a plan to support more than one scenario at the time in the software and to allow real time interaction by command via the user console area. Finally, SIMACT will be used in future by students of the LIARA (and hopefully of other laboratories) to test their work and improve their efficiency.

Acknowledgements

We would like to thank our main sponsors for their financial support: the Natural Sciences and Engineering Research Council of Canada (NSERC), the Quebec Research Fund on Nature and Technologies, the Canadian Foundation for Innovation (CFI) and our University. We also thank to our psychologist partners Julie Bouchard, Audrey Potvin and Hugo Laprise. Finally, special thanks to the CSSS (readaptation center) of La Baie, which provides us with help, tips, and access to patients.

References

[1] J. Diamond, “A report on Alzheimer disease and current research”, Technical report, Alzheimer Society of Canada, **(2006)**, pp. 1-26.
[2] Annual report, Alzheimer Society of Canada, **(2008-2009)**, pp. 1-12.
[3] World Alzheimer Report 2009, Alzheimer’s disease International, **(2009)**, pp. 1-18.
[4] C. Ramos, J. C. Augusto and D. Shapiro, “Ambient Intelligence: the Next Step for Artificial Intelligence”, IEEE Intelligent Systems, vol. 23, **(2008)**, pp. 5-18.
[5] H. Tibben and G. West, “Learning User Preferences in an Anxious Home, Smart Homes and Beyond”, Proc. of the 4th International Conference On Smart homes and health Telematics (ICOST’06), Milwaukee, USA, **(2006)** December 10-13, pp. 188-195.
[6] B. Bouchard, A. Bouzouane and S. Giroux, “A Keyhole Plan Recognition Model for Alzheimer's Patients: First Results”, Journal of Applied Artificial Intelligence (AAI), Taylor & Francis publisher, vol. 22, no. 7, **(2007)**, pp. 623–658.
[7] S. Carberry, “Techniques for plan recognition”, User Modeling and User-Adapted Interaction, vol. 11, no. 1–2, **(2001)**, pp. 31–48.
[8] K. Haigh, L. Kiff and G. Ho, “The Independent LifeStyle Assistant: Lessons Learned”, Assistive Technology vol. 18, no. 1, **(2006)**, pp. 87–106.
[9] J. Hoey, A. von Bertoldi, P. Poupart and A. Mihailidis, “Assisting persons with dementia during handwashing using a partially observable Markov decision process”, Int. Conference on Vision Systems (ICVS’07), Best Paper award, **(2007)**, pp. 89-99.
[10] D. J. Patterson, H. A. Kautz, D. Fox and L. Liao, “Pervasive computing in the home and community”, in: Pervasive Computing in Healthcare, J.E. Bardram, A. Mihailidis and D. Wan, eds., CRC Press, **(2007)**, pp. 79–103.
[11] P. Roy, B. Bouchard, A. Bouzouane and S. Giroux, “Challenging issues of ambient activity recognition for cognitive assistance”, Ambient Intelligence and Smart Environments, IGI global, F. Mastrogiovanni and N. Chong Editors, (in press **2010**), pp. 1-25.
[12] K. InSu, P. HeeMan, N. BongNam, L. YoungLok, L. SeungYong and L. HyungHyo, “Design and Implementation of Context Awareness Simulation Toolkit for Context learning”, IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, vol. 2, **(2006)**, pp. 96-103.

[13] E. Lazovik, A. C. T. den Dulk, M. Groote, A. Lazovik and M. Aiello, "Services inside the Smart Home: A Simulation and Visualization tool", In Int. Conf. on Service Oriented Computing (ICSOC-09), **(2009)**.
[14] J. Park, M. Moon, S. Hwang and K. Yeom, "CASS: A Context-Aware Simulation System for Smart Home", 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007), **(2007)**, pp. 461-467.
[15] T. V. Nguyen, H. A. Nguyen and D. C. Nguyen, "Development of a Context Aware Virtual Smart Home Simulator", The 3rd Internationnal Conference on Ubiquitous Information Technologies and Applications (ICUT), **(2008)**.
[16] C. Krzyska, "Smart House Simulation Tool", Master thesis, Informatics and Mathematical Modelling, Technical University of Denmark (DTU), **(2006)**.
[17] M. Refaat Nasr, "Open Source Software: The use of open source software and its impact on organizations", Master Thesis, Middlesex university, 129 pages, **(2007)** June.
[18] M. Powell, "Java Monkey Engine (JME)", http:// www.jmonkeyengine.com, **(2008)**.
[19] Google ©, SketchUp 7, http://sketchup.google.com/, **(2009)**.
[20] M. F. Schwartz, M. Segal, T. Veramonti, M. Ferraro and L. J. Buxbaum, "(NAT) The Naturalistic Action Test: A standardised assessment for everyday action impairment", Neuropsychological Rehabilitation, vol. 12, Issue 4, **(2002)** August, pp. 311-339.
[21] H. Pigot, J. P. Savary, J. L. Metzger, A. Rochon and M. Beaulieu, "Advanced technology guidelines to fulfill the needs of the cognitively impaired population", in 3rd Int. Conference on Smart Homes and Health Telematic (ICOST'05), **(2005)**, pp. 25-32.
[22] R. Dechter, I. Meiri and J. Pearl, "Temporal Constraint Networks, Artificial Intelligence", vol. 49, **(1991)**, pp. 61-95.
[23] R. Weida, "Terminological Constraint Network Reasoning and its Application to Plan Recognition", Columbia University Department of Computer Science, **(1993)** September 2.