3-2016

Interactive teachable cognitive agents: Smart building blocks for multiagent systems

Budhitama SUBAGDJA

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Interactive Teachable Cognitive Agents: Smart Building Blocks for Multiagent Systems

Budhitama Subagdja and Ah-Hwee Tan, *Senior Member, IEEE*

***Abstract*—Developing a complex intelligent system by abstracting their behaviors, functionalities, and reasoning mechanisms can be tedious and time consuming. In this paper, we present a framework for developing an application or software system based on smart autonomous components that collaborate with the developer or user to realize the entire system. Inspired by teachable approaches and programming-by-demonstration methods in robotics and end-user development, we treat intelligent agents as teachable components that make up the system to be built. Each agent serves different functionalities and may have pre-built operations to accomplish its own design objectives. However, each agent may also be equipped with in-built social-cognitive traits to interact with the user or other agents in order to adapt its own operations, objectives, and relationships with others. The results of adaptation can be in the form of groups or multiagent systems as new aggregated components. This approach is made to tackle the difficulties in completely programming the entire system by letting the user teaches or exemplifies its components towards the desired behaviors in the situated context of the application. We exemplify this novel method with cases in the domains of human-like agents in virtual environment and agents for in-house caregiving.**

***Index Terms*—Learning Systems, Cooperative Systems, Software Engineering.**

I. Introduction

Developing a system consisting of interacting elements may require considerable efforts to specify how it behaves appropriately according to its desired properties. Commonly, the system's behavior and functionality are carefully considered and designed before it is realized in the running context. This issue is conventionally tackled by abstracting the problem and the system using a programming or modeling language. In most current software engineering practices, the abstractions are constructed through coding processes which then the desired system can only be separately executed or deployed.

Different engineering paradigms, like structured, functional, or object orientation methodologies, offer means to manage the intricacies by promoting modularity. Contemporary models like component-based, services-oriented architecture, and cloud computing have also offered reusability and flexibility. Another new paradigm considers multiple autonomous agents working together in the system. They interact and communicate with each other in the same environment. Multiagent System [18] can be considered as a set of computational agents each with its own objectives. They interact and communicate with each other in the same environment.

Although the above approaches have offered encapsulation that hides the developer from the intricate details letting one to focus on more important matter, the assumptions are still that the abstraction or program code are detached from the running context.

In this paper, we present a radical approach of software engineering in which the human user interacts directly with smart components that make up the system in its running context. The component can be considered as a situated (semi-) autonomous agent that can be taught by the user through instructions and examples. This approach is not just simplifying the development process but also allowing it to be conducted during its deployment. The system can be taught through instructions or demonstration while it interacts with the real user in the relevant context. Inspired by teachable and programming by demonstration approaches like in smart applications [12], [7] and robotics [2], the challenges to tackle include how to deal with incompleteness, inconsistency, and error-prone instructions or demonstrations provided by the user. The uncertainty of the instructor intention requires the learner to make guesses, inferences, predictions, or sometimes ask for clarification which are also the common issues to tackle in designing collaborative user interface [15], [12], [13].

The contributions provided in this paper are the software framework for developing interactive multiagent systems allowing end-users or non-technical experts to be dominantly involved in the development process by teaching or instructing each elementary unit (building block) that makes up the system. The teachable feature is applied not just to serve the user needs or tasks but also to allow a non-technical instructor to specify how each component of the system behaves and relates to each other. The instructor can communicate with each component or a group of components directly in order to instruct, inform, or ask something related to the tasks or operations in hand at runtime. To simplify the process, each component is attributed with intuitive mental notions (e.g belief, desire, intention) and procedural knowledge (e.g rules or plans) similar to BDI agent model [14] so that it can be perceived as an autonomous agent that can reason and understand the tasks by the instructor.

Parts of the novelty of the proposed approach include embeddable capabilities of learning and teachable features to a pre-existing domain level component with specific operations and functionalities so that it can be associated with mental notions of itself and other agents. Thus, the instruction can

B. Subagdja is with Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly, Nanyang Technological University, Singapore (e-mail: budhitama@ntu.edu.sg)

A.-H. Tan is with School of Computer Engineering, Nanyang Technological University, Singapore (e-mail: asahtan@ntu.edu.sg)

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office.

be targeted to particular components so that they can reason about and coordinate with each other. Similar notions are used to represent users and instructors so that the teachable features can handle multiple targeted end-users and collaborating instructors as well. Each component may also deal with uncertainties in the instructions by pro-actively asking the instructor for confirmation or clarification. It is worth noting that the proposed framework does not emphasize the intelligence capability of a single component to solve the problem. Instead, the use of multiple components and their interactions with the instructor and each other as a whole may produce the solution to the complex domain problem.

As proofs-of-concept, we demonstrate how the framework can ease the development of adaptive software or applications in their actual runtime context. Simulations of virtual campus and aging-in-place demonstrate that the components can be taught to respond in various ways to different contexts of end-user behavior in virtual environment.

This paper is organized as follows. Section II discusses existing related works. Section III describes the proposed framework and the mechanisms of the teachable components. Section IV exemplifies the implemented models.

II. Related Work

Teachable systems and applications for end user development have been around for decades to enable a lay person to demonstrate the process in accomplishing the tasks of using the application [12], [7]. The system's responses to the demonstration can range from a shallow-level of macro-recording to a deep generalization of the user actions. Usually, software applications employing this feature are made to learn procedures which are commonly performed manually by the user.

Most of them are made as assisting agents that watch the user interaction "over-the-shoulder" while creating the model of the user and the plan to accomplish the task. These agents require a model of cognition to reason about the user intention and the tasks taught to accomplish. Various types of cognitive architecture (e.g SOAR [11], ACT-R [1], BDI (Belief Desire Intention) framework [14]) have been used as the reasoning engine that drives the reasoning processes of the agents. Some of these cognitive models may also support learning by generalizing the decisions and outputs to deal with the unanticipated conditions. For example, in SOAR [11], new production rules may be created that summarize the processes that solve the *impasse* conditions. Other models suggest adaptation of procedural knowledge or recipes in BDI architecture through the execution of meta-level plans initiated by similar *impasse* conditions [16] or by induction of decision trees [8]. All these learning are assumed to be conducted autonomously in which the agent decides when to learn and to generalize knowledge by itself based on particular algorithms or programmed knowledge. The reasoning processes comprised in the learning still need to be described using a particular symbolic representation or programming language, even though the language itself can be generic, abstract, and expressive for many different real situations.

Most, if not all, of teachable applications with cognitive models mentioned above are made to be able to adapt with the needs of a single user. In this case, the main assumption is that there is only a single intelligent agent handling all the processes of learning from the user and the applications of all the learnt knowledge. Consequently, all computational and adaptation burdens must be handled by a single entity of computational agent. Moreover, the goal of the learning (or demonstration) is mainly to help or support the tasks of the end-user.

In robotic domain, programming by demonstration has been considered to be the approach beyond learning for a particular task. Since a robot may include a complex intertwining set of operations, demonstrating a robot requires more variations of interactions and mechanisms [2]. In contrast to teachable software or applications, the aim of a demonstrable robot is usually to aid the development of the robot behavior rather than to help the end user to accomplish the tasks. Beyond a single agent that "watches over-the-shoulder", robotic programming by demonstration has been considered to include multiple robots to be taught simultaneously in a coordinated fashion [6], [3]. In this case, the robots must identify the instructor intention and coordinate with each other to realize the desired objectives. The current multi-robots learning still adopts the autonomous learning that the user or external human interventions must be minimal. Although it can relieve the user from extensive involvements in the development process, letting agents learn by themselves may also produce undesirable effects. Given that incoming information or examples may be incomplete or erroneous, it is useful to let the user know about the actual conditions internally so that user can react to it and fix the problem immediately. Instead of relying on itself, the agent may get help from the user or a human participant.

In online and interactive programming by demonstration, the robot may also provide feedback in terms of social cues to help the developer to understand the internal state of the robot without looking at the specific trace of its running program. For example, one approach employs emotive cues on the robot's embodiment to indicate the condition of learning [4] (e.g thinking, confused, excited) so that the teacher may adapt his/her teaching strategy when needed. Active Learning (AL) approach can also be employed to let the robot use different types of queries asked at the right context [5].

The proposed teachable framework in this paper adopts a similar objective to most demonstrable robotics approaches that the teaching-learning process is to simplify and ease the development of robots behavior but to employ it in software or application development. We employ BDI model to let the components in the system predict and anticipate the user intention from instructions and provide meaningful feedbacks regarding the development. The instructions can be targeted to particular components or groups of component while they are predicting the user's intention and inferring the operations to learn. This approach lets the whole system to be developed through a collaborative process among the smart components and the user.

III. Agent Architecture and Teachable Framework

In this section, we describe the teachable framework which includes the software architecture, learning mechanism, and protocols. A teachable component is a wrapper object that encloses a domain-specific object or service that may interact directly with the domain environment. It encapsulates all methods and functionalities related to the task domain.

A. Agent Architecture

A teachable component is an active instantiation of a running object or program that can be customized or reconfigured by the user at runtime. The component may sense and take action on its domain environment autonomously. However, the user may communicate with the component to instruct or teach new tasks on the run. A domain specific object is enclosed within the component to serve some prescribed functionalities. Over time, the object may become more autonomous that some operations may be initiated or suppressed when certain conditions or events occur.

Figure 1(i) illustrates the teachable wrapper as an agent. As a part of the teachable component, *agent façade* controls the operations and execution of the domain object. It may receive status update or feedback from the domain object as a result of an operational call or control. It also makes the component teachable by the user (developer) and is able to communicate with other agents.

The *façade* maintains a collection of data structures representing BDI [14] mental notions (*belief*, *desire*, *intention*, and *schema*). The agent also maintains a finite trace of events and actions in *episodic buffer*. The trace facilitates learning and keeps the dialogs with the user and other agents in context. Figure 1(ii) shows the internal architecture of the agent façade with all the data structures. It receives all incoming events of the domain specific object. It can also access possible actions produced by the domain object to manipulate or update the state of the domain environment. In this way, the *façade* knows every signal received and every action initiated by the domain object. On the other hand, the *façade* may control the domain object by initiating an action directly without the corresponding events to trigger. It may also suppress or halt the execution initiated by the domain object.

Event and Assertion Representation

An event or action can be stated as an assertion. An assertion consists of attributes each specifies a condition that holds at a moment. An assertion $P = \{p_1, .., p_m\}$ is a set of attributes and $P \in \mathcal{V}$ in which $\mathcal{V}$ is the set of all possible assertions. Attribute $p_i \in P$ can be expressed as $p_i = (\eta(p_i) : v(p_i))$ wherein $\eta(p_i)$ is the name or identifier of p_i and $v(p_i)$ is the value or content of p_i. For example, p_i can be expressed as $(\texttt{age} : 50)$ to represent an attribute specifying that `age` is 50. $v(p_i)$ can be a number or a text data. An attribute can also recursively be a list of assertions. This nested attributes-values representation make the assertion expressive enough to describe situations in the world and internal states of agents.

To measure how much an assertion implies or contained by another, *implicative match* $\mathbf{m}_P$ can be defined such that, given two different assertions P_i and P_j, $\mathbf{m}_P$ can be measured as follows

$$\mathbf{m}_P(P_q, P_r) = \frac{\sum_{k=1}^{N} \|v(p_k^q)\|.(1 - |\delta(v(p_k^q), v(p_k^r))|)}{\sum_{k=1}^{N} \|v(p_k^q)\|}, \quad (1)$$

wherein $|v|$ is the absolute value of v and $\|v(p)\|$ is the total value or magnitude of attribute p. For a number-typed attribute p, it can be simply written as $\|v(p)\| = v(p)$. However, a different type of value may require a different way to evaluate. The magnitude of a text or string attribute can be equivalent to its number of characters or word. $\delta(v(p_k^q), v(p_k^r))$ above measures the difference between $v(p_k^q)$ and $v(p_k^r)$. For number-typed values, the difference can be $\delta(v(p_k^q), v(p_k^r)) = v(p_k^q) - v(p_k^r)$. For text or string attributes, it can be based on the number of characters co-occured in both assertions with similar order. When the assertion consists of nested attributes, the difference and the total value can be measured recursively.

The *implicative match* enables one assertion to be evaluated in an inexact manner to know whether it implies another or not. One assertion can be defined to *approximately imply* another for a certain matching treshold such that

$$P_q \tilde{\Rightarrow}_\rho P_r \;\equiv\; \mathbf{m}_P(P_q, P_r) \geqslant \rho. \quad (2)$$

In this case, ρ is the *vigilance parameter* or the matching threshold that P_q can be considered to (approximately) imply P_r. For example, $P_q \tilde{\Rightarrow}_1 P_r$ holds when both P_q and P_r have the same assertion as $\{\texttt{subject} : \texttt{"user01"}, \texttt{age} : 25, \texttt{gender} : \texttt{"female"}\}$[1]. $P_q \tilde{\Rightarrow}_1 P_r$ still holds when $P_q = \{\texttt{subject} : \texttt{"user01"}, \texttt{age} : 25\}$. With a smaller ρ, a small discrepancy may still be tolerated, so that, for instance $P_q \tilde{\Rightarrow}_{0.75} P_r$ still holds when $\texttt{age} : 20$ is in P_q.

Given two different but similar assertions P_q and P_r, their values can be generalized by the function $\text{Gen}(P_q, P_r)$ such that

$$\text{Gen}(P_q, P_r) = \{p_k^z : v(p_k^z) \leftarrow \text{gen}(p_k^q, p_k^r)\} \quad (3)$$

$$\text{gen}(p_k^q, p_k^r) = \begin{cases} v(p_k^q)\} & \text{if } v(p_k^q) = v(p_k^r) \\ ? & \text{if } v(p_k^q) \neq v(p_k^r) \\ \text{null} & p_k^r \notin P_r, \end{cases} \quad (4)$$

where "?" is a *don't-care* condition that the attribute will always match (and be bounded) with any value of *implicative match*. The null value indicates the omission of the corresonding attribute from the generalized set. For example, if $P_q = \{\texttt{a} : 10, \texttt{b} : 5, \texttt{c} : 15\}$ and $P_r = \{\texttt{a} : 10, \texttt{b} : 3\}$, then $\text{Gen}(P_q, P_r) = \{\texttt{a} : 10, \texttt{b} : \texttt{"?"}\}$.

The generalization function $\text{Gen}(P_q, P_r)$ enables learning a new assertion based on the generalization of two different assertions (P_q and P_r).

BDI Mental Representation

Belief

The *Belief* data structure can be considered as a collection of assertions believed to hold or happen. For example, a belief

[1]The curve brackets for attributes are omitted for brevity

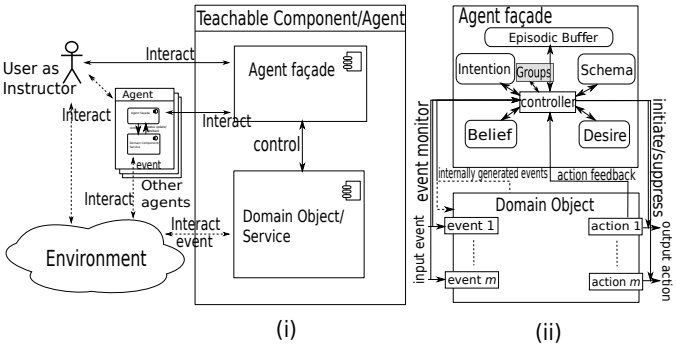

Fig. 1. The Agent wrapper and its relationship with the user, other agents, and the environment

assertion {start : 10.30, location : "corridor"} may denote that the agent believes that starting at time 10.30, the location is in "corridor". It is retrospective when 10.30 indicates time in the past or it may be predictive if it is in the future. The assertion may also describe the action or operation conducted including its parameters or arguments. For example, {do : "walk", distance : "5", unit : "step"} indicates that the agent has just walked 5 steps. The belief assertion may also be associated with another individual agent or group including the description of other mental states. For example, {belief : {do : "walk", distance : "5", unit : "step"}, agent : ["agent-01"]} denotes the agent believes that another agent agent-01 believes it has just walked 5 steps. Here, the walk action is not about its own action but the action of the other (agent-01). The agent attribute refers to the group of agents having the corresponding mental notion. In the example above, agent is a singleton referring to a single agent (agent-01). As illustrated in Figure 1(ii), the information about all known agents are maintained in Groups so that the agent's assertions may refer to another agents as well.

Desire

The *Desire* data structure consists of assertions each represents something not held yet or still wanted to be achieved. For example, a desire assertion {do : "walk", distance : "5", unit : "step"} means that the agent has a desire to walk 5 steps, though it does not happen yet. The agent will no longer have the same desire when the assertion is already believed (or held in the *belief* data structure) . Just like in *belief*, a desire assertion may refer to another agent's (or group's) mental states.

Intention

The *intention* structure can be represented in two forms as follows:

- *rule-based* intention with attributes specifying triggering conditions and the actions that will be initiated or executed when triggered. The general structure is as follows:
  {trigger-cond : {< trigger attributes >},
  actions : {< actions that follows >}}.
- *plan-based* intention with attributes specifying the desired condition to achieve, precondition that makes it applicable (or executable), and the actions sequence to execute. Its general structure is as follows:
  {goal : {< conditions to achieve >},
  precond : {< pre-conditions >},
  actions : {< actions to execute when applicable >}}.

An incomplete assertion of intention (e.g a rule assertion without the trigger attribute or a plan goal without actions) will initiate a search process or reasoning to complete the assertion. For example, when {goal : {location : "classroom"}} is put in the *intention*, a search process will be initiated to find the actions part (e.g through means-end reasoning, or asking the instructor to inform or to teach the actions). In this case, the assertion of *intention* may contain actions specification for immediate execution or desired conditions pending achievement. The actions attribute may have a structure as follows:
{actions : { sequence : [< action 1 >, ..., < action n >] } to represent a sequence. The actions will be executed one after another in consecutive order. The sequence attribute is changing when the next action to execute is unfolding and the previously executed actions is removed.

Schema

Similar to *intention*, the *schema* repository consists of rules and/or plans. A schema assertion can be a generalized or abstracted version of assertions in *intention*. In other words, an intention is an instantiation of a schema as a recipe for actions. Given P_i as an assertion of intention instantiated from a schema assertion P_s, it holds that $P_s \Rightarrow_1 P_i$. In this teachable framework, the target of learning are new updated schemas. A plan-based schema may be selected and instantiated as an intention when its goal attribute is implied by a desire selected in the intention and its precondition implies those in *belief*.

Deliberation and Execution Cycle

As a BDI agent, all the data structures are continually interpreted and updated in a deliberation cycle [14]. It consists of basic operations such as *belief updating*, *generating options* to achieve, *deliberation* by weighing the options, *means-end reasoning* and *execution control*. The continual interpretation suggests that operations included in the execution cycle above are conducted concurrently. A step of operation or control may wait and require another operation to finish. However, independent operations can be executed concurrently. This also allows incoming input to be received and processed asynchronously.

Let $\mathcal{B}$, $\mathcal{D}$, $\mathcal{I}$, Π, and M be the collection of belief, desire, intention, schema, and episodic buffer respectively corresponding to the structures in Figure 1(ii). Algorithm 1 shows the abstract pseudo-code of the adapted deliberation cycle. Belief $\mathcal{B}$ is updated based on events from the environment and other related agents ($\mathcal{E}$ and $\mathcal{E}_A$). In general, the execution loop goes through the normal cycle of basic operations in the deliberation cycles as mentioned above.

Algorithm 1: Extended BDI Deliberation Cycle

1 **WHILE** True
2 Receive events $\mathcal{E}_A$ (from agents A) and/or $\mathcal{E}$ (from domain environment)
3 $\mathcal{B} \leftarrow$ `belief_update`$(\mathcal{B}^{(old)}, \mathcal{E}, \mathcal{E}_A)$ /*belief update */
4 $\mathcal{D}' \leftarrow$ `options`$(\mathcal{D}, \mathcal{B}, \mathcal{I})$ /* options generation */
5 $\mathcal{D} \leftarrow$ `filter`$(\mathcal{D}', \mathcal{B}, \mathcal{I})$ /* filter options and deliberation */
6 $\mathcal{I} \leftarrow$ `planning`$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \Pi)$ /* means-end reasoning */
7 $\mathcal{I}' \leftarrow$ `execute`$(\mathcal{I})$ /* execute $\mathcal{I}$ and return all intentions that end ($\mathcal{I}'$) */
8 $M^{(new)} \leftarrow$ `append_episode`$(M, \mathcal{B}, \mathcal{D}', \mathcal{D}, \mathcal{I}, \mathcal{I}', \mathcal{E}, \mathcal{E}_A, t)$

The façade may receive both the component's domain-specific and the agent's domain independent events to update its belief. As shown in Algorithm 1, the `belief_update` function can make some changes to the belief set $\mathcal{B}$ based on the received domain-related events ($\mathcal{E}$) and events about the other agents ($\mathcal{E}_A$). The events $\mathcal{E}$ and $\mathcal{E}_A$ enable the agent to not just perceive the environment but also identify or recognize other agents' intentions. Other functions like `options`, `filter`, `planning`, and `execute` generate information and control the execution as mentioned above.

The `planning` function selects the applicable plans that match with the selected desire ($\mathcal{D}$) and based on the criteria that `goal`$(P_{plan_x}) \hat{\Rightarrow}_{\rho_d} P_d$ and $P_b \hat{\Rightarrow}_{\rho_b}$ `precond`(P_{plan_x}) wherein `goal`(P_{plan_x}), `precond`(P_{plan_x}), $P_d \in \mathcal{D}$, and $P_b \in \mathcal{B}$ are the goal attribute of the selected plan assertion P_{plan_x}, precondition attribute of P_{plan_x}, a desire assertion, and a belief assertion respectively.

One distinction in the execution cycle is the `append_episode` function that keeps all the generated information on the record in *episodic buffer*. Time point t is used as an argument in the function to indicate the timing and so the sequential order in the execution loop.

Besides the domain level actions, which is expressed as {do :< action name >, ...}, there are generic meta-level actions that can be initiated by the agent façade as shown in Table I to update the data structures above and to control the execution. Each new information generated or updated like the events, belief $\mathcal{B}$, possible options $\mathcal{D}'$, desire $\mathcal{D}$, intention $\mathcal{I}$, execution status $\mathcal{I}'$, and schema Π can be updated directly through the meta actions. By the meta actions, rule-based or plan-based intentions (and schemas) can also be inserted on-the-fly to update the data structures and control the execution as above. In this case, schemas can be made not just for domain level actions but also for learning and reasoning processes.

TABLE I
FAÇADE META-LEVEL ACTIONS

Action name	Description
`insert` **P**	insert assertion **P** to a data structure
`remove` **P**	remove **P** from a data structure
`stop` **P**	stop **P** from execution, drop it from intention, and post an event indicating that **P** is stopped
`halt` **P**	pause the execution of **P** and set the intention status to `paused`
`resume` **P**	resume the execution of **P** and remove the `paused` status
`fail` **P**	stop **P** from execution, drop it from intention, and post an event indicating the failure of **P**
`success` **P**	stop **P** from execution, drop it from intention, and post an event indicating the success of **P**

B. Interactive Learning

Most aspects of the components' behavior can be taught by the user on the run. Through the user interactive dialogs and demonstration, the component, as an agent, can acquire new behaviors by updating the schema. An agent follows what is instructed, but the agent may ask the instructor for confirmation or for more information.

Instruction Protocol

Figure 2(i) shows the general protocol of interaction between the instructor and the agent. The messages exchanged between the two are as follows:

- From the instructor to the agent: *instruct* message tells the agent to perform some actions; *inform* message tells the agent about information or a fact to believe or as a response to a question asked by the agent; and *ask* message indicates an inquiry to ask for information about why, who, when, where, or what known or believed by the agent.
- From the agent to the instructor: *inform* message tells the instructor about particular information or as a response to the instructor question; and *ask* message indicates an inquiry from the agent regarding an instruction or activity conducted by the agent. The agent can be asking for confirmation whether something is true (agreed) or not, asking the instructor to choose from multiple choices, or asking for help to accomplish a task.

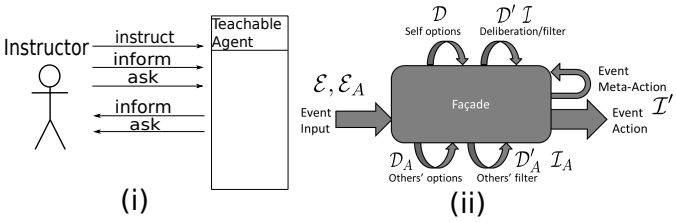

Fig. 2. (i) general protocol of instruction for teachable agent; (ii) continual generations of self-options and others' options in the façade execution cycle.

An event can be represented as an assertion or as a set of attributes. Attribute named `event-type` indicates its kind of event. The event may also include `source` and `destination` attributes to indicate respectively which agent (or group) the event is originated and which one it is sent to. For example, `{event-type:"domain-input", destination:["agent01"], event:{ weather:"cloudy", temperature: 27}}` is an event indicating an update in the domain environment sent to the agent (`agent01`) about the current weather condition (`weather:"cloudy"`) and the temperature (`temperature:27`). The example does not include the source attribute since it is assumed that the event is coming from the environment as implied by its type of attribute.

Instructor Intervention and Disruption

As shown in Figure 2(ii), the agent *façade* continually generate options regarding its own states and the other agents' reflecting Algorithm 1. When the user instructs the agent to

do some actions, the user intervenes the running deliberation and the control process. The interventions may disrupt the on-going process of deliberation and execution, especially when they constitute changes to mental data structure (e.g inserting, changing, removing items in $\mathcal{B}$, $\mathcal{D}$, $\mathcal{I}$, Π, or events). Due to the incomplete or erroneous message in the intervention, the instruction may not be correctly interpreted or false response can be generated to a disruption. For any event received from or about another agent, there are several possibilities regarding the meaning or intention in the message as follows:

1) The instructor just instructs the agent to do something without requiring it to learn anything. It can be a wrong or false instruction made by a mistake. It may also be made just for immediate testing or something irrelevant to learning.
2) The instructed action should be learnt but the whole presentation is due to complete since it may be a part of a long presentation sequence and the current instruction may still be on-going.
3) The instructed action is made as a response to a former particular condition or situation.
4) The instructed action updates or corrects a currently on-going actions or intentions.
5) The observed or informed action from another agent (or the instructor) is received as the action to follow or imitate.

The façade must detect a disruption when the user gives an instruction to the agent. The kind of disruption may range from simply an interruption that may halt an on-going planned execution temporarily to an intention that potentially keeps the agent from achieving another existing one. Once a disruption is detected, the agent may ask the instructor for confirmation or clarification while indicating the potential implications.

Besides user interventions, disruptions can be sourced from the internal deliberation cycles like failures in searching for the applicable schema or in executing actions. Different types of disruptions can be defined as external or internal disruptions as follows:

1) External disruption is coming from an external source of the agent like the instructor message, domain environment, or other agents communication that modifies (insert, remove, or update) $\mathcal{B}$, $\mathcal{D}'$, $\mathcal{D}$, $\mathcal{I}$, $\mathcal{I}'$, or Π. Different types of disruption event include *mutual exclusive* (`mx`), *contradictory* (`contra`), *insertion* (`insert`), and *similarity* (`similar`) indicating conflicting actions, contradictory assertions, insertion of a new assertion P_i ($\mathcal{B}^{(old)} \tilde{\Rightarrow}_1 \mathcal{B} \cup P_i$), and a similarity (or a difference) between a new incoming assertion with an existing one in the corresponding intention or schema respectively.
2) Internal disruption is coming from the internal process of deliberation cycle. Different types of internal disruption include *failed planning* and *failed execution* indicating that no matching applicable plan can be found for desire assertions $\mathcal{D}^*$ that $\mathcal{D}^* \subset \mathcal{D}$ and there are failed intentions $\mathcal{I}^-$ that $\mathcal{I}^- \subset \mathcal{I}$ respectively.

A disruption is indicated as a `disrupt` type event. The disruption message contains not just the main incoming event description (`event` attribute) but may also include the current disrupted or affected assertion. For example, `{event-type:"disrupt", distype:"contra", event:{event-type:"instruct", event:{do:"insert", belief:{ weather:"sunny"}}}, affected:{belief:{weather:"rainy"}}` indicates a possible contradiction when the agent is instructed to make belief that the current weather is `sunny` while currently it believes that it is `rainy` instead.

Algorithm 2: Interactive Learning Cycle

WHILE True

FOR every external disruption event $\varepsilon \in \mathcal{E} \cup \mathcal{E}_A$
  inform the instructor about the disruption, and ask for a response
  WAIT until the instructor responds with event ε_i
  IF ε_i is an instruction to perform an action
    ask what cause the choice of the instructor action
    inform the instructor possible causes based on similar rule schemas Π^*
    WAIT until the instructor responds with event ε_i'
    IF ε_i' indicates a selection of $\pi^* \in \Pi^*$
      generalize $\texttt{actions}(\pi^*) \leftarrow \text{Gen}(\texttt{actions}(\pi^*), \texttt{actions}(\varepsilon_i))$
    ELSE IF ε_i' indicates an assertion P_x^π as the cause of action in ε_i
      generate rule π' with P_x^π as the trigger, and $\texttt{actions}(\varepsilon_i))$ as the actions
      insert the new schema $\Pi^{new} \leftarrow \pi' \cup \Pi$
    ELSE IF the instructor indicates the actions are not finished yet
      append the action to the sequence as $\texttt{actions}(\varepsilon_i))$
    ELSE BREAK

FOR every internal disruption event $\varepsilon^* \in \mathcal{E} \cup \mathcal{E}_A$
  IF ε^* indicates a failed planning for $\mathcal{D}^*$
    ask the instructor to show how to achieve the goal in ε^* by actions π_p^*
    REPEAT
      indicate all applicable $\pi^{*'} \in \Pi$, in which $\texttt{goal}(\varepsilon^*) \tilde{\Rightarrow}_\rho \texttt{goal}(\pi^{*'})$
      WAIT until the instructor responds with event ε_i^*
      IF ε_i^* is instruction to do action P_α
        append P_α to π_p^*
      IF ε_i^* indicates a selection to $\pi^{*'}$ as a plan to achieve $\texttt{goal}(\varepsilon^*)$
        generalize $\texttt{goal}(\pi^*) \leftarrow \text{Gen}(\texttt{goal}(\pi^*), \texttt{goal}(\varepsilon^*))$; **BREAK**
      IF ε_i^* indicates a selection to $\pi^{*'}$ as an intermediate goal for $\texttt{goal}(\varepsilon^*)$
        append $\texttt{goal}(\pi^{*'})$ to π_p^*
      IF ε_i^* is instruction to follow or imitate agent i
        append observed action P_α^i of agent i as its own action to π_p^*
    UNTIL $\texttt{goal}(\varepsilon^*)$ is achieved or the instruction is dropped/canceled
    IF $\texttt{goal}(\varepsilon^*)$ is achieved
      IF $\texttt{goal}(\varepsilon^*)$ is achieved by another agent j (j can be the instructor)
        append asking j to achieve $\texttt{goal}(\varepsilon^*)$ as a step in π_p^*
      set $\texttt{goal}(\pi_p^*) \leftarrow \texttt{goal}(\varepsilon^*)$; $\Pi^{(new)} \leftarrow \pi_p^* \cup \Pi$; **BREAK**

  IF ε^* indicates a failed execution of planned actions in $\mathcal{I}^-$
    ask what is the cause that make the plan in ε^* failed
    inform the instructor possible preconditions based on similar plan schemas
    WAIT until the instructor responds with event ε_i^-
    IF ε_i^- indicates a condition that the plan in ε^* can be applicable
      modify the plan schema in ε^* with the condition in ε_i^- as the precondition

---

Learning Schemas

Learning can be initiated by inserting a new intention to obtain information for generating schemas so that the disruption may be resolved and the computational burden to obtain and resolve the issues may be reduced in a similar situation in the future. Since the mechanism in the BDI architecture allows several intentions or goals to be active in parallel, the learning intention enables learning to be conducted concurrently with the main deliberation cycle. Consequently, the teaching process can be continuous and interactive as well while the domain specific tasks are conducted simultaneously.

Some domain independent rules can be constructed to learn rule-based schemas, but some other may be designed to acquire plan-based schema on-the-fly. As a schema, a learning rule can be made to trigger a sequence of actions allowing it to engage a long dialog or conversation with the user.

To simplify the matter, the overall learning rules are described as abstract execution cycles in Algorithm 2. In the algorithm, the learning process is considered as interactive processes between the learner and the instructor. When the event indicating disruptions or unanticipated conditions is received, the agent informs the potential issue to the instructor and asks for more advices to guide the resolution. Based on the instructor feedback, new rules or procedures can be learnt. A rule-based schema can be learnt by asking the instructor about what cause the action chosen by the instructor to resolve the disruption (as shown in Algorithm 2 line 3 to 16). On the other hand, when no matching schema can be found for a particular selected desire (as a condition initiating the disruption event), the instructor can teach the agent the series of actions to perform to achieve it (Algorithm 2 line 18 to 35). Similarly, when a plan execution fails, the instructor may respond by informing the agent the condition that should be held to prevent the failure (Algorithm 2 line 36 to 41) so that it can be used as the precondition attribute of the learnt plan-based schema.

Learning a schema may include generalization of attributes. In Algorithm 2, a selection of a similar schema as informed by the agent will be followed by generalization of the corresponding schema. For example, generalization of actions (line 10) is conducted after the selection (line 9). Another generalization is on the goal of a plan-based schema (line 26).

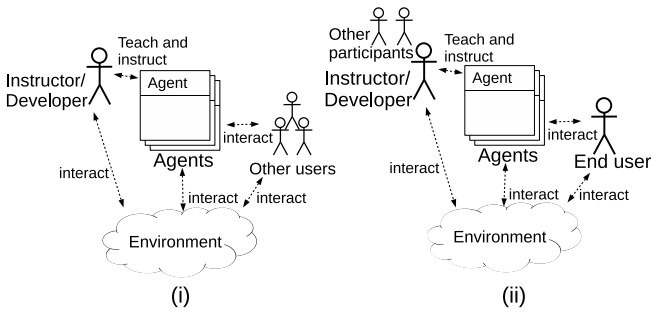

Fig. 3. (i) teaching agents to interact with multiple users; (ii) collaborative teaching by instructor and other participants.

When asked for advices, the instructor may also perform something or letting another agent to achieve the goal. In this case, the agent may socially learn by adding a step of action to ask for help from another more capable agent (or from the instructor) in the schema. As shown in Algorithm 2, when the goal is observed to be achieved by another agent (line 33), asking the other agent to achieve the goal (line 34) will be learnt as an action step in the schema. Learning the action steps to take to achieve the goal may also be based on the imitation or copying another agent's actions.

Algorithm 2 (line 29 to 30) shows that the agent may be instructed to imitate the actions of another agent rather than generating the action by itself. However, the effectiveness of this imitative action depends highly on the specific implementation of the domain level component. The capability of observing and recognizing the actions and intentions of another agent or component is assumed to be domain specific. However, the capability to recognize the activity from other agents may still be learnt at the façade level through user instructions.

By associating an agent's identity with an attribute of an assertion, an agent can learn to associate its tasks with another's. This also enables the teaching-learning process to conduct in its social context that other parties may also interact with. As exemplified further in the next section, it allows the instructor to teach the agent on-the-run to interact with or to serve another user. It is worth noting that since the learning cycle as shown in Algorithm 2 is realized as intentions, the actual process is conducted concurrently with the main BDI cycle and other domain level processes.

IV. Scenarios and Case Study

As proofs-of-concept, the teachable framework is applied to two different domains of application. The first domain is the teachable human-like NPCs (Non-Player Characters) in NTU Co-Space wherein the framework is exemplified as an approach of developing autonomous NPCs that roam and interact with user avatars in a custom-built virtual reality environment. It is to exemplify the interactive incremental process of learning that the component (as an agent) acquires new rules and plan-based schemas from the user instructions. The second domain is the aging-in-place simulation in which several agents act as in-house caregivers by providing advices and recommendations to the user in daily living. This second domain demonstrates the acquisition of rules or knowledge to deal with exceptions. The framework is exemplified based on scenarios that show a domain expert teaching the agents to collectively and cooperatively advise and persuade the occupant towards well-being and healthier lifestyle.

A. NTU Co-Space

NTU Co-Space is a multi-user online virtual environment representing a real actual place in the world including events happening in it (in this case the NTU campus) [9]. The user may explore the environment by controlling a human-like avatar to roam the environment and perform certain tasks. Inside the virtual environment, the user may meet and interact with other users' avatars. However, it is also possible to encounter a number of (semi-) autonomous human-like NPC (Non-Player Character) each with its own personality, agendas, and tasks to perform [10].

To develop autonomous behaviors of the NPCs, the teachable framework can be applied so that rules and procedural schemas can be instructed at runtime. In the NTU Co-Space, the instructor can take over the control of the NPCs movements and actions. The instructor can also select any object in the virtual environment (e.g by mouse pointing and clicking) to obtain the options of what can be done with or to the object

for selection. For example, when the instructor makes the NPC select an object (e.g coin) in the enviroment, the user interface shows the options of possible actions that can be applied to it. The instructor can also observe the agent's mental state and may enter a command directly in a command line interface while performing some actions in the environment. Figure 4(i) shows the user interface in NTU Co-Space for monitoring and control of the agent. When an object is selected, a dialog is opened showing the options like in Figure 4(ii). Similarly, when another agent (or another user) is selected, options will be displayed (Figure 4(iii)) but with more elaborated items to select.

NTU Co-Space is implemented using Unity 3D game engine[2]. The parallelism aspect in the framework is made possible by the inherent feature of Unity 3D framework in which different scripting modules and game assets can have independent execution cycles.

In what follows, scenarios and illustrative examples in the use of the framework in NTU Co-Space will be explained from simple navigational tasks to complex interaction and collaboration.

Simple Navigation Task

Scenario 1: Simple Navigation Task

Scene 1:	The instructor selects a desired location for the agent (the user points and mouse-clicks a location in the environment to set it as a desired condition of the agent)
Scene 2:	`Agent-1` selects the desire, finds a schema to achieve it, and execute the intentions
Scene 3:	`Agent-1` moves towards the goal location (based on the intention selected)
Scene 4:	the instructor intervenes the journey by turning the agent's orientation following the side of an obstacle
Scene 5:	A disruption event fires, `Agent-1` informs the user: "achieving at desired location is disrupted by `following-object-side`."
Scene 6:	The instructor continues following the pole side
Scene 7:	`Agent-1` asks the instructor "why did you instruct me to `following-object-side`?"
Scene 8:	The instructor selects the obstacle object `building-part` so that the agent perceives it and associates it with the reason of `following-object-side` action (disruption)
Scene 9:	Generate rule schema and learn {`trigger-cond`: {`object-type`: `"building-part"`,...}, `actions`: {`do`: `"following-object-side"`,...}
.....	.....
Scene 10:	`Agent-1` has a desire to be at `frontdesk` and an intention to move towards it
Scene 11:	`Agent-1` moves towards `frontdesk` as the goal location
Scene 12:	In the middle of the path texttt`Agent-1` updates its belief that {`object-type`: `"building-part"`,...} is detected
Scene 13:	learnt schema with {`object-type`:`"building-part"`,...} fires and `Agent-1` executes `"following-object-side"` to move around the obstacle before continuing the move towards the goal

The first scenario (Scenario 1) shows a simple use of the teachable framework to develop a rule-based schema. This scenario illustrates how a new rule-based behavior schema can be acquired through a combination of interaction and instruction. In particular, the user wants to make the NPC develops a schema for avoiding obstacles supporting the existing navigation strategy to move from one place to another. Moving in a straight line towards the destination may fail since there can be objects or obstacles along the way potentially blocking the straight path (Figure 5). The agent needs a mechanism to avoid the obstacles.

To teach the agent the avoidance strategy, the instructor can take over the control intervening the on-going execution (Scene 4). The intervention creates disruption that initiates the agent's attention to the actions taken over by the instructor (detected as `following-object-side` by the agent in Scene 5). As a part of the interactive learning strategy, the agent asks the reason of that particular intervention to the user (Scene 7). Based on the user response, a new rule (schema) can generated as in Scene 9. Based on the learnt schema, the agent will execute the same action `following-object-side` the next time it encounters `building-part` to avoid the obstacle (Scene 12-13).

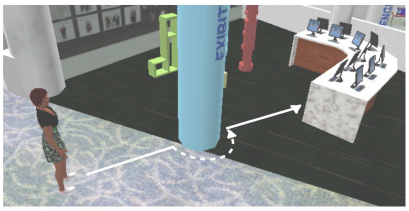

Fig. 5. Teaching simple navigation task scenario in NTU CoSpace.

Interactive Planning

The second case (Scenario 2) demonstrates the teachable feature for developing inter-agent activities beyond learning the internal schema. In this scenario, a group of two NPCs is taught to interact and serve an approaching player, as an avatar (left-hand side of Figure 6a). Both NPCs have different capabilities and particular knowledge to interact with the player.

Scenario 2: Interactive Planning

Scene 1:	`Player1` approaches `Agent1` and `Agent2`
Scene 2:	`Player1` asks : "I want to go to the thesis section in this library?"
Scene 3:	`Agent1` replies `Player1` request : "You may go there", and insert the intention to make `Player1` at location `thesis-section`
Scene 4:	An internal disruption event of `Agent1` fires, `Agent1` fails to find a matching schema for the desire
Scene 5:	`Agent1` asks the instructor. "How can I make `Player1` at location `thesis-section` ?"
Scene 6:	The instructor asks `Agent2` to make `Player1` at location `thesis-section`
Scene 7:	`Agent2` sets the intention to make `Player1` at location `thesis-section`, finds a matching schema, and initiates the intention execution
Scene 8:	`Agent2` asks `Player1`: "Follow me!", and move to the intended location while Player1 follows
Scene 9:	Eventually `Agent2` and `Player1` arrive at location `thesis-section`
Scene 10:	`Agent1` receives the event that `Player1` is at location `thesis-section` indicating the desire has been achieved
Scene 11:	`Agent1` appends the action to ask `Agent2` to make `Player1` at location `thesis-section` as the step of action to in the schema to achieve the same goal

The scenario demonstrates group reliance in planning. Here, it is assumed that only `Agent1` as a virtual librarian NPC can interpret the user inquiry (Figure 6) as shown in Scene 2 and 3. It adopts the predicted goal of the user to be at the particular

[2]http://unity3d.com/

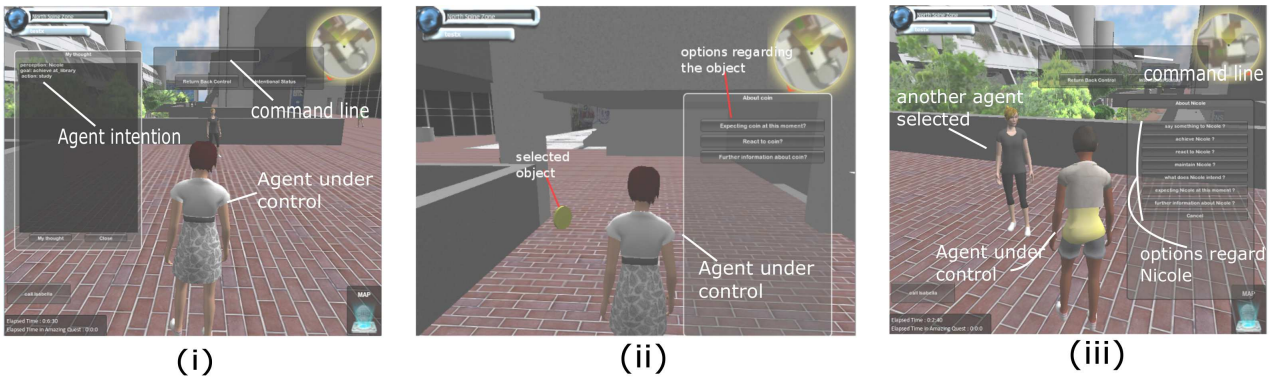

Fig. 4. (i) Third-person control view of the virtual character by the user. Information about mental state of the agent can continuously be monitored and a command can be initiated at runtime. (ii) Options menu showing what actions can be applied regarding the selected object (coin object) (iii) Options menu showing what actions can be applied regarding the selected agent (Nicole). The selection can be made through the menu or typing the command in command line.

location (`thesis-section`) as shown in Scene 3. Since no schema is available to directly achieve the player goal, `Agent1` asks the instructor for possible actions to achieve it. However, the instructor responds the inquiry by asking `Agent2`, instead, to realize the goal (Scene 6). In this case, `Agent1` is taught by the instructor to ask for help from another agent in the group. In this case, `Agent2` is asked by `Agent1` to respond according to the user needs (Scene 9-11). This feature of learning delegating actions is the part of the learning mechanism as shown in Algorithm 2 line 32 to 33.

The scenario above demonstrates the learning to delegate some tasks to another agent. This approach of learning can also be useful when the system includes multiple users letting the agents learn directly from the real context of the system use. In this second scenario, however, only a single agent learns even though the schema learnt involves another agent.

Fig. 6. Teaching serving player avatar through interactive planning.

Group Learning

When several agents are grouped together, they may behave like a single entity but with more complex goals and contexts than any individual. When the user asks a group of agent to achieve a goal, the outcomes may likely be very different when the same request is given to a single agent. For example, the desire of `Agent1` to be at location `West_Lobby` may only make `Agent1` move to the location (if the right schema is available) or ask the instructor for help. The instructor can just control the agent to navigate to the location as described in Algorithm 2 line 21 to 34. However, if the desire is given to the group so that, for example, `Agent1`, `Agent2`, `Agent3`, `Agent4`, and `Agent5` altogether must be at location `West_Lobby`, each member would try to make every other member to be at the same place besides itself. Some members in the group can be instructed to pursue their actions (`Agent1` and `Agent5` in Scene 4) but some others may be made to imitate others letting them learn from each other (Scene 6). In the end, all members learn from each other (Scene 9).

Scenario 3: Group Learning

Scene 1: The instructor set the group (consisting of Agent1, Agent2, Agent3, Agent4, and Agent5) to have desire to be at location `West_Lobby`
Scene 2: Internal disruptions fire from all members of the group since no exact matching schema can be found to achieve the goal; every member informs the user about the disruption
Scene 3: `Agent1` and `Agent5` indicate plan schemas that can bring (only) oneself to be at `West_Lobby` but not together with the other members
Scene 4: The instructor select the partially match schemas and let `Agent1` and `Agent5` execute them
Scene 5: `Agent1` and `Agent5` execute the schemas and move to the location `West_Lobby`
Scene 6: The instructor instruct `Agent2`, `Agent3`, and `Agent4` to follow (imitate) `Agent1` and `Agent5`
Scene 7: All members arrive at the destination location `West_Lobby`
Scene 8: `Agent1` and `Agent5` generalize their goal schemas to include all members of the group
Scene 9: Other members learn the schemas to be at `West_Lobby` together

Fig. 7. Teaching collaboration among members of the group.

This scenario demonstrates the imitative learning of agents in the group. However, in this domain, it is assumed that all teachable agents are implemented in the same way and have

the same capability to recognize other agents activities.

The scenarios from the NTU Co-Space case study above suggest that teaching can be conducted in both level of internal process within a single agent and at a group level to learn interdependencies among different agents in the group. The user may start with some sketches of initial grouping of components which can either be internal activity states or entire agents. However, the internal ones may anytime be created, removed, or replaced by the execution mechanism of the agent architecture. In the level of individual agents, the components or modules are persistent and may instantiate for the entire system lifetime.

One important feature of the framework as shown in the scenarios above is teaching the agents to serve or interact with the target end users directly at runtime in the situated environment as illustrated in Figure 3(i). When the deployed system will be used by or interacting with different users, this model of agents as trainees enables the learning by instruction to be conducted in the proper context.

B. Teachable Caregiving Agents for Ageing-in-Place

We also apply the framework for developing coordination mechanisms of multiple assistive agents in a simulated smart-home environment in the domain of in-house caregiving for elderly (Aging-In-Place) [17]. The virtual home environment is built with a virtual elder occupant residing as the subject. The virtual occupant is initially built to follow a routine habit of daily task. Four computational agents are included in this case. Two agents collect data from the virtual environment and produce beliefs about the subject's activity of daily living. The other two are health-caring agent and butler agent that concern the subject's health (physical and mental) and quality of daily life (happiness, emotion, and social) respectively.

In the simulation, the artificial occupant is made as an avatar in which its behavior can be controlled manually by the user or can follow a programmed script. The implementation of the occupant's avatar is similar to the NPCs in NTU Co-Space. However, it also consists of an assembly of virtual assistant agents each is designed to serve the user for different aspects of caregiving. Different agents may take particular roles ranging from updating perceptual information to persuading the user. An agent by itself is fully functional as an application which can be deployed independently. Two different persuasive agents have been implemented. *Smart butler* is a virtual assistant resided in a mobile device (smartphone or tablet) that focuses on daily activities, social connections, and sustainable living. It mostly takes the role as a reminder and recommender in relation to daily living and social events. On the other hand, *Virtual nurse* is a virtual assistant running in a dedicated computing device or personal computer displayed with human-like appearance. It can communicate naturally with the user using voice-based interaction. The main role of the nurse is a healthcare advisor that gives advice and persuades the user regarding choices of healthier lifestyle.

Figure 8 shows the multi-agent system for aging-in-place wherein heterogeneous smart applications interact with each other as assistant agents through the shared repositories. The agents can be heterogeneous in which each one may have particular specialties and capabilities. Each agent can work independently to address a particular issue by monitoring and interacting naturally with elderly. The domain also allows collaborative instructions involving other parties of human participants (Figure 3(ii)).

The teachable model is embedded into two of the agents above that act as advisors. The domain expert can control the agents in advising the occupant. Figure 8(ii) shows the screenshot of the simulation with 3D virtual environment GUI in a web browser (implemented with Unity 3D and HTML5). Figure 8(ii) shows that the instructor can observe the options, predictions, and scheduled intentions on the activity timeline display. The domain expert can also interact directly with the agents as an instructor using a natural language interface. In the dialog, the instructor may indicate the selection of an agent belief or intention by choosing the activity bar in the timeline. The corresponding activity can be suppressed or modified on the fly.

Here are example scenarios excerpted from the interaction between the domain expert as the instructor and the advisor agents in providing the appropriate advices to the virtual occupant.

Rule Generalization

Scenario 4: Rule Generalization

Scene 1:	`virtualnurse` advises the occupant to take a medicine: "Hi, it's time for you to take the medicine".
Scene 2:	`smartbutler` believes: the occupant is less active.
Scene 3:	`smartbutler` advises the occupant: "It's a good time to get some fresh air. Going out perhaps?".
Scene 4:	The instructor selects the recent `smartbutler` advising action (by selecting it in the timeline) and stops its operation.
Scene 5:	`smartbutler` receives a disruption event about its the stopping operation.
Scene 6:	`smartbutler` asks the instructor: "why did you stop my dialog asking: 'It's a good time to get some fresh air. Going out perhaps?' to the occupant?".
Scene 7:	`smartbutler` indicates a similar rule that previously a dialog 'it's a good time to exercise now' is stopped because `virtualnurse` reminded the occupant to take the medicine.
Scene 8:	The instructor selects the similar rule as indicated so that `smartbutler` generalizes the rule and the association becomes stopping advising any activity whenever `virtualnurse` reminds her to take the medicine.
Scene 9:	`smartbutler` asks the instructor: "Do you mean IF I detect that Virtual Nurse advise the occupant to take medicine, THEN I drop my intention to recommend activity? (Yes/No)".
Scene 10:	The instructor responds by answering 'Yes' and `smartbutler` learn by generalizing the rule to suppress its recommendation.

This scenario demonstrates the generalization of actions description based on a previously learnt rule in the context of conflict detection and resolution. In Scenario 4, an advice provided by smart butler is conflicting with the virtual nurse's. While virtual nurse is intending to let the occupant immediately take the medicine, smart butler suggests the occupant to go out instead (Scene 3) which likely may cancel out the intended effect of taking the medicine on time (Scene 1).

When the instructor stops the advising intention of smart butler (Scene 5), a disruption event is generated leading smart butler to question the reason of intervention to the instructor

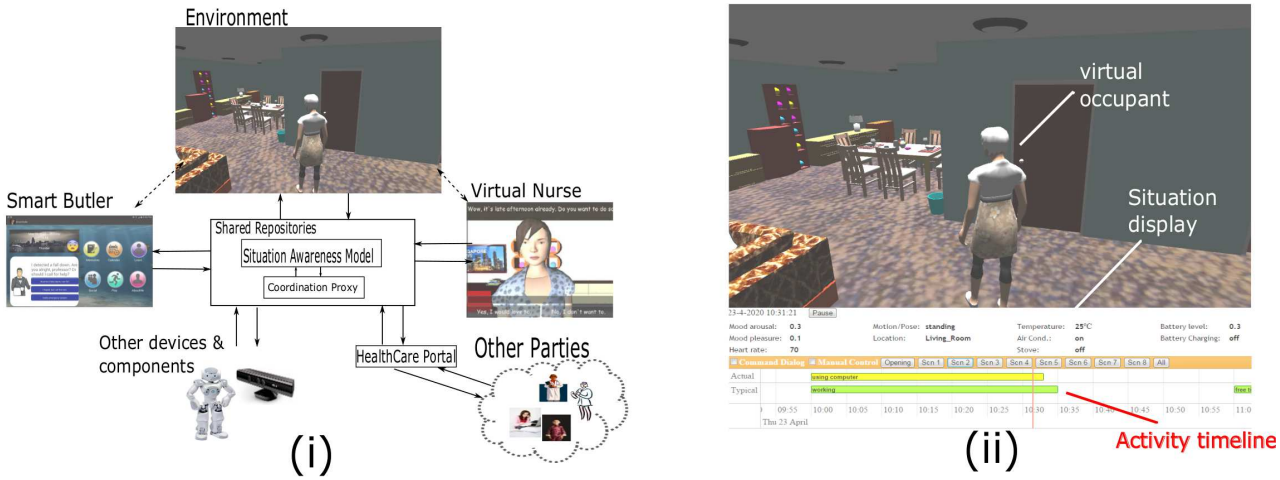

Fig. 8. (i) Architecture of Multi-agent Smarthome system for Ageing-in-Place; (ii) GUI of simulation platform for ageing-in-place

(Scene 6). In the scenario, smart butler has previously learnt a similar rule which is also resulted when the instructor stopped its operation. As a general strategy for learning, smart butler indicates this similarity to the instructor (Scene 7).

The conflict can be resolved by letting smart butler to generalize its previously learnt rule on stopping the provision of a specific advice given the same condition of the virtual nurse activity (Scene 9). In this case, smart butler always suspend or stop the advice provision to the occupant whenever virtual nurse reminds the occupant to take the medicine.

Recognizing False Alarm

Scenario 5: Recognizing False Alarm

Scene 1: The occupant walks to the living room and lying down on the sofa.
Scene 2: `smartbutler` believes: occupant falls (agent detects lying pose as fall condition); `virtualnurse` believes: occupant is on the sofa in the living room (the agent can locate position more precisely in the living room).
Scene 3: `smartbutler` opens a dialog with the occupant: "are you okay sir? are you hurt?".
Scene 4: The instructor selects the recent `smartbutler` action (from the timeline) and stops its operation so that the disruption event is received by `smartbutler`.
Scene 5: Smart Butler ask the instructor: "why did you stop my dialog asking: 'are you okay sir? are you hurt?' to the occupant?".
Scene 6: The instructor selects the belief of `smartbutler` (in the shared timeline) that the occupant fell and remove it from the timeline (which is again signaling a disruptive event to `smartbutler`).
Scene 7: Smart Butler asks the instructor: "why did you remove my belief that the occupant falls?".
Scene 8: The instructor selects the belief of `smartbutler` (in the shared timeline) that the occupant is on the sofa in the living room to indicate the cause.
Scene 9: `smartbutler` associates the condition that `virtualnurse` believes that occupant is on the sofa in the living room with the removal of belief that the occupant falls".
Scene 10: `smartbutler` asks the instructor: "Do you mean IF I detect that `smartbutler` believes that the occupant is on the Sofa in the living room, THEN I remove my belief that the occupant falls? (Yes/No)".
Scene 11: The instructor responds by answering 'Yes' and `smartbutler` learn the new rule to update its belief.

In this scenario, the smart butler can be taught to suppress its own belief (detecting fall based on the lying pose) based on the other agent's belief (Virtual Nurse) of the occupant's location. In this case, the functionality of the domain module can be extended by incorporating beliefs or perception of another agent in the domain.

The smart butler agent has a limited capability to identify some critical conditions of the occupant. As shown in Scene 2, it concludes a fall condition only based on the occupant's pose. It may generate a false warning and potentially a false response like in Scene 3.

The user can teach smart butler to suspend or cancel the response like in Scene 4 which triggers a disruption event and initiates the agent's inquiry about the reason (Scene 5). Since no more information from the sensors can be derived by smart butler, the condition for cancelation can be the other agent's action. As shown in Scene 9, smart butler put the identification of the occupant's location by virtual nurse as a condition to cancel the default response of a critical warning.

Cooperative Persuasion Dialog

In this scenario, the instructor can teach more than one agents to cooperate in persuasion tasks. Smart butler may ask for help to the instructor whenever it fails to achieve the goal by itself. In Scene 3 Scenario 6, smartbutler asks the instructor for help since there is no more schema for persuading the occupant when the occupant is not interested. A cooperative behavior can be taught by instructing virtual nurse (Scene 4) to help smart butler by persuading the occupant (Scene 5) so that virtual nurse may learn to start to help the smart butler when it is asked for (Scene 8).

When virtual nurse successfully persuade the occupant, smart butler may detect the condition (Scene 10) and ask the instructor for confirmation to learn a new action to ask virtual nurse for help (Scene 11).

The rule learnt by virtual nurse above is still specific to the condition and intention of smart butler in the particular scenario. However, when a similar situation happens (e.g the occupant loses her interest to eat healthy food), the same rule may be put as an option so that virtual nurse can learn to always help smart butler whenever it asks for.

All scenarios on Ageing-In-Place domain show that relationships and connectivity among the agents enable incremental and scalable learning by a single component. Many agents can be taught together not just to do domain tasks independently but also to coordinate with each other. The agents can be taught together as a team. In a similar setting, more than one instructors may also be involved to teach the agents based on the same principles. Each instructor may take the control of a different agent while teaching it and collaborate with the other instructors.

Scenario 6: Cooperative Persuasion Dialog

Scene 1:	`smartbutler` advises the occupant to do exercise as it is detected that the occupant has been sitting down too long: "It's a good time to do a little stretching".
Scene 2:	The occupant does not follow what is advised and still sitting for long.
Scene 3:	`smartbutler` can not find another plan to persuade the occupant and asks others for help to make the occupant intend to do the workout.
Scene 4:	The instructor instructs the `virtual nurse` to intend that the occupant do the exercise.
Scene 5:	`virtual nurse` knows how to motivate the occupant (has the schema) and starts persuading her: "do you know, working out for few minutes can reduce the effect of...".
Scene 6:	`virtual nurse` receives disruptive event (instruction to insert an intention) and asks the instructor "why did you instruct me to make the occupant intend to do the exercise?".
Scene 7:	The instructor selects the recent request for help by `smartbutler`.
Scene 8:	`virtual nurse` asks the instructor: "Do you mean IF I detect that`smartbutler` ask for help to achieves that the occupant intend to do exercise, THEN I intend that the occupant intends to exercise? (Yes/No)"
Scene 9:	The instructor responds 'Yes' and `virtual nurse` learns the rule.
Scene 10:	`smartbutler` detects that `virtual nurse` has persuaded the occupant and the intention is achieved.
Scene 11:	`smartbutler` asks the instructor: "The schema to achieve that the occupant intends to exercise includes asks `virtual nurse` to help to achieve the occupant intends to exercise? (Yes/No)".
Scene 12:	The instructor responds 'Yes' and `smartbutler` learns the plan.

V. Conclusion

We have presented a framework for developing situated interactive systems wherein the components of the system can be considered as autonomous agents that can be configured and teachable by the user. The framework offers a methodology of developing a system allowing end users or non-technical experts to teach the components at runtime. Each teachable component is an encapsulated set of domain level attributes and actions but possessing generic mental attributes like beliefs, desires, and intentions. In every step of its execution cycle, the teachable agent may indicate possible operations, goals, and intentions letting the instructor to decide if some intervention is necessary. During the learning, the agent may also indicate possible rules or procedural schemas that can be learnt based on similar knowledge that are pre-existing or learnt in the past. The instructor can teach not just the domain level operations but also ways to reason, to make decision, and to relate with other agents (or other human participants).

We have exemplified the interactive learning processes in our custom-built virtual characters in NTU Co-Space environment. The scenarios demonstrate how learning incrementally acquire knowledge and how imperfect or incomplete instruction can be dealt with similarity matching and interactivity. The framework has been applied to a virtual aging-in-place environment wherein a virtual occupant and some persuasive agents are teachable. The case study shows that the proposed framework allows a caregiver, as the domain expert, to integrate diverse intelligent capabilities through teaching and interactions. The scenarios also show that this instructional-based learning can be applied beyond a single agent but can be used to teach a couple or even a group of agents to coordinate and perform some tasks together.

There are many outstanding aspects and issues deserve further exploration and study. More comprehensive investigation including user evaluation is necessary to get more complete picture of the effectiveness of this approach for developing a large scale system or application.

References

[1] J. R. Anderson and C. Lebiere. *The Atomic Component of thought*. Lawrence Erlbaum Associates, Mahwah, 1998.

[2] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano, editor, *Handbook of Robotics*, chapter 59, pages 1371–1394. Springer, 2008.

[3] M. Blokzijl-Zanker. Multi robot learning by demonstration (extended abstract). In *Proceedings of the eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1207-1208, 2012.

[4] C. Breazel, M. Berlin, A. Brooks, J. Gray, and A. L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems*, 54:385–393, 2006.

[5] M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *Proceedings of the Seventh annual ACM/IEEE international conference on Human-Robot Interaction (HRI'12)*, pages 17–24, 2012.

[6] S. Chernova and M. Veloso. Confidence-based multi-robot learning from demonstration. *International Journal of Social Robotics*, 2(2):195–215, 2010.

[7] A. Chyper, M. Dontcheva, T. Lau, and J. Nichols, editors. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann, Amsterdam, 2010.

[8] A. G. Hernandez, A. E. Segrouchini, and H. Soldano. BDI multiagent learning based on first-order induction of logical decision trees. In S. O. N. Zhong, J. Liu and J. Bradshaw, editors, *Intelligent Agent Technology: Research and Development*. World Scientific, New Jersey, pages 160–169, 2001.

[9] Y. Kang, B. Subagdja, A.-H. Tan, Y.-S. Ong, and C.-Y. Miao. Virtual Characters in Agent-Augmented Co-Space (Demonstration). In *Proceedings of the eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 1465-1466, 2012.

[10] Y. Kang, A.-H. Tan, F.-H. Nah. Agent-based Virtual Humans in Co-Space: An Evaluative Study. In *Proceedings of 2012 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2012)*, pages 59–66, 2012.

[11] J. E. Laird. *The SOAR Cognitive Architecture*. MIT Press, Cambridge, 2012.

[12] H. Lieberman, editor. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, 2001.

[13] V. K. Mavromichalis and G. Vouros. Building intelligent collaborative interface agents with the ICagent development framework. *Autonomous Agents and Multiagent Systems*, 13:155–195, 2006.

[14] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the first International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[15] T. Selker. COACH: A teaching agent that learns. *Communications of The ACM*, 37(7):92–99, 1994.

[16] B. Subagdja, L. Sonenberg, and I. Rahwan. Intentional learning agent architecture. *Autonomous Agents and Multi-Agent Systems*, 18(3):417–470, 2009.

[17] D. Wang, B. Subagdja, Y. Kang, A.-H. Tan, and D. Zhang. Towards intelligent caring agents for aging-in-place: Issues and challenges. In *Proceedings of 2014 IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI 2014)*, pages 102–109, 2014.

[18] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, Chichester, 2009.