Knowledge and Information Systems (2020) 62:3881–3910
https://doi.org/10.1007/s10115-020-01476-x

REGULAR PAPER

Modeling, learning, and simulating human activities of daily living with behavior trees

Yannick Francillette[1] · Bruno Bouchard[1] · Kévin Bouchard[1] · Sébastien Gaboury[1]

Received: 25 June 2019 / Accepted: 25 April 2020 / Published online: 1 June 2020

Abstract
Autonomy is a key factor in the quality of life of a person. With the aging of the population, an increasing number of people suffers from a reduced level of autonomy. That compromises their capacity of performing their daily activities and causes safety issues. The new concept of ambient assisted living (AAL), and more specifically its application in smart homes for supporting elderly people, constitutes a great avenue of the solution. However, to be able to automatically assist a user carrying out is activities, researchers and engineers face three main challenges in the development of smart homes: (i) how to represent the activity models, (ii) how to automatically construct theses models based on historical data and (iii) how to be able to simulate the user behavior for tests and calibration purpose. Most of recent works addressing these challenges exploit simple models of activity with no semantic, or use logically complex ones or else use probabilistically rigid representations. In this paper, we propose a global approach to address the three challenges. We introduce a new way of modeling human activities in smart homes based on behavior trees which are used in the video game industry. We then present an algorithmic way to automatically learn these models with sensors logs. We use a simulator that we have developed to validate our approach.

Keywords Behavior tree · Machine learning · Visualization · Human activity modeling

✉ Yannick Francillette
yannick.francillette@uqac.ca

Bruno Bouchard
bruno.bouchard@uqac.ca

Kévin Bouchard
kevin.bouchard@uqac.ca

Sébastien Gaboury
sebastien.gaboury@uqac.ca

1 LIARA, Université du Québec à Chicoutimi, Saguenay, Canada

1 Introduction

The aging of the populations all around the world has an impact that increase the number of people with physical or mental disabilities [59]. Consequently, many of these elderly people suffer from some kind of loss of autonomy. The notion of autonomy can be characterized as the ability of an individual to perform his activity of daily living (ADL). A decreased autonomy is characterized by having difficulties in carrying out common ADLs such as preparing a meal, washing, and dressing [38]. More precisely, when functional abilities are affected, it may prevent the efficient performance of certain important tasks. Therefore, the loss of autonomy can be described as an imbalance between the activities that someone should or would like to carry out and their functional capacities to perform them adequately. At a certain level, there is even a need for constant assistance to the person to ensure its safety and comfort.

The main issue related to this reality is that we have limited capacity and capability, as a society, to support all these persons in need. Of course, we could just build much more specialized care centers for the such people to live in, designed to accommodate their needs, but at what cost? From an economical point of view, it would be terribly expensive. On the human side, the segregation in a care center is demonstrated to be damaging for the autonomy, the dignity, and the well-being of a person [11]. Indeed, most people prefer to stay at home because they feel more comfortable and more autonomous. This choice involves many risks that must be controlled. To address these risks, the physical and human environment should be specifically designed to compensate for the physical and/or cognitive impairments and the loss of autonomy [31]. Moreover, it is crucial for the environment to be well adapted for its specific resident' characteristics (biological or psychological needs, values, goals, abilities, personality, etc.). In psychology, that key concept is called *Person-Environment Fit*. The basic tenets of that approach are: (a) The person and the environment together predict human behavior better than each of them does separately; (b) outcomes are most optimal when personal attributes (e.g., needs, values) and environmental attributes (e.g., supplies, values) are compatible, irrespective of whether these attributes are rated as low, medium, or high; and (c) the direction of misfit between the person and the environment does not matter [60].

1.1 The need of assistive technology

Around the globe, many researchers think that technology could play a fundamental role in finding solutions for these important social issues [9]. This is why a growing worldwide community of scientists [1,9,11,21] now works on the development of new technologies based on the emerging concept of ambient intelligence (AmI) [52]. The term ambient intelligence refers to an approach that consists of placing devices in a place (such as sensors and effectors) and associating artificial intelligence (AI) with them in order to have a system that is capable of making decisions automatically to assist the occupants in is immediate environment. Smart homes [15], which are an application of AmI to the home domain, constitute an avenue of solution for maintaining people with disabilities at home [31]. The principle of smart homes is to retrieve data on the state of the environment from the various sensors deployed in everyday objects in order to infer, using AI techniques, the ongoing activities. Thereafter, the smart home system determines if the situation required and intervention (e.g., giving hints, reminders, etc.) and proposes, in real time, an assistive solution (cueing, hints, or reminders) using the actuators available in the room.

1.2 Challenges in developing assistive technology

A key challenge of any AI assistive system in a smart home is how to efficiently represent, in terms of knowledge engineering, the activity models [46]. Since the AI system is constantly performing inferences on these models based on the observation of the sensors' activation, the way activities are modeled, manipulated, and recognized is one of the core elements. The impact of a poor representation can be, for instance, an inaccurate or too slow recognition of the activities, preventing the system from making good decisions. A rigid representation can also make difficult to simulate the performance of activities based on the model. In the past, a lot of human activities' models has often been handcrafted by knowledge engineers, using logical representations [13,37], hidden Markov model [29,33], or Bayesian Networks [19,50]. More recently, machine learning techniques, such as clustering [63], have been widely used in order to avoid handcrafting the models and learn them automatically. The problem is that most of these approaches are generic, and the models are not easily adaptable. It is why we need to develop better models of residents behaviors so that the technology can be more in line with the person-fit theory [60] and better serve the needs of its' users.

1.3 Toward a solution to model human activities

In this paper, we address, as a whole, three challenges related to modeling human activities considering the need to adapt the model to user. To do that, our contribution consists in spreading new light on three questions: (i) How can we model the activity of daily living of user in a simple and flexible way; (ii) how can we automatically learn/construct these models based on a low-level actions recognition layer; (iii) how can we use these learned models to simulate the specific behavior of a targeted user. In our previous work, we briefly introduced the concept of representing activity of daily living using behavior trees [12], which are widely used in the video game industry [42]. This approach can be used to model a resident's behavior when performing an activity of daily living (ADL). In this paper, we push the concept a lot further using a customizable virtual smart home environment [27], where you can put virtual sensors and simulate the behavior of a resident in a smart environment. We then propose a global solution to the three questions asked using behavior trees. We introduce a series a new formal operator allowing to automatically generate (learns) the behavior trees, which are in fact activity models, from low-level recognized actions from sensors' activation. Finally, we introduce a new way of simulating user's behavior using these two formal tools in combination with our virtual smart home simulator. All these tools make together a complete contribution in the form of new software solution freely available online in open source and called the LIARA Smart Homes BT Global. Kit.[1]

This paper is organized as follows. Section 2 presents the related works. Section 3 presents our behaviour tree model for modelling activities. Section 4 presents our approach for automatic behaviour tree generation (learning behaviour trees from sensors logs). Section 5 presents our validation method and our experiments. Finally, Sects. 6 and 7 present our discussion on our proposal as well as the conclusion of the paper.

[1] https://github.com/Iannyck/shima.

2 Related work

The literature on how to model activity [18,20,55,65], how to learn these models [4,23,32,44,61,62], and how to test and simulate ADLs [2,3,14,27,34,35,41,56,57], is quite vast. As we said, scientists have proposed many different ways of modeling an activity and activities library. The way they choose to represent activities, as previous works clearly shown, directly affects the capacity of an intelligent agent to infer information from the library, to learn, to simulate human activities, to reason on how to give assistance in the performance of an activity, etc. Therefore, how we model the activities library is one of the key components of all smart home systems. Depending on how we choose to represent activities [18,20,55,65], we then have different ways of learning activities models [4,23,32,44,61,62], and to simulate them [27]. In fact, it is like a pipeline.

2.1 Modeling the activity of daily living

The first challenge that we cited is how to model a human activity in terms of a usable formal or data structure? The literature on how to models a human activity can be categorized in three families of approaches: (i) sensors-based models, (ii) logical representations, (iii) stochastic models. The first family of activities modeling approaches concerns formal representations directly based on sensors [20]. From a pragmatic point of view, it is easy, simple, and efficient to model the activities based directly on the observed inputs, which are sensors streams of data. In this kind of approach [39], an activity model is represented directly by a sequence of sensors events (e.g., activation of sensors). Indeed, that representation may slightly vary from different systems, depending on needs. For instance, some teams introduce the possibility to define a partial order between sensors, instead of a rigid sequence [39]. However, the fundamental modeling principles remain the same. The main limitation of that kind of approach is that there is no real abstraction in the model of activities. If we change environment and put different sensors, the activities' library becomes obsolete. It is also very difficult to reason at the high-level (understanding intentions, errors, etc.) with such a low-level representation. Finally, these models of activities can only be used in a simulation process in the same environment with the exact same set of sensors. The second family of activities modeling approaches is based on a logical formalism [13,17,37,64]. A logical approach to model activities consists of representing basic actions, activities, and relations between them with logical axioms. With that kind of model, the activity recognition process, for instance, can be structured by a set of inference rules defined with the same formalism. Different logical theory can be used. For instance, first-order logic has been used by Kautz [37], Camilleri [17] and several others [55] to represent a collection of activities (action and activity types) in many activity recognition systems. Some teams also proposed to represent the activities library using situation theory [6], which is a particular case of possible world's theory Wobcke [64], or with description logic (DL) [5]. The main problem with logical representations is that they are very complex, despite their rich expressivity. Reasoning with that kind of representation often constitutes a computational challenge [55]. The last family of activities modeling approaches includes representation based on probabilistic graphical model [19,29,33,45,50,54]. Most researchers in the field of ambient assisted living [9] exploit Bayesian networks [19,50] and hidden Markov models (HMMs) [29,33], conditional random fields (CRF) [45], or fuzzy finite state machines (FuFSM) [28]. Stochastic models are very rigid and difficult to modify. For instance, once modeled, it is complicated to add an activity, because of the important supplementary amount of data required to relearn the probability distribution of the model to avoid the imbalance class problem. This constitutes a clear limitation in a context where one wishes to learn new behavior. In addition, the learned probabilities are very user specific.

2.2 Learning automatically the activities models

In parallel with the challenge of modeling activities, the issue of automatically learning and building the activities models has also been investigated in many different ways [23]. As we pointed out, the approach we choose to represent ADLs has a big impact on how we can learn automatically or semiautomatically an activity model. With a sensors-based low-level modeling of activities, the log of data from the various sensors is often processed using support vector machine (SVM) [32] to learn activities and classify them using, for instance, temporal frames [24]. Other scientists make use of deep neural network (DNN), which is a particular form of traditional artificial neural networks, more capable of learning from large data [61]. Many teams also exploited the well-known C4.5 algorithm and tried to build decision trees representing activities with sensors activation [53]. At the end, learning sensors-based activity models using machine-learning techniques are relatively simple. The problem with that kind of learning solution derives from the modeling approach itself, the limitation of it remains the same, the learned models contain no real abstraction of the behavior of the user, and if we change the sensors, the environment, or if the behavior of the user is variable over the time, the models will become obsolete. On the other hand, if one uses stochastic models to represent activities, learning and building these models automatically is not as simple. In the field of human activity recognition (HAR), the most commonly used representation models is HMM. However, most of teams using HMM for activity recognition simply handcraft their models [43]. They usually only focus on a small set of targeted activities to recognize, which are analyzed and modeled directly by an expert. Only the probability of transition between states and observations is automatically learned thereafter. However, HMMs activity models have been built automatically using feature selection techniques, which heuristics and a genetic algorithm (GA) [22]. The well-known team of Professor Diane J. Cook also proposed a constraint-based (CB) Bayesian structure learning algorithms, to automatically build activity's models in the form of a Bayesian network [44]. Nevertheless, while HMM and Bayesian networks are widely used for modeling activities in the context of smart homes, they are usually handcrafted and rarely automatically learned [43]. Learning these kinds of models from logs of data is complex, and the quality of resulted models is debatable. In some recent works, researchers have tried to learn human activity' models using fuzzy finite state machine (FFSM) in an intelligent environment [28]. The proposed approach, called neuro-fuzzy finite state machine (N-FFSM), is able to learn the parameters of a rule-based fuzzy system, which processes the numerical input/output data gathered from the sensors and/or human experts' knowledge. However, the representation of activities with finite state machines results in a relatively rigid model that is based on environmental states rather than the actions, intentions and subtleties of the observed user's behavior. Finally, regarding learning techniques for automatically constructing logical representation, several attempts have been made using, for instance matrix factorization [62] or C4.5 decision trees [4]. The problem of theses approaches is how to manage the observations containing a great deal of noise or errors. Logical representation models are best effective in a deterministic environment. In the context of AAL, the generalization of learned logical rules might easily lead to inferring inconsistent behaviors [13].

2.3 Simulating users' behavior in a smart environment

In the literature, we can find several works that focus on how to generate datasets or simulate behaviors in smart homes [2,3,14,34,41,56,57]. For instance, our team proposed in 2012 an open-source smart home simulator [14]. It provided an interface to design the smart home and to write scripts. In the scripts, the user has to define the sequence of steps involved in the performance of an activity. The user can set parameters such as the completion time of a step and the objects that are involved. The simulator was made in Java and runs into a 3D environment designed with SketchUp. This first simulation tool from our team was rigid and not easily customizable. PerSim 3D is another simulation tools proposed in [34]. It uses a model-based virtual approach for the modeling sensors' behavior. It also provides users with several categories of sensors. It also allows the user to design the architecture of the smart home. However, the scripting power is also limited. UbikSim is a simulator of the intelligent environment proposed in [10,56]. It uses multi-agent based simulation (MABS) to perform the occupants' behaviors. More precisely, it uses the Java library multi-agent simulator of neighborhoods (MASON) [40] for the simulation of occupants. Consequently, UbikSim supports simulations involving several occupants. It uses Sweet Home 3D [51] (a computer-aided design software for designing interiors) to support the design of the environments. UbiKSim provides users with two binary sensors, door sensors, and pressure sensors. It works in a 3D world and provides a real-time rendering. However, UbikSim does not work with representations such as BT. The intelligent environment simulation (IE Sim) [41,57] is used to generate simulated datasets that capture normal and abnormal activities of daily living (ADLs) of inhabitants. This tool provides users with a 2D graphical top view of the floor plan to design a smart home. It proposes different types of sensors such as temperature sensors and pressure sensors. Simulation is also performed in a 2D world. Ariani et al. [3] proposed a smart home simulation tool that uses ambient sensors. The solution provides users with an editor that allows the design a floor plan for a smart home by drawing shapes on a 2D canvas. Once this step is over, we can add ambient sensors to the virtual home. The solution can simulate binary motion detectors and binary pressure sensors. In [47], Park et al. proposed a 3D simulator to generate inhabitants' datasets for classification problems. Their simulator is built with Unity3D [58], which is a professional multiplatform game engine used in the videogame industry. They have analyzed and collected data to generate a user activity-reasoning model in a virtual living space. We can also notice OpenSHS, which is an hybrid open-source cross-platform 3D simulator, thanks to the use of Blender [2] and Python. During simulations, users control an avatar. However, the solution proposes a fast-forward mechanism to allow users to mimic, but not perform a whole activity in real time. This mechanism uses a replication algorithm to extend and expand the dataset. It simply copies and repeats the existing state of all sensors and devices during the specified period. These simulation solutions are mostly based mainly on raw sensor data and do not support a representation of high-level activities such as the proposed approach with behavior trees. Moreover, most of them are not very customizable. In 2017, we introduced a new 3D open-source smart home simulator [27]. This simulator uses the Unity game engine and allows you to build in 3D your environment (house, apartment, office, etc.) and add a set of sensors. It can simulate RFID antennas, power consumption, ultrasonic sensors as well as binary sensors such as contact sensors, pressure plates, or motion detectors. To simulate human activity, the simulator allows direct control of an avatar using the keyboard and mouse or it proposes to model interaction scenarios from behaviour trees. The result of the simulation is a database with raw data such as signal strengths and distances. Ho et al. propose the SESim 3D simulator which also uses Unity engine and aims to simulate realistic datasets to help research in the field of recognition of activities of daily life [35]. As for our simulator [27], it defines different data generation models for each type of sensor. The sensors are activated by the actions of the avatars which are defined by scripts.

3 Behavior tree model for modeling ADL

Our approach to human activity modeling is based on principles defined in Humphreys and Fordes's [36] hierarchical organization model of human activities. In this model, an activity (or routine) can be split into a sequence of actions that can themselves be broken down again into a sequence of atomic actions (here the atomic term means that the actions cannot be broken down into a relevant smaller unit). We chose to model human activities in a hierarchical way and adapted our behavior trees. Behavior trees are a formalism that is regularly used in the field of planning, and in video game development to define and implement the behavior of non-player entities [42]. "ConcurTaskTrees" can also be seen as an adaptation of behavior trees to the field of user interface design [48,49]. Indeed, we find the concept of control nodes and leaves represent tasks instead of "atomic" behaviors. In our context, we use behavior trees to model the behavior of the occupant in an intelligent environment and to reproduce this behavior in a simulator [8,12,27].

A behavior tree is a tree oriented $\mathcal{T}(\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges. The fact that the tree is oriented means that the edges have a node considered as the parent (the node at the start of the edge) and a child node (the node at the finish of the edge). The nodes can be of one of two types:

- Parent nodes are flow (or composite) control nodes; they control their children's execution order;
- Leaves are execution nodes.

The execution and updating of a behavior tree is based on discrete updates called "Tick." In each "tick," the tree is traversed in depth from the root. During this update, each traversed node calculates its state, and if the node is a composite, it defines which of its threads will be traversed. For many composites, the path is done (considering that the nodes are aligned horizontally) from left (the first node) to right (the last). However, some compositions we will see define another order of path. If the node is a leaf, the tick triggers the execution of an action or condition. In all cases, the node state calculation returns one of three values: *Success*, *Failure*, *Running*. The calculation of a condition node returns only one of the following two values : *Success* or *Failure*. *Success* and *Failure* states are end states. This means that when one of these two states is reached, the node can no longer change state. The *Running* state is an intermediate state where the conditions of success or failure are not reached. The state *Failure* means that from the current conditions it will be impossible to achieve the conditions of success. The state *Success* means that the action has been completed or that the condition to be verified is *true*.

We just saw the global operation of a BT. New, let us shift our focus on the different types of composite nodes. Table 1 summarizes the conditions that must be met for a node to be found in each state. The list of composite is as follows:

- Selector: The principle of this composite is to "try" children's behaviors sequentially until the behavior is successful or all behaviors fail. Thus, it ticks sequentially his children until it has a child who finishes in the state of *Success*. If all its children finish in the state of *Failure*, it finishes in *Failure*.

Table 1 Composite node types of a BT

Node type	Symbol	Succeeds if	Fails if	Runs if
Root	$\emptyset$	Tree S	Tree F	Tree R
Selector	?	1 Ch S	N Ch F	1 Ch R
Random sequence	$\sim$?	1 Ch S	N Ch F	1 Ch R
Sequence	$\rightarrow$	N Ch S	1 Ch F	1 Ch R
Random sequence	$\rightsquigarrow$	N Ch S	1 Ch F	1 Ch R
Parallel	$\Rightarrow$	$\geq$ A Ch S	$\geq$ B Ch F	Otherwise
Decorator	δ $\boxed{\lambda}$	Varies	Varies	Varies

The following notation is adopted: $Ch \equiv Children$, $S \equiv Success$, $F \equiv Failure$, $R \equiv Running$, $N \equiv$ *number of children*, A, $B \in \mathbb{N}$ and λ are node parameters

- Random Selector: The principle is the same as the selector except that here the nodes are tried randomly, and not sequentially, it randomly ticks one of its children. If the selected node ends in *Success*, this composite ends in *Success*. If all its children end in *Failure*, it ends in *Failure*.
- Sequence: The principle is to do all behaviors successfully. Thus, this composite tick all its children sequentially, if all its children end in *Success* it ends in *Success*. As soon as one of its children finishes in *Failure*, it finishes in *Failure*.
- Random Sequence: The principle is the same as the composite sequence except that the activation is done randomly. So, this composite ticks all its children randomly, if all its children end in *Success* it ends in *Success*. As soon as one of its children finishes in *Failure*, it finishes in *Failure*.
- Continuation: The principle is to perform the behaviors sequentially until reaching a predetermined number A ($A \in \mathbb{N}$) of behaviors that end in *Success* or a predetermined number B ($B \in \mathbb{N}$) that end in *Failure*. Thus, it ticks sequentially his children until reaching the number A of *Success* to pass to the state of *Failure*, or reaching the number B of *Failure*s to pass to the state of *Failure*.
- Decorator: The principle of a decorator is to pass into a state of end by transforming the state of its only child. The decorators are as follows:
  - Inverter: Its end state is the opposite state of its child (*Success* is transformed into *Failure* and inversely)
  - Repeat: It repeats the tick of his son until he reaches an A number of *Success*.
  - Error: It has a probability not to tick its child and go to the state *Success*.

Behavior trees allow us to model ADLs by constructing complex behaviors from a set of atomic elements that are actions and conditions. In our context, an action will correspond to an action that the occupant can perform on his environment, for example take an object, activate a device, open a cupboard, pour something, etc. A condition is a check on the condition of an entity or object that the occupant can perform, such as "check if an appliance is turned on." Thus, our behavior trees are built from a set S containing the subsets of actions $A = \{a_1, a_3, a_2, \ldots, a_i\}$ and conditions $C = \{c_1, c_3, c_2, \ldots, c_j\}$. It should be noted that these actions and conditions are configurable, so, for example, the "take" action receives as parameters: the target object and the time required to perform the action. These parameters

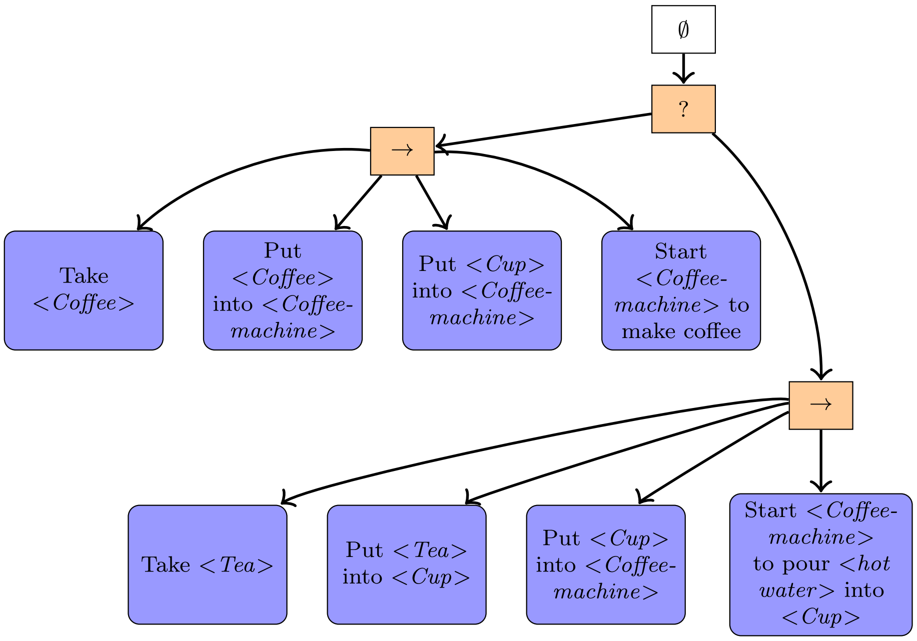

∅
?
→
Take <Coffee>
Put <Coffee> into <Coffee-machine>
Put <Cup> into <Coffee-machine>
Start <Coffee-machine> to make coffee
→
Take <Tea>
Put <Tea> into <Cup>
Put <Cup> into <Coffee-machine>
Start <Coffee-machine> to pour <hot water> into <Cup>

Fig. 1 BT that manages the scenario of preparing a coffee or otherwise a tea

make it possible to reduce the size of the A and C sets. Indeed, it is not necessary to define an action "take" for each object.

Now we will present the modeling of a scenario to illustrate our approach. We will model the scenario where the occupant has to make a coffee. To create this scenario, we need the following action set (here, we note the target objects of the actions between the symbols "< >"): Take <*object*>; put <*object1*> on <*object2*>; put <*object1*> into <*object2*>; turn on <*device*>. From our set, the construction of this scenario is done using the composite "sequence" and assigning it as threads (in the respective order of activation): (1) Take <*coffee*>; (2) put <*coffee*> into <*coffee-machine*>; (3) take <*cup*>; (4) put <*cup*> into <*coffee-machine*>; (5) turn on <*coffee-machine*>. This model gives the following behavior: (1) the occupant takes coffee; (2) he puts the coffee in the coffee machine; (3) he takes his cup; (4) he places his cup in the coffee machine; (5) he starts the machine. However, if any of these actions cannot be done, the composite will fail in the state indicating that the activity has not been completed.

The flexibility of the behavior trees allows us to quickly and simply model more complex behaviors. Thus, the behavior which makes it possible to make coffee can be made more complex by allowing for example to make tea if one of the actions allowing to make coffee cannot be accomplished. For this, we will use the composite "selector." Thus, we obtain the behavior tree presented in Fig. 1. With this model, the occupant will try to make a coffee then in case of failure (for example no more coffee), he will try to make a tea.

Our approach also allows us to introduce errors in the implementation of ADLs. For that, we introduced the decorator "*Error*" having for parameter λ a number between 1 and 100. This value indicates the probability that the node switches to the state *Success* without activating its child. This operator allows us to create errors of omission that can be committed by people

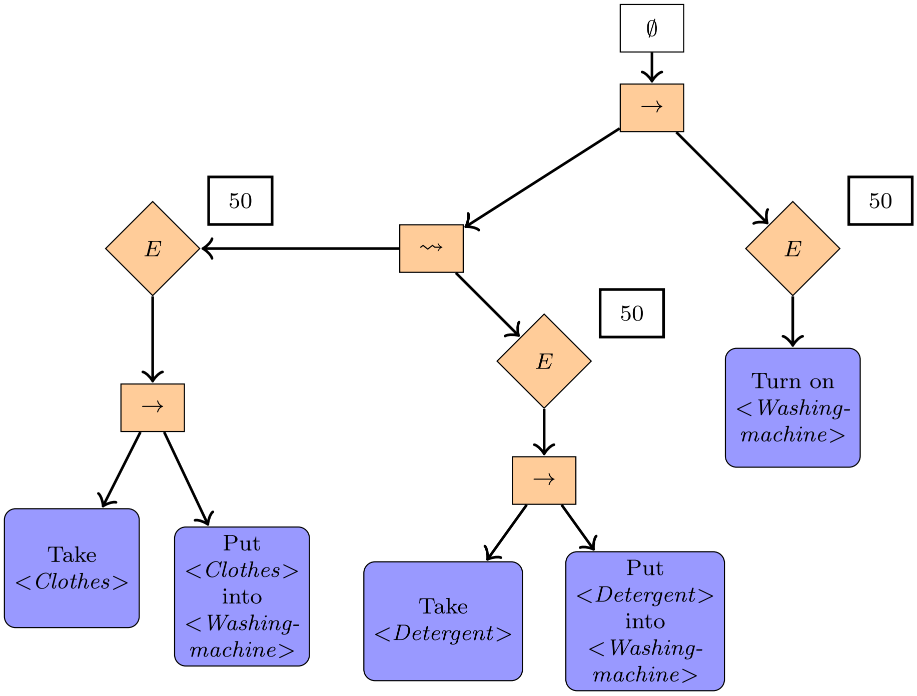

∅
→
50
E
⇝
50
E
50
E
→
Turn on <Washing-machine>
→
Take <Clothes>
Put <Clothes> into <Washing-machine>
Take <Detergent>
Put <Detergent> into <Washing-machine>

Fig. 2 BT that manages the scenario of doing the laundry and having a certain probability (here 50%) to omit certain steps throughout the completion of this ADL

with cognitive impairments. Thus, we can take a behavior tree to model the behavior of a healthy person and modify it to introduce possibilities of error thanks to this operator.

We will illustrate this decorator with the ADL: "doing the laundry." To model this scenario, we can use the tree shown in Fig. 2. With this behavior tree, the occupant does the laundry in the following order: (1) Put the clothes and detergent in the machine; (2) start the machine. The sub-activity "Put clothes and detergent in the machine" is divided into two sequences that can be performed in any order (clothes or detergent first), that is why we use the composite "random selector." As we added three "*Error*" composites in this tree, during the simulation the occupant can omit to: (a) Take the laundry and put it in the machine; (b) take the detergent and put it in the machine; (c) start the machine. We have defined a 50% chance of omission for each activity. Thus, he can possibly make no mistakes or omit everything.

4 Learning BT from datasets

The behavior trees we introduced in the previous section allow us to achieve our first objective, which is to have a simple and flexible approach to modeling ADLs. Behavior trees are easily interpretable for a human and a computer. In addition, it is easy to modify them to create new scenarios. In this section, we are interested in achieving our second objective, which is to have a method for automatically learning these trees from a dataset. Our main contribution in this paper is to propose a method that allows to automatically build a behavior tree that

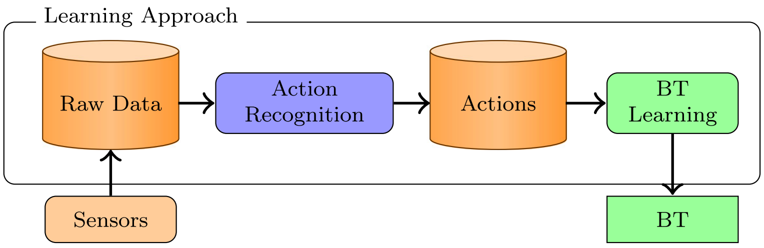


Fig. 3 The main components of our approach

represents the different ways a human user performs activities of daily life in order to use this tree to simulate new scenarios.

To achieve this learning, our approach meets a constraint resulting from the principle of how intelligent environments work. The sensors in these environments produce raw data that is used by an activity recognition system to recognize patterns. However, the behavior trees we use do not work on raw sensor data. They use a set of atomic actions and conditions. Thus, the first step in our approach is to transform sensor data into action logs to learn. Once this log of atomic actions is obtained, we can use a learning method derived from the suffix tree generation algorithm to generate the behavior trees [30]. The steps of this approach are summarized in Fig. 3.

4.1 Generation of atomic action logs

The transformation of raw sensor data is the first step in our learning process. These data come from different types of sensors and create heterogeneous datasets (numerical, Boolean, string, etc.). Table 2 gives an example of a dataset, and our objective is to transform datasets of this type into a log that looks like the one given in Table 3.

Table 3 contains the atomic action sequence that was done to perform an ADL. These actions are part of the set of atomic actions we use to build trees. To generate this log, we need to recognize our set of atomic actions. To do this, we will select for each atomic action a recognition technique to apply to the dataset. The choice of technology depends on the types of sensors available. For example, we can analyze the evolution of power consumption to recognize the activation and deactivation of a particular electrical device [7]. By using RFID antennas and trilateration [16,25,26], we can estimate the position of objects and track their movements in order to recognize their use by an occupant, etc.

The use of this action log gives us a level of abstraction from working directly with the raw data. This level of abstraction is very interesting, because it allows us to be independent of the architecture of the environment. Indeed, the models resulting from the learning of these action logs are more flexible than the direct exploitation of raw data, because the latter are closely linked to the sensors. Therefore, if the sensors change some data may not be available.

4.2 Learning BT from atomic action logs

At the end of the action log generation step, we have the action sequence done to perform an ADL. This sequence represents one habit of the user in relation to the realization of the ADL. However, a person may have several ways of proceeding to carry out the same ADL. For example, to prepare a coffee, the person will not do the actions in the same order and

Table 2 Sample of the raw sensor data

Timestamp	SensorId	Type	Value
05:56:45:625	Living room	PIRMotion	True
05:56:48:341	Living room	PIRMotion	True
05:56:54:818	Living room	PIRMotion	True
	...		
05:58:21:866	Bathroom sink	BinaryFlowMeter	True
05:58:36:848	Bathroom sink	BinaryFlowMeter	False
	...		
06:00:09:836	Kitchen shelf top left	ContactSensor	False
06:00:15:855	Living room	PIRMotion	True
06:00:17:18	Living room	PIRMotion	True
06:00:31:252	Kitchen shelf top left	ContactSensor	True
06:00:47:218	Kitchen shelf top middle	ContactSensor	False
06:00:48:919	Living room	PIRMotion	True
	...		
06:09:34:810	RFID0	RFID Sensor	− 24.471
06:09:35:003	RFID1	RFID Sensor	− 41.184
06:09:35:262	RFID2	RFID Sensor	− 52.847
	...		

Table 3 Expected action log

Timestamp	Action
06:08:09:836	Take cup
06:08:27:836	Put cup in coffee machine
06:08:41:252	Take coffee
06:09:03:295	Put coffee into coffee machine
06:09:18:299	Start coffee machine
06:10:08:471	Take cup

sometimes he will add sugar to his coffee. However, because we want to learn the general behavior of a person for the realization of an ADL, we need to have all his habits when performing the ADL. To do this, it is necessary to generate several datasets to have at least one dataset and a sequence after generating the log by habit. Once all these sequence logs have been obtained, we can combine them into a single log that represents all the habits of a person for the realization of an ADL. Table 4 gives an overview of this log.

Our objective is to generate from this log a behavior tree that represents all the different ways for a person to perform an ADL. We can notice that the structure we want to obtain has common characteristics with the suffix trees; it must reflect the internal characteristics of sequences (actions in our case and letters for an application of the suffix trees). Thus, our learning approach is based on algorithms for creating suffix trees.

The principle is as follows: We proceed through the action sequences action by action, and we treat the following two cases:

1. The action is the same for all sequences, we add this action in the sequence, and then we get the next one.
2. The action is not the same for all sequences; we create as many branches as there is a different action. To create these branches, we create as many "sequence" nodes as there are branches to create. The corresponding action is added to each created node. A selector node is created to which all the created "sequence" nodes are added as sons. Finally, the selector is added as son of the current node; the data are divided into continue processing from the action of the new branch.

In this algorithm, lines 1–4 initialize the tree by creating the root node. Lines 7–16 deal with the first case; lines 18–25 deal with the second case.

ALGORITHM 1: LEARNING: Tree learning and generation algorithm

```
Input: $data$ the behavior tree, $current\_node$ the current node to develop
if $bt = \emptyset$;
then
    add $root\_node$ into $bt$;
    $current\_node \Leftarrow root\_node$;
end
if $data \neq \emptyset$;
then
    $actionList \Leftarrow$ Get the list of the next action in each sub sequence;
    if Next actions are the same for all sub sequences;
    then
        if $current\_node$ is a sequence node;
        then
            Add $action$ as son of $current\_node$;
        else
            Create a composite $sequence\_node$ and add the $action$ as son;
            $sequence\_node$.SETPARENT($current\_node$);
            $current\_node \Leftarrow sequence\_node$;
        end
        Remove $actionList$ from $data$;
        LEARNING($data$, $current\_node$);
    else
        Create a composite $random\_selector$;
        $random\_selector$.SETPARENT($current\_node$);
        for each $action$ in $actionList$ do
            Create a composite $sequence$ and add the $action$ as son;
            $sequence\_node$.SETPARENT($random\_selector$);
            $current\_node \Leftarrow sequence\_node$;
            $subdata \Leftarrow$ Get action sequences that start with $action$ from $data$ ;
            LEARNING($subdata$, $current\_node$);
        end
    end
end
```

4.3 Algorithm: a step-by-step example

Let's take a simple example to illustrate how the algorithm works. We take two different people who have to do a coffee making activity. With our method, these people perform

Table 4 Sequences log

#	Sequence
1	A, B, C, D, E, F
2	A, C, D, G, H
3	A, C, E, H, D
4	E, G, F, J
5	E, G, F, J
6	A, C, D, G, H

Each letter presents an atomic action that has been recognized by an action recognition system

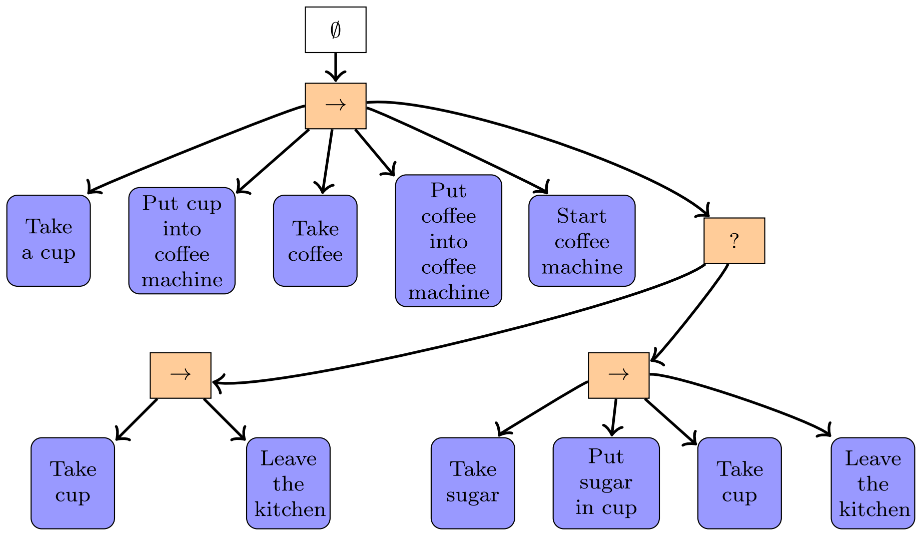


Fig. 4 BT for the person 1

these ADLs several times in order to generate a dataset containing all the ways people use them. In our example, these datasets give the following sequence sets for person 1:

– Sequence 1: Take a cup; place the cup in the machine; take coffee; place the coffee in the machine; start the machine; take the cup; leave the kitchen with the cup.
– Sequence 2: Take a cup; place the cup in the machine; take coffee; place the coffee in the machine; start the machine; take the cup; take sugar; put sugar in the cup; take the cup; leave the kitchen.

If we apply our algorithm to this set of sequences, we obtain the behavior tree in Fig. 4. We notice that the first 5 actions are identical for the two ways of doing the ADLs. This fact implies that the algorithm executes lines 6–16 and produces the first sequence. The 6th action is different for the two ways of doing things. This is where the algorithm performs lines 18–25 which will create the new branches (in our case, 2 branches). The data will also be divided into n parts (n is equal to the number of branches); each part will contain the subsequences related to each new branch.

For the second example, let us consider that the second person produces the following sequences of actions:

- Sequence 1: Take cup; take sugar; put sugar into the cup; take milk; pour milk into cup; put cup in coffee machine; take coffee; put coffee in the machine; start coffee machine; take cup; leave the kitchen.
- Sequence 2: Take sugar; take milk; take cup; put cup in coffee machine; take coffee; put coffee in the machine; start coffee machine; put sugar into the cup; take cup; leave the kitchen.
- Sequence 3: Take cup; take sugar; put sugar into the cup; take milk; put cup in coffee machine; take coffee; put the coffee in the machine; start coffee machine; take cup; pour milk into cup; leave the kitchen.

If we apply our algorithm to this set of sequences, we obtain the behavior tree in Fig. 5. In this example, the first actions are not the same for the three sequences. Only sequences 1 and 3 have the first four identical actions. Thus, with this sequence set, the algorithm creates 2 branches from the beginning and separates the data into two subsets. One subset from sequences 1 and 3, the other from sequence 2. The subset derived from sequence 2 gives the right branch of this behavior tree. In the subset derived from sequences 1 and 3, the fifth action is different in these two sequences, so our algorithm creates two new branches at this point and generates two new subsets, one for each sequence. At this point, the algorithm processes a subsequence twice to complete its current branch.

5 Validation

In the previous sections, we presented a model based on behavior trees to represent ADLs and an approach to learn these behavior trees. These methods meet our first two challenges: (1) Construct a model to represent the sequences of actions necessary for the realization of an ADL and (2) design an approach to automatically learn this model. The intelligent environment simulator helps us to meet the third challenge, which is to have a tool that allows us to generate datasets in order to test new AI algorithms [27].

In this section, we present the experiment carried out to validate our approach. The formal contribution of this paper consists mainly in introducing new operators and tools extending the model of behavior trees in order to use it to automatically learn human activity models. The scope and the goal of the proposed experiment were only to demonstrate that. The experimental question was: Is the proposed new leaning technique and the new formal tools are able to learn behavior tree' models that are equivalent to the one observed in the smart home from sensors' activation?

To answer this question, we used our simulation tools to automatically generate a behaviour tree that allows to reproduce the behaviour of a cognitively impaired person in the performance of an activity of daily life, and this from the log of the sequences of atomic actions performed by him/her. These scenarios are inspired from previous experiments that our team performed with end-users. Thus, the objective of the test was to verify if the tree built from a log file allows to reproduce all the cases that are present. We focus specifically on the verification of the tree generation and not on the recognition of the atomic actions. We assume that the global system has tools capable of performing atomic action recognition.

Thus, to perform the validation we used a simulator to generate a log file containing the atomic actions. The particularity of this simulator is that it uses behavior trees to simulate the performance of everyday activities. We will therefore generate a new behavior tree from the log file and check if the new tree is equivalent to the one used to generate the log.

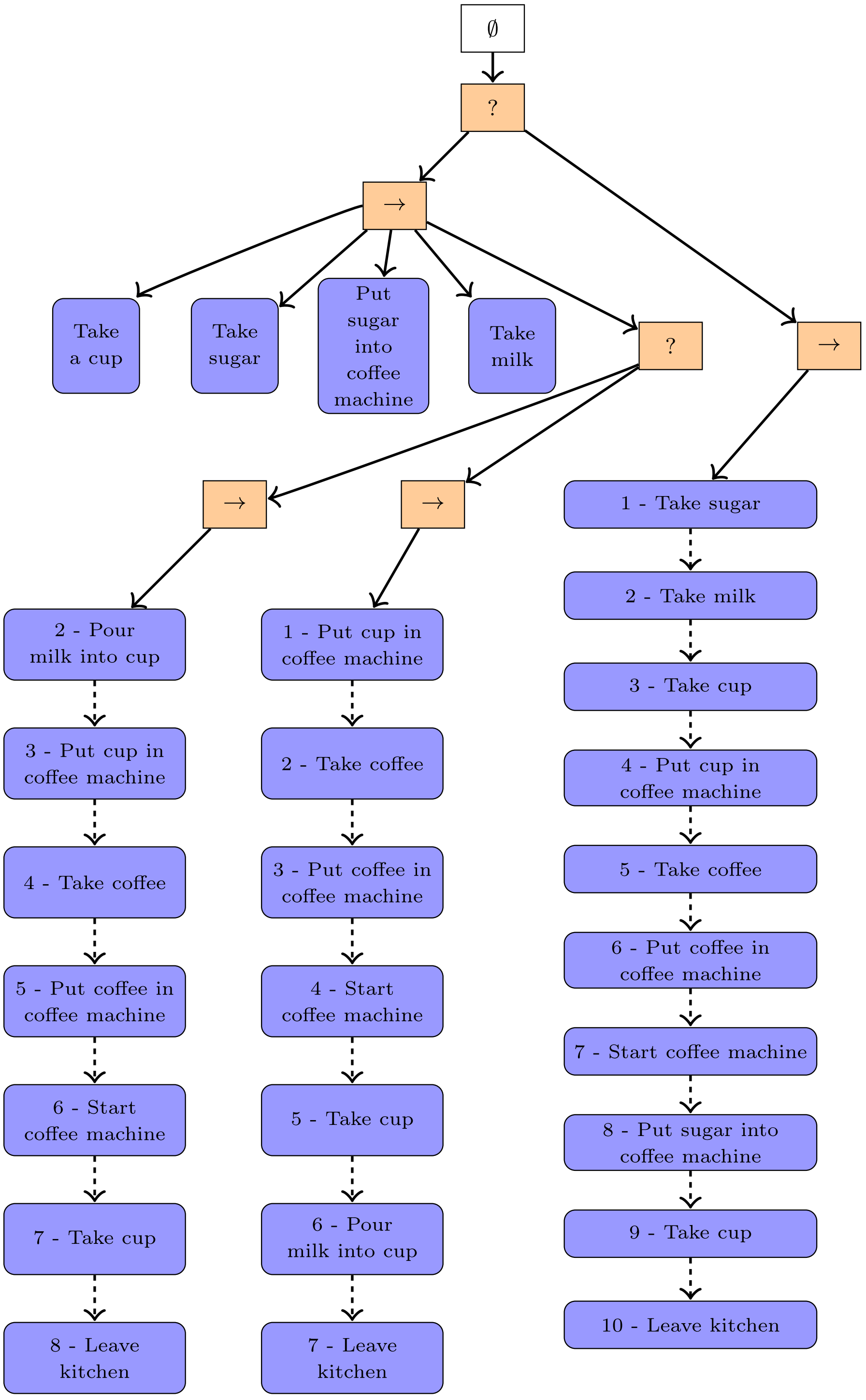


Fig. 5 BT for the person 2. The arrows in tilted dots mean that the nodes are brothers

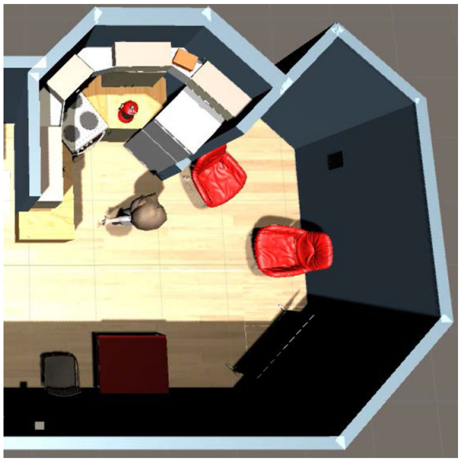


Fig. 6 Screenshot of a virtual smart home designed with our simulator

5.1 Generation of the atomic action dataset

In order to generate our atomic action dataset, we use our simulator described in the article [27]. This simulator allows to generate datasets thanks to its behavior tree interpreter. We define the tree in Fig. 7 representing the ADL of preparing a coffee and a dish and consisting of the following atomic actions: Take a cup (t_{cup}); take a coffee capsule (t_{coffee}); place the cup in the machine (p_{cup}); place the coffee in the machine (p_{coffee}); start the coffee machine (s_{coffee}); leave the kitchen with the cup ($l_{kitcheen}$); take a frypan (t_{frypan}); place the frypan on the cooker (p_{frypan}); start the cooker (s_{cooker}); turn the cooker off (to_{cooker}); take a plate (t_{plate}); pour meal (from the frying pan) into the plate (po_{plate}). We define the following non-regular actions: Take milk (t_{milk}), pour milk into the cup (po_{milk}), take sugar (t_{sugar}), pour sugar in the cup (po_{sugar}), take salt (t_{salt}), pour salt into the dish (po_{salt}). These non-regular actions are, by definition, not carried out in some of ADL's achievements. To define these non-regular actions, we specify the "Ignore" operator who takes as parameter the percentage of chance not to realize his son.

In our tree, the "random sequence" operators allow to create variations in the completion of the ADL. We simulate this ADL 20 times and record the atomic action sequences. For each action, we record the time when the action begins and the time when the action is completed. The simulation environment is shown in Fig. 6. We conduct two tests, the first on a log of 25 sequences generated using the trees shown in Figs. 7, 8, 9 and 10. The second test is performed on a log of 10 sequences generated using the same tree, but setting the "Ignore" operators to 100 (actions will never be performed). Table 5 presents the log generated for the first test. Table 6 shows the log generated for the second test.

5.2 Behavior tree learning

By applying Algorithm 1 to the data in Table 5, we obtain the behavior trees shown in Figs. 11, 12 and 13. The tree is split and rendered on multiple figures to make it easier to read.

We can notice that the trees learned from the atomic action logs are bigger than the tree used as the source to generate the log. This specificity comes from the approach we use to learn the sequences of identical actions that can be performed in a different order. This is specifically

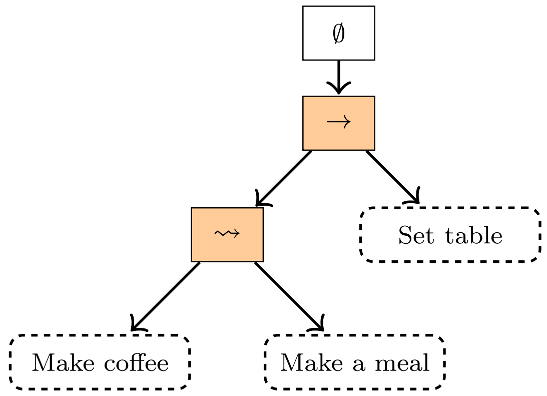


Fig. 7 BT for the experiment. The nodes between dashed lines are detailed in Figs. 8, 9 and 10

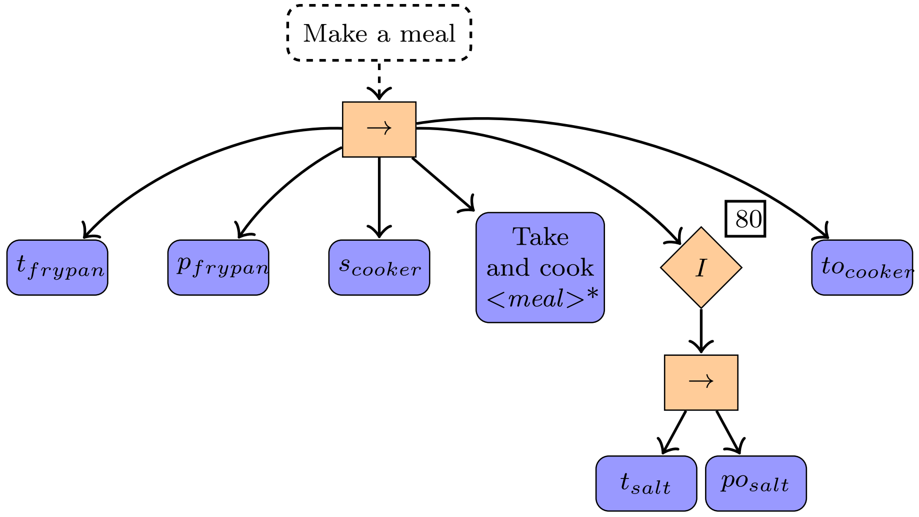


Fig. 8 Sub-tree "Make a meal" of the BT of the experiment of Fig. 7

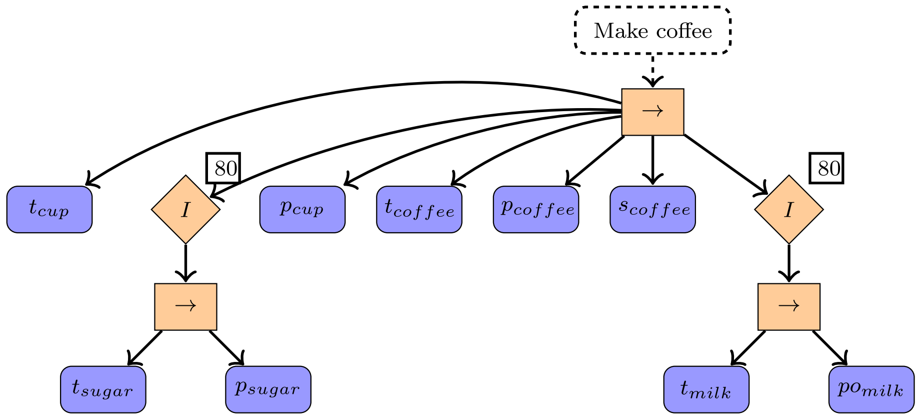


Fig. 9 Sub-tree "Make coffee" of the BT of the experiment of Fig. 7

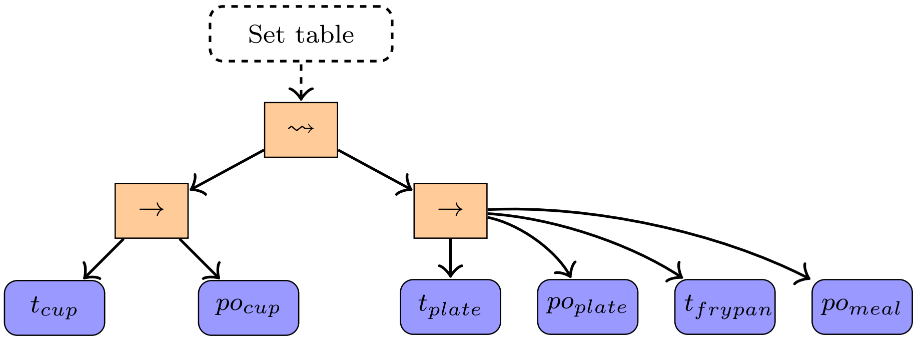


Fig. 10 Sub-tree “Set table” of the BT of the experiment of Fig. 7

possible with the “random sequence” operator. With our approach, we use a combination of the “selector” operator and a “sequence” operator for each sequence possibility.

The irregular actions that appear in some sequences are also the cause of a larger tree. Indeed, these actions that sometimes insert themselves into sequences will be interpreted by our algorithm as new subsequences. It will therefore create new branches using the combination of the “selector” operator and a “sequence” operator.

We have seen the differences between the three trees, now let us check if the trees produced are equivalent to the trees used to generate the logs (which we will call “original trees”) and using more different operators. To achieve this, let us compare the trees. The first original tree defines a sequence of 3 activities: (a) Make a coffee; (b) make a meal; (c) set the table. In addition, it uses the “random sequence” operator to allow activity “b” to be performed before activity “a.” This is reflected in the log by action sequences (see Tables 5, 6) starting either:

– by the first action belonging to the sequence of activity group “a” (sequences, 1, 5, 7, 8, 11, 12, 15, 20, 23, 24 and 25);
– by the first action belonging to the sequence of activity group “b” (sequence 2, 3, 4, 6, 9, 10, 13, 14, 16, 17, 18, 19, 21 and 22).

The first generated tree starts with a structure using a “selector” with two “sequence” operators as children. The actions in each sequence belong to one of the sequences in activity group “a” or “b.” This structure makes it possible to reproduce the behavior of doing one or the other activity first.

Let us focus on the activity “make a meal,” it consists of a sequence of actions including two irregular actions (“take the salt” and “pour the salt”). In the original tree, the non-regular nature of these actions is modeled using the “ignore” operator which allows the actions to be performed with a 20% probability. This irregularity potentially creates two different subsequences of actions. This is represented in our tree by the presence of the “selector” operator with two “sequence” operators as children after the action “start the stove.” This case appears twice in the generated tree. The first case in Fig. 12 represents the case where the individual started by making a coffee without sugar or milk before making the meal. The second case appears in Fig. 13 and represents the case where the individual started by making the meal. This structure is not present on all branches (for example, the case where the individual started by making coffee, did not put sugar, but put milk), because even if the structure of the original tree means that the individual can potentially add salt in this case,

Table 5 Sequences log for test 1

#	Sequence
1	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{fry};\ p_{fry};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{fry};\ po_{meal};\ t_{cup};\ po_{cup}$
2	$t_{fry};\ p_{fry};\ s_{cooker};\ t_{salt};\ po_{salt};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{plate};\ po_{plate};\ t_{fry};\ po_{meal};\ t_{cup};\ po_{cup}$
3	$t_{fry};\ p_{fry};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{plate};\ po_{plate};\ t_{fry};\ po_{meal};\ t_{cup};\ po_{cup}$
4	$t_{fry};\ p_{fry};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
5	$t_{cup};\ t_{sugar};\ po_{sugar};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
6	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
7	$t_{cup};\ t_{sugar};\ po_{sugar};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{fry};\ p_{fry};\ s_{cooker};\ to_{cooker};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{fry};\ po_{meal}$
8	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
9	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
10	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
11	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ t_{salt};\ po_{salt};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
12	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
13	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
14	$t_{frypan};\ p_{frypan};\ s_{cooker};\ t_{salt};\ po_{salt};\ to_{cooker};\ t_{cup};\ p_{cup};\ s_{coffee};\ t_{coffee};\ p_{coffee};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
15	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ t_{salt};\ po_{salt};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
16	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
17	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
18	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ t_{sugar};\ po_{sugar};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
19	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
20	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
21	$t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{cup};\ po_{cup};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal}$
22	$t_{fry};\ p_{fry};\ s_{cooker};\ t_{salt};\ po_{salt};\ to_{cooker};\ t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
23	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{milk};\ po_{milk};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
24	$t_{cup};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$
25	$t_{cup};\ t_{sugar};\ po_{sugar};\ p_{cup};\ t_{coffee};\ p_{coffee};\ s_{coffee};\ t_{frypan};\ p_{frypan};\ s_{cooker};\ to_{cooker};\ t_{plate};\ po_{plate};\ t_{frypan};\ po_{meal};\ t_{cup};\ po_{cup}$

$t_{fry} = t_{frypan},\ p_{fry} = p_{frypan}$

Table 6 Sequences log for test 2

#	Sequence
1	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
2	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
3	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; po_{cup}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}$
4	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
5	$t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{cup}; po_{cup}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}$
6	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
7	$t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
8	$t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; po_{cup}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}$
9	$t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}; t_{cup}; po_{cup}$
10	$t_{frypan}; p_{frypan}; s_{cooker}; to_{cooker}; t_{cup}; p_{cup}; t_{coffee}; p_{coffee}; s_{coffee}; t_{cup}; po_{cup}; t_{plate}; po_{plate}; t_{frypan}; po_{meal}$

$t_{fry} = t_{frypan}, p_{fry} = p_{frypan}$

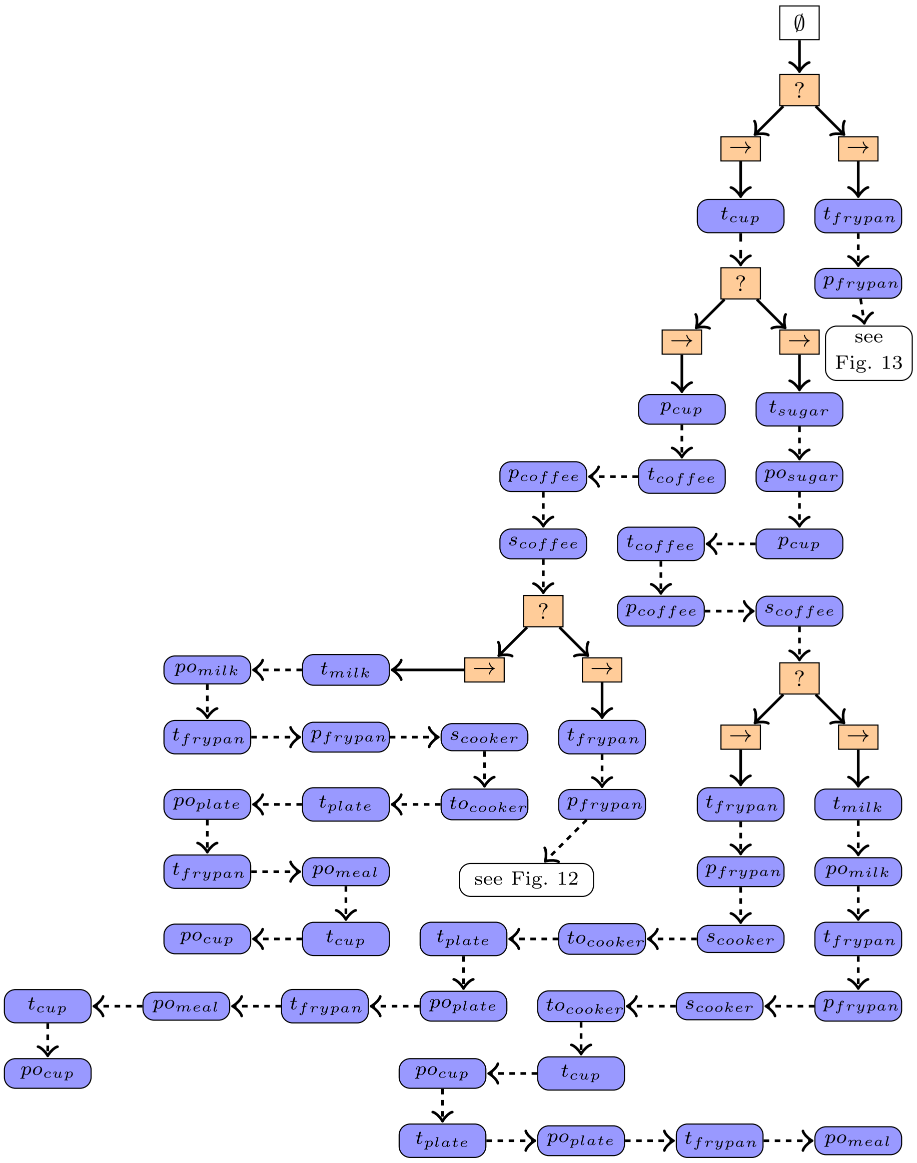


Fig. 11 BT obtained by applying Algorithm 1 to the data in Table 5

this has never been done. We can see this on the logs. This is a difference with the original tree, but the generated tree highlights this irregular characteristic of the action.

The activity "make a coffee" has similar characteristics to the activity "make a meal." It consists of a sequence of actions with 4 non-regular actions (these actions are linked to the use of milk and sugar or not). We therefore find in the generated tree similar characteristics to the treatment of the activity "make a meal" for this activity. These characteristics consist in the presence of the "selector" and "sequence" operator structure and the non-presence of this structure on all branches, because in the logs we do not have all the potential cases.

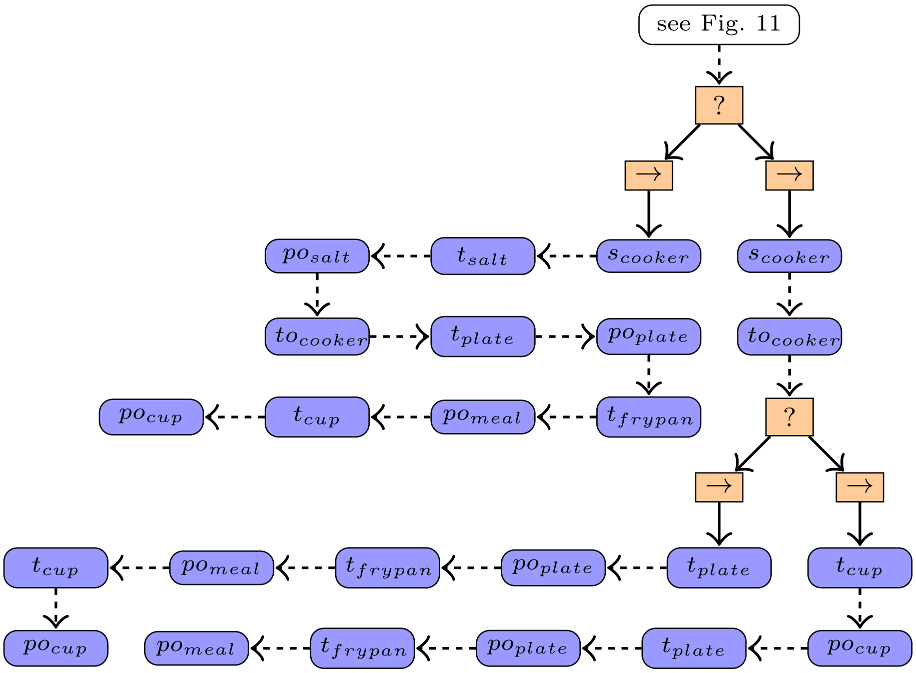


Fig. 12 BT obtained by applying Algorithm 1 to the data in Table 5

Finally, the last activity is to “set the table.” This consists of two sub-activities that can be performed in any order (place the cup and place the meal). Again, we can notice that when both variations have been made, the curent branch is divided into two parts. However, this is not the case for all branches, as the simulation did not cover all possible cases.

The second tree generated is smaller than the first one in particular, because there are no longer any non-regular actions. It starts with the same structure as the first tree generated, with a “selector” and two “sequences,” because the individual can start by making a meal or a coffee. Then, each branch is divided into two new branches, as the individual finishes the activity in two ways (place the cup first or the meal). This tree covers 4 ways of doing the activity just like the original tree; we can consider the two trees as equivalent (Fig. 14).

6 Discussions

We noticed that the first original tree and the first generated tree are not strictly equivalent, in particular, because the original tree did not generate a log containing all possible cases. The tree generated from this log does not cover cases that have not been produced. The first original tree can generate 32 different sequences. The first generated log shows 11 possibilities or 34.375% of the possibilities, so our tree covers 34.375% of the possibilities of the first original tree.

If we learn on a log with all 32 possibilities we get a bigger tree, but one that will be equivalent to the original tree. However, the flexibility of the behavior trees allows us to develop the one we have obtained in order to cover all possibilities. Indeed, if we generate a

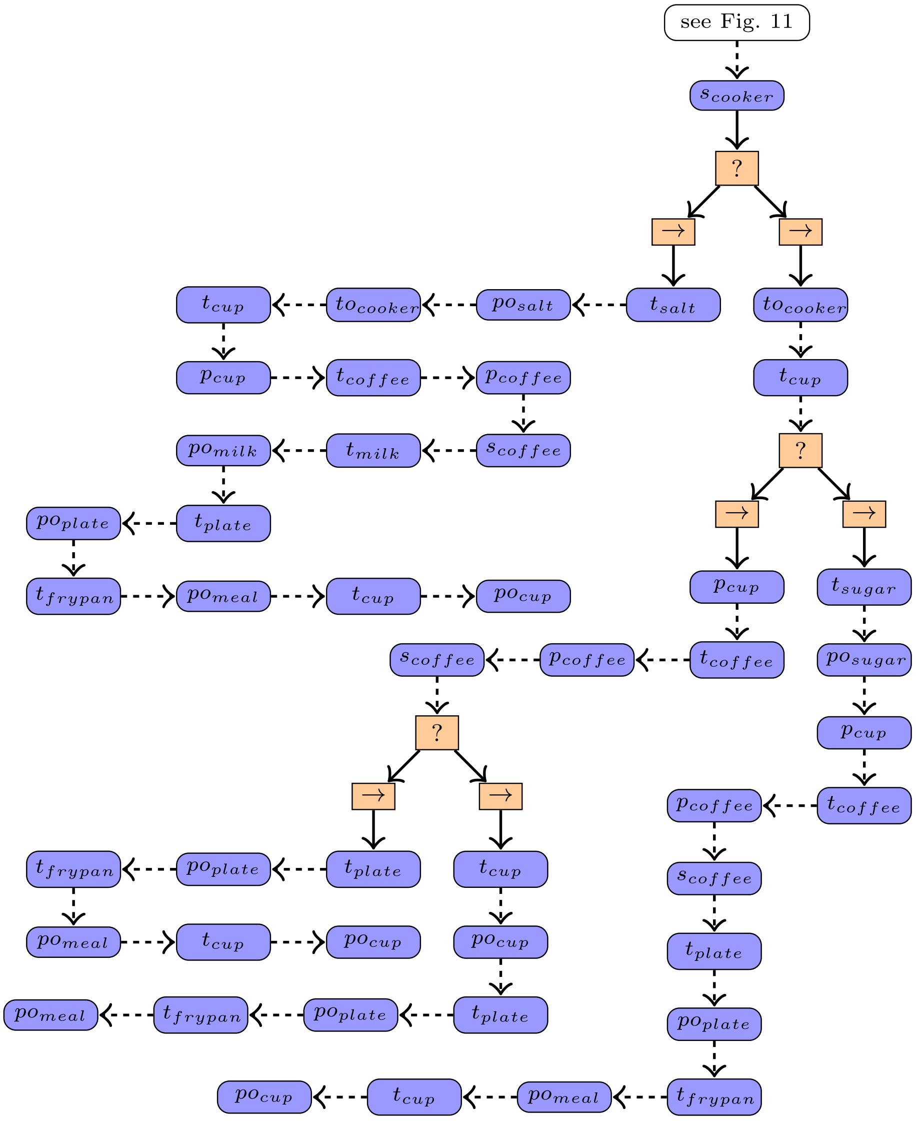


Fig. 13 BT obtained by applying Algorithm 1 to the data in Table 5

new log containing new sequences, we can use an algorithm that will verify for each sequence if it is present in the current tree and if not, it will add a branch to cover the sequence.

We can notice in our logs that the action sequences do not have the same frequencies. To improve the matching of the generated behavior tree to the reality represented by the log, we can learn these frequencies in order to assign them to the "selectors" that control the choice of sequences. For example, we can see in our first log that the scenario starts 11 times out of 25 times with the activity "make a coffee." We can therefore learn this frequency to associate it with the first "selector." In addition, by computing the time of each action, we can calculate an average time that will be assigned to the action nodes in the tree.

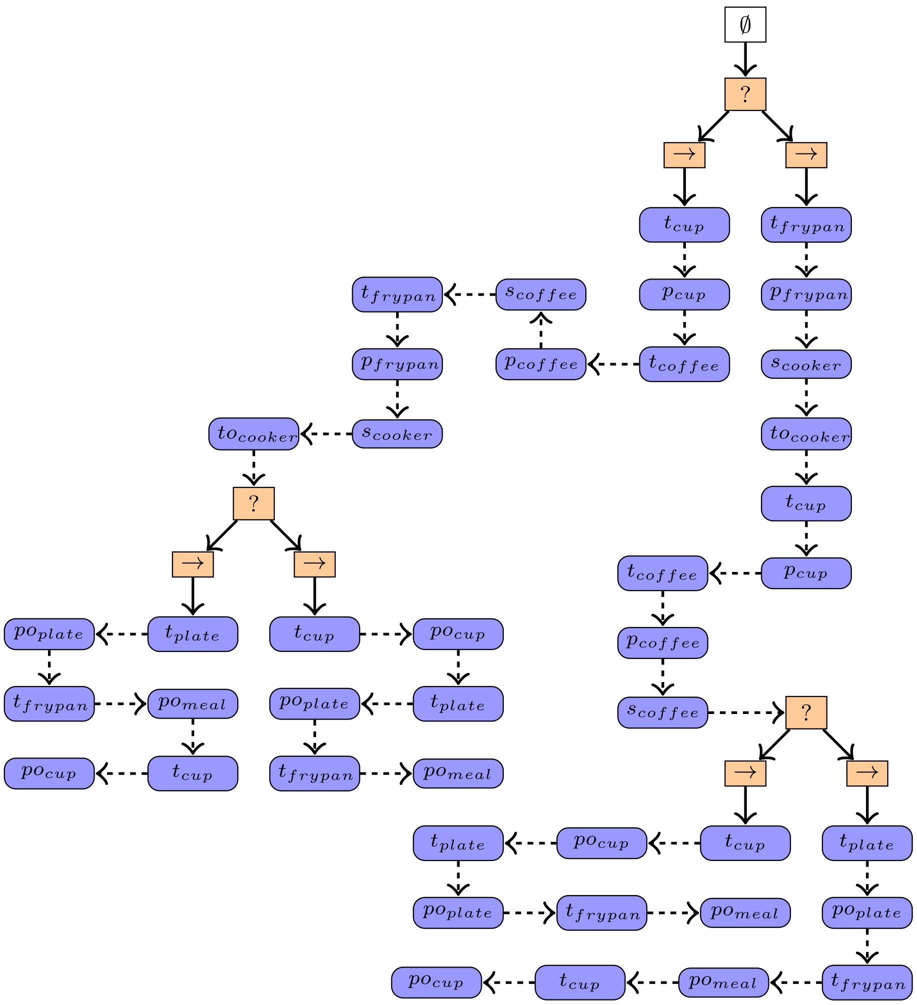


Fig. 14 BT obtained by applying Algorithm 1 to the data in Table 6

Finally, the flexibility of the model allows us to easily create new scenarios from the behavior trees already learned. To do this, we can add action nodes to the trees already learned, but also combine several trees to create a new behavior from the old ones. For example, we can combine the two trees in the previous section to create a tree that will cover the possibilities covered by these two trees.

Finally, the fact that the generated tree is large enough (in our example they are larger than the original trees) raises the question of reducing the size of the trees. Indeed, in order to make it easier to read, we can try to reduce the tree in order to reduce the number of nodes. To reduce the tree, a first idea is to search in two sibling branches for identical subsequences (the actions and their order must be identical) and then to bring this subsequence up to the same level as the branches by adding a new "sequence" operator. Each original branch will be separated into two branches.

7 Conclusion

The impact of having an increasingly aged population will have, in the next decades, significant consequences on the healthcare system in many developed countries [59]. Because most countries want to promote the concept of living at home longer, it is important to find news innovative and technological solutions to help seniors stay in their residence in a safe and comfortable space. Ambient intelligence [52], with its application in Smart Homes [31], constitutes one of the best opportunities to achieve the dreams of people aging at home in an ambient assisted living [9] environment. However, as we have described in Introduction, before being able to deploy such systems, scientists and engineers are facing three key research challenges, which are: (i) how to represent, in the sense of knowledge engineering, the computer model of an activity of daily living (ii) how to be able to automatically create the library of models based on observation of the resident's behavior, and (iii) how to be able to rigorously test the assistive algorithm by conducting experiments in their laboratory with simulated scenarios or with recorded data.

In the paper, we addressed these three challenges and proposed a solution that brings answers to theses issues. To address the first challenge, we proposed an approach to model human activities, based on behavior trees, that is easily interpretable and flexible [12]. This approach is inspired by the research done in the videogame industry to model the behavior of artificial players. Thereafter, we presented a solution to the second challenge, in the form of nan-adapted learning approach for automatically generates behavior trees from low-level action recognition corresponding to the occupant behavior. In previous works [7,25], we have already shown how to recognize and transform sensors' data into semantically significant low-level actions. This simple and efficient approach allows constructing an entire library of activity models generalizing the behavior of a specific user. Finally, to answer the last challenge, we introduce a new way of simulating user's behavior using these two formal tools in combination with our virtual smart home simulator, which is an open-source tool freely downloadable online.[2] The tests that we conducted on our global solution showed that our formal tools can be used in a smart home environment to answer the three challenges.

A limitation of our proposal come from designed scenarios that has been automatically generated. While our generation parameters and crafting has been based on previous experimentation with real people, it is not as accurate as reality. Moreover, the system could benefit of been improved in multiple way by applying, for the further design steps, a co-design approach implying targeted end users. Of course, in the near future, more tests will be needed to assess the potential and the weakness of the proposed tools. By giving our tools freely online, we sincerely hope that many scientists and engineers will use them to simulate a real environment and to develop new AI algorithms. Their feedback will be precious to orientate the future development of our LIARA Smart Homes BT Global Kit. On our side, we already planned to use the software solution in experiments with real users in order to test the approach on a large scale and with real targeted users.

References

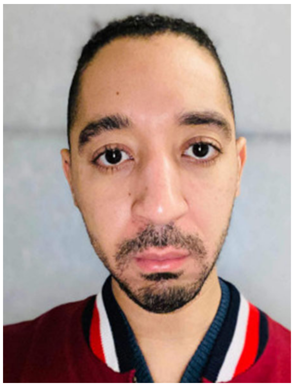
1. Al-Shaqi R, Mourshed M, Rezgui Y (2016) Progress in ambient assisted systems for independent living by the elderly. SpringerPlus 5(1):624
2. Alshammari N, Alshammari T, Sedky M, Champion J, Bauer C (2017) OpenSHS: open smart home simulator. Sensors 17(5):1003

[2] https://github.com/Iannyck/shima.

3. Ariani A, Redmond SJ, Chang D, Lovell NH (2013) Simulation of a smart home environment. In: 3rd International conference on instrumentation, communications, information technology, and biomedical engineering (ICICI-BME). IEEE, pp 27–32
4. Artikis A, Sergot M, Paliouras G (2010) A logic programming approach to activity recognition. In: Proceedings of the 2nd ACM international workshop on events in multimedia. ACM, pp 3–8
5. Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (eds) (2003) The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York
6. Barwise J, Perry J (1981) Situations and attitudes. J Philos 78(11):668–691
7. Belley C, Gaboury S, Bouchard B, Bouzouane A (2015) Nonintrusive system for assistance and guidance in smart homes based on electrical devices identification. Expert Syst Appl 42(19):6552–6577
8. Bergeron F, Giroux S, Bouchard K, Gaboury, S (2017) RFID based activities of daily living recognition. In: IEEE SmartWorld, ubiquitous intelligence computing, advanced trusted computed, scalable computing communications, cloud big data computing, internet of people and smart city innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp 1–5. https://doi.org/10.1109/UIC-ATC.2017.8397548
9. Blackman S, Matlo C, Bobrovitskiy C, Waldoch A, Fang ML, Jackson P, Mihailidis A, Nygård L, Astell A, Sixsmith A (2016) Ambient assisted living technologies for aging well: a scoping review. J Intell Syst 25(1):55–69
10. Botía JA, Campillo P, Campuzano F, Serrano E. UbikSim website. https://github.com/emilioserra/UbikSim/wiki. Accessed 28 May 2020
11. Bouchard B (2017) Smart technologies in healthcare. CRC Press, Boca Raton
12. Bouchard B, Gaboury S, Bouchard K, Francillette Y (2018) Modeling human activities using behaviour trees in smart homes. In: Proceedings of the 11th PErvasive technologies related to assistive environments conference. ACM, pp 67–74
13. Bouchard B, Giroux S, Bouzouane A (2007) A keyhole plan recognition model for alzheimer's patients: first results. Appl Artif Intell 21(7):623–658
14. Bouchard K, Ajroud A, Bouchard B, Bouzouane A (2012) Simact: a 3d open source smart home simulator for activity recognition with open database and visual editor. Int J Hybrid Inf Technol 5(3):13–32
15. Bouchard K, Bouchard B, Bouzouanea A (2017) Practical guidelines to build smart homes: lessons learned. In: Hasan SF (ed) Opportunistic networking: vehicular, D2D and cognitive radio networks. CRC Press, Boca Raton, pp 206–234
16. Bouchard K, Fortin-Simard D, Gaboury S, Bouchard B, Bouzouane A (2013) Accurate rfid trilateration to learn and recognize spatial activities in smart environment. Int J Distrib Sens Netw 9(6):936816
17. Camilleri G (1999) A generic formal plan recognition theory. In: International conference on information intelligence and systems. Proceedings. IEEE, pp 540–547
18. Carberry S (2001) Techniques for plan recognition. User Model User Adap Interact 11(1–2):31–48
19. Charniak E, Goldman RP (1993) A bayesian model of plan recognition. Artif Intell 64(1):53–79
20. Chen L, Hoey J, Nugent CD, Cook DJ, Yu Z (2012) Sensor-based activity recognition. IEEE Trans Syst Man Cybern Part C (Appl Rev) 42(6):790–808
21. Chen W, Augusto JC, Seoane F (2015) Recent advances in ambient assisted living-bridging assistive technologies, e-health and personalized health care, vol 20. IOS Press, Amsterdam
22. Cilla R, Patricio MA, García J, Berlanga A, Molina JM (2009) Recognizing human activities from sensors using hidden markov models constructed by feature selection techniques. Algorithms 2(1):282–300
23. Fahad LG, Ali A, Rajarajan M (2015) Learning models for activity recognition in smart homes. In: Kim KJ (ed) Information science and applications. Springer, Berlin, pp 819–826
24. Fleury A, Vacher M, Noury N (2010) Svm-based multimodal classification of activities of daily living in health smart homes: sensors, algorithms, and first experimental results. IEEE Trans Inf Technol Biomed 14(2):274–283
25. Fortin-Simard D, Bilodeau JS, Bouchard K, Gaboury S, Bouchard B, Bouzouane A (2015) Exploiting passive RFID technology for activity recognition in smart homes. IEEE Intell Syst 30(4):7–15
26. Fortin-Simard D, Bilodeau JS, Gaboury S, Bouchard B, Bouzouane A (2015) Method of recognition and assistance combining passive RFID and electrical load analysis that handles cognitive errors. Int J Distrib Sens Netw 11(10):643273
27. Francillette Y, Boucher E, Bouzouane A, Gaboury S (2017) The virtual environment for rapid prototyping of the intelligent environment. Sensors 17(11):2562
28. Gadelhag Mohmed Ahmad Lotfi CLAP (2018) Unsupervised learning fuzzy finite state machine for human activities recognition. In: Conference: the 11th PErvasive technologies related to assistive environments conference, pp 1–8
29. Geib CW, Goldman RP (2005) Partial observability and probabilistic plan/goal recognition. In: Proceedings of the international workshop on modeling other agents from observations (MOO-05), vol 8

30. Giegerich R, Kurtz S (1997) From ukkonen to mccreight and weiner: a unifying view of linear-time suffix tree construction. Algorithmica 19(3):331–353
31. Giroux S, Pigot H (2013) Smart homes for people suffering from cognitive disorders. In: Computer science and ambient intelligence, pp 225–262
32. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. Adaptive computation and machine learning series. MIT Press, Cambridge, p 800
33. Grześ M, Hoey J, Khan SS, Mihailidis A, Czarnuch S, Jackson D, Monk A (2014) Relational approach to knowledge engineering for POMDP-based assistance systems as a translation of a psychological model. Int J Approx Reason 55(1):36–58
34. Helal A, Cho K, Lee W, Sung Y, Lee J, Kim E (2012) 3d modeling and simulation of human activities in smart spaces. In: 9th International conference on ubiquitous intelligence and computing and 9th international conference on autonomic and trusted computing (UIC/ATC). IEEE, pp 112–119
35. Ho B, Vogts D, Wesson J (2019) A smart home simulation tool to support the recognition of activities of daily living. Proc S Afr Inst Comput Sci Inf Technol 2019:1–10
36. Humphreys G, Forde E (1998) Disordered action schema and action disorganisation syndrome. Cogn Neuropsychol 15(6):771–812
37. Kautz HA (1991) A formal theory of plan recognition and its implementation. In: Reasoning about plans. Morgan Kaufmann Publishers Inc., San Francisco, pp 69–124. http://dl.acm.org/citation.cfm?id=117019.117021. Accessed 28 May 2020
38. Kingston A, Collerton J, Davies K, Bond J, Robinson L, Jagger C (2012) Losing the ability in activities of daily living in the oldest old: a hierarchic disability scale from the newcastle 85+ study. PLoS ONE 7(2):e31665
39. Krishnan NC, Cook DJ (2014) Activity recognition on streaming sensor data. Pervasive Mob Comput 10:138–154
40. Luke S, Cioffi-Revilla C, Panait L, Sullivan K, Balan G (2005) MASON: a multiagent simulation environment. Simulation 81(7):517–527
41. Lundström J, Synnott J, Järpe E, Nugent CD (2015) Smart home simulation using avatar control and probabilistic sampling. In: IEEE international conference on pervasive computing and communication workshops (PerCom workshops). IEEE, pp 336–341
42. Marcotte R, Hamilton HJ (2017) Behavior trees for modelling artificial intelligence in games: a tutorial. Comput Games J 6(3):171–184. https://doi.org/10.1007/s40869-017-0040-9
43. Mojidra HS, Borisagar VH (2012) A literature survey on human activity recognition via hidden Markov model. In: IJCA proceedings on international conference on recent trends in information technology and computer science, pp 1–5
44. Nazerfard E, Cook DJ (2012) Bayesian networks structure learning for activity prediction in smart homes. In: 8th International conference on intelligent environments (IE). IEEE, pp 50–56
45. Nazerfard E, Das B, Holder LB, Cook DJ (2010) Conditional random fields for activity recognition in smart environments. In: Proceedings of the 1st ACM international health informatics symposium. ACM, pp 282–286
46. Ni Q, Pau de la Cruz I, García Hernando AB (2016) A foundational ontology-based model for human activity representation in smart homes. J Ambient Intell Smart Environ 8(1):47–61
47. Park B, Min H, Bang G, Ko I (2015) The user activity reasoning model in a virtual living space simulator. Int J Softw Eng Appl 9(6):53–62
48. Paternò F (2004) ConcurTaskTrees: an engineered notation for task models. The handbook of task analysis for human–computer interaction, pp 483–503
49. Paternò F, Mancini C, Meniconi S (1997) ConcurTaskTrees: a diagrammatic notation for specifying task models. In: Human–computer interaction INTERACT'97. Springer, pp 362–369
50. Patterson DJ, Fox D, Kautz H, Philipose M (2005) Fine-grained activity recognition by aggregating abstract object usage. In: Ninth IEEE international symposium on wearable computers. Proceedings. IEEE, pp 44–51
51. Puybaret E (2016) Sweet home 3d. https://sourceforge.net/projects/sweethome3d/. Accessed 28 May 2020
52. Ramos C, Augusto JC, Shapiro D (2008) Ambient intelligence-the next step for artificial intelligence. IEEE Intell Syst 23(2):15–18
53. Rook A, Knauss A, Damian D, Thomo A (2014) A case study of applying data mining to sensor data for contextual requirements analysis. In: IEEE 1st international workshop on artificial intelligence for requirements engineering (AIRE). IEEE, pp 43–50
54. Roy PC, Abidi SR, Abidi SS (2017) Possibilistic activity recognition with uncertain observations to support medication adherence in an assisted ambient living setting. Knowl Based Syst 133:156–173

55. Roy PC, Bouchard B, Bouzouane A, Giroux S (2013) Ambient activity recognition in smart environments for cognitive assistance. Int J Robot Appl Technol (IJRAT) 1(1):29–56
56. Serrano E, Botia J (2013) Validating ambient intelligence based ubiquitous computing systems by means of artificial societies. Inf Sci 222:3–24. https://doi.org/10.1016/j.ins.2010.11.012
57. Synnott J, Chen L, Nugent C, Moore G (2014) The creation of simulated activity datasets using a graphical intelligent environment simulation tool. In: 36th Annual international conference of the IEEE on engineering in medicine and biology society (EMBC). IEEE, pp 4143–4146
58. Technologies U (2016) Unity-game engine. https://unity3d.com. Accessed 28 May 2020
59. United Nations D.o.E., Social Affairs, PD (2019) World population ageing 2019, p 46
60. Van Viamen AE (2018) Person-environment fit: a review of its basic tenets. Ann Rev Organ Psychol Organ Behav 5:75–101
61. Wang J, Chen Y, Hao S, Peng X, Hu L (2018) Deep learning for sensor-based activity recognition: a survey. Pattern Recognit Lett 119:3–11
62. Wang WY, Cohen WW (2016) Learning first-order logic embeddings via matrix factorization. In: IJCAI, pp 2132–2138
63. Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Burlington
64. Wobcke W (2002) Two logical theories of plan recognition. J Logic Comput 12(3):371–412
65. Ziaeefard M, Bergevin R (2015) Semantic human activity recognition: a literature review. Pattern Recognit 48(8):2329–2345



Yannick Francillette is a professor at the Université du Québec à Chicoutimi since August 2019. He received a Ph.D. in Computer Science from the University of Montpellier (France). He completed a postdoctoral fellowship at the LIARA laboratory of the Université du Québec à Chicoutimi in the field of intelligent environments. He worked on simulating intelligent environments and activities of daily life in smart houses. His main research interests are adaptive systems and serious games.

Bruno Bouchard is a full professor and a researcher at the LIARA laboratory of the University of Quebec at Chicoutimi (UQAC). He received a Ph.D. in computer science from the University of Sherbrooke (Canada) and he completed a postdoctoral fellowship at the University of Toronto (Canada) in 2007. He was the cofounder of the LIARA lab in 2008, which develops smart home technologies dedicated to people suffering from cognitive impairments. The lab conducts real size experiments in a smart home prototype infrastructure financed by the Canada Foundation for Innovation (CFI Leaders Opportunity Fund). Dr. Bouchard has received the precious support of multiples sponsors during his career, such as: NSERC, FQRNT, CFI, CIHR, Bell Canada, UQAC and multiples companies. His main research interests are ambient intelligence, smart environments, sensors, data mining, and activity recognition. Bruno Bouchard has been elevated to the rank of Senior IEEE member in 2016.

Kévin Bouchard is a professor at the Université du Québec à Chicoutimi since August 2016. He completed his doctorate in computer science in 2014. He carried out a postdoctoral fellowship on ambient intelligence for the support of people who suffered traumatic brain injuries at the Université de Sherbrooke before spending a short time at CASAS lab of Washington State University. He then worked as Project Scientist at the Center for SMART Health at the University of California Los Angeles where he co-led a deployment project of ambient technology for a rehabilitation center in Santa Monica. His research interests mainly relate to the exploitation of artificial intelligence and machine learning for the development of technologies for health. He has, among other things, worked on RFID localization, ambient sensing, activity recognition with UWB, wearable technologies, etc. He has published over 70 papers and has been supervising/co-supervising more than twenty graduate students.

Sébastien Gaboury received the Ph.D. degree in mathematics from the Royal Military College of Canada, Kingston, ON, Canada, in 2012. He is currently an associate professor and a Scientist with the Department of Mathematics and Computer Science, University of Quebec at Chicoutimi, Chicoutimi, QC, Canada, where he is also the head of the Ambient Intelligence Laboratory for the Recognition of Activities. His research is sponsored by the Natural Sciences and Engineering Research Council of Canada, the Québec Research Fund on Nature and Technologies, and the Canadian Foundation for Innovation. His current research interests include assistive technologies for elders and cognitively impaired people and artificial intelligence.