Synthesizing Daily Life Logs through Gaming and Simulation

Mario Caruso
Sapienza Univ. Roma
DIAG, via Ariosto 25
Roma, 00185 Italy
caruso@diag.uniroma1.it

Massimo Mecella
Sapienza Univ. Roma
DIAG, via Ariosto 25
Roma, 00185 Italy
mecella@diag.uniroma1.it

Çağrı İlban
Sapienza Univ. Roma
I3S, via Ariosto 25
Roma, 00185 Italy
ilban.1375845
@studenti.uniroma1.it

Stavros Vassos
Sapienza Univ. Roma
DIAG, via Ariosto 25
Roma, 00185 Italy
vassos@diag.uniroma1.it

Francesco Leotta
Sapienza Univ. Roma
DIAG, via Ariosto 25
Roma, 00185 Italy
leotta@diag.uniroma1.it

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

http://dx.doi.org/10.1145/2494091.2495977

Abstract
In the recent years there has been a growing interest in the design and implementation of smart homes, and smart buildings in general. The evaluation of approaches in this area typically requires massive datasets of measurements from deployed sensors in real prototypes. While a few datasets obtained by real smart homes are freely available, they are not sufficient for comparing different approaches and techniques in a variety of configurations. In this work, we propose a smart home dataset generation strategy based on a simulated environment populated with virtual autonomous agents, sensors and devices which allow to customize and reproduce a smart space using a series of useful parameters. The simulation is based on declarative process models for modeling habits performed by agents, an action theory for realizing low-level atomic actions, and a 3D virtual execution environment. We show how different configurations generate a variety of sensory logs that can be used as input to a state-of-the-art activity recognition technique in order to evaluate its performance under parametrized scenarios, as well as provide guidelines for actually building real smart homes.

Author Keywords
simulation environment for smart spaces, daily life logs, planning, gaming

Introduction

In the last years there has been a blooming interest in *smart spaces* both from researchers and practitioners. A smart space is a physical space (e.g., a residential house, an office building) in which inhabitants and pervasively spread-out devices implicitly interact in order to provide continuous sensor information and react to service requests from users, according to the vision of the *Internet-of-Things* (IoT, [2]).

The correct design and deployment of such spaces (from a practitioner's point of view), as well as the definition of novel architectures, techniques and methods (from a reseachers's point of view) require extensive experimental facilities. Some freely-available datasets are published in the research community (cf. CASAS[1] and MIT House_n[2] projects); smart facilities are planned to be built in the nearby future (cf. EU initiatives for Living Labs[3], US NIST initiative for pervasive computing [15], etc.). Unfortunately, these installations and testbeds only contain limited types and number of sensors that are not sufficient to validate algorithms for smart spaces in a thorough way; without customizable datasets available, it is hard to design a smart space, in such a way that maximizes employed algorithms performance. This led us to realize the need for a configurable dataset generation tool that is able to generate realistic logs based on a virtual smart space consisting of agents that behave "as if" they were real inhabitants of the actual smart space.

In this paper we propose a method for generating synthetic datasets for experimentation by *simulating* (virtual) smart spaces. As an example application of such a dataset, we will focus on the problem of recognition of *activities of daily life* (ADLs) [3, 16] (or more simply *habits*). Habit recognition can be performed over *logs* resulting from a set of installed sensors (door state detectors, light switches, movement sensors, item presence sensors, etc.). Given a set of ADL models (previously discovered, either automatically or manually) and the log resulting from the installed sensors, a habit recognition algorithm is able to recognize the ADLs users are performing into the environment. Nonetheless, as we explained earlier, only a limited number of such datasets are available.

To this aim, our tool starts from a set of *habit templates* and periodically assigns some of them to be performed by a given agent. From this assignment a random *trace* is generated as a list of actions to be performed by the agent, possibly interleaving the different habits he is pursuing. These are essentially high-level actions, called h-*actions*, each of which corresponds to a goal to be realized according to a lower-level underlying action theory. In the next step, a planner is employed that produces a sequence of low-level atomic p-*actions* for each h-action in the trace, generating a more detailed *planner log*. All these actions are finally executed by a virtual agent into a 3D environment populated with a customizable set of sensors that trigger while the agent is acting. In such a way, from the planner log it is possible to produce a *sensor log*, which associates to each p-*action* a set of sensor measurements. The proposed tool can be customized in order to produce logs of a user-defined length.

Preliminaries

A smart environment (e.g., a smart house) produces a *log* containing a sequence of events generated by sensors and

[1] http://ailab.wsu.edu/casas/
[2] http://architecture.mit.edu/house_n/
[3] http://www.openlivinglabs.eu/

Linear Temporal Logic - LTL

Temporal logics are a special class of modal logics where modalities are interpreted as temporal operators, used to describe and reason about how the truth values of assertions vary over time.

In this class, LTL can be considered as being: *(i)* propositional, as formulae are defined from atomic propositions, whose truth values change over time; *(ii)* linear, as temporal operators predicate on the truth of propositions along a single timeline and each moment has a single next future moment; *(iii)* qualitative, as temporal operators are used to express qualitative time relations between propositions; *(iv)* point-based, as temporal operators and propositions are evaluated over points in time; *(v)* time discrete, as the next moment corresponds to the immediate successor state induced by the occurrence of an event; *(vi)* future-tense, as temporal operators predicate on the occurrence of events in the future.

LTL formulae are defined using atomic propositions (with true and false constants), propositional connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$), and temporal operators ($\bigcirc$ next time, $\Box$ globally, $\Diamond$ eventually, $\mathcal{U}$ until).

represented as n-uple $\langle ts, src, val \rangle$ where for each event we know the source sensor (src), the value associated to the event (val) and the execution timestamp (ts). These are either directly or indirectly influenced by people executing habits into the environment.

People's interaction is typically studied in terms of *habits* or *activities of daily life (ADLs)*. A habit is essentially a loosely specified sequence of high-level actions that are related to each other and aim at pursuing a goal (e.g. cleaning the house). The way a habit is performed may portray a high degree of variability between different users or even between the same user in different time frames. This characteristic of habits makes them suitable to be described using *declarative* process modeling formalisms (such as DECLARE [14]) rather than *procedural* ones.

In order to generate a dataset that represents habit realizations by people one cannot relying solely on the enactment of a habit model, as a habit does not provide information about how the tasks are actually realized. Habit templates essentially refer to high-level events that may be executed in different ways depending on the circumstances. As far as low-level execution is concerned, an action theory can be used as a fine-grain representation of the environment where tasks composing habits may be pursued using planning.

DECLARE and LTL

Instead of rigidly defining the flow of interaction, DECLARE [14] focuses on the (minimal) set of rules which must be satisfied in order to correctly carry out a process (flow of tasks). A DECLARE model is defined as a couple $\mathcal{CM} = \langle T, \mathcal{C}_m \rangle$, where *(i)* T is a set of tasks, represented as boxes containing their name, and *(ii)* $\mathcal{C}_m$ is a set of mandatory constraints.

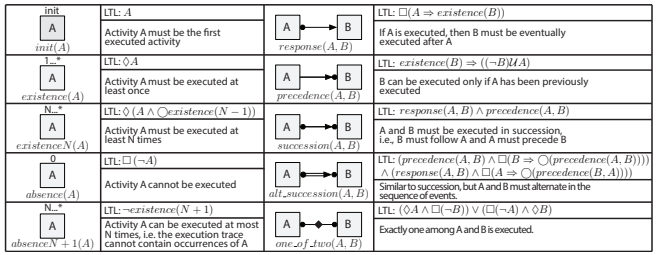

Figure 1: A selection of DECLARE constraints

DECLARE constraints, represented using arrows which connect task boxes and annotations, are grouped into four families: *(i)* existence constraints are unary cardinality constraints expressing how many times an activity can/should be executed; *(ii)* choice constraints are n-ary constraints expressing the necessity to execute some activities between a set of possible choices, independently from the rest of the model; *(iii)* relation constraints are binary constraints which impose the presence of a certain activity when some other activity is performed, possibly imposing also temporal constraints on such a presence; *(iv)* negation constraints are the negative version of relation constraints, and are employed to explicitly forbid the execution of a certain activity when some other activity is performed.

The semantics of a DECLARE model is obtained by translating each constraints into LTL (Linear Temporal Logic) formulas. Figure 1 shows all the graphical DECLARE elements which are used throughout this paper together with their correspondent LTL formulas. We can add to the DECLARE language new constraints, as well as we can associate a graphical symbol to it, and we can also provide an LTL semantics for it. Starting from the single constraints of the model, the LTL formula representing

STRIPS

In STRIPS, the representation of the initial state, the actions, and the goal condition is based on first-order logic literals.

The initial state is specified as a set of positive ground literals which give a complete specification of the world based on a closed-world assumption.

An action is specified in term of its preconditions and effects, also expressed as sets of literals. In the case of preconditions a set of positive literals specify what conditions need to be true in order for the action to be executable. Similarly for the effects of an action, a set of positive and negative literals specify how the state should be transformed when the action is performed: positive literals in the set of effects are added in the set describing the state and negative literals are removed.

A goal condition is also a set of positive ground literals and it is satisfied in a state if all the literals listed in the goal condition are included in the set that describes the state.

the DECLARE model is obtained by the logical AND of the formulas associated to the single constraints.

STRIPS Planning
In STRIPS planning [7] one is faced with the following task. Given *(i)* a complete specification of the initial state of the world, *(ii)* a set of actions that describe how the world may change, and *(iii)* a goal condition, one has to find a sequence of actions such that if they are executed sequentially starting from the initial state, checking for corresponding preconditions, and applying the effects of each action one after the other, they lead to a state that satisfies the goal condition.

In this paper we focus on a STRIPS planning following the Planning Domain Definition Language (PDDL) [8]. In PDDL, the specification of a planning task (planning model) is separated into two parts, the *domain description* that lists the available predicates and actions, and the *problem description* that specifies the initial and goal state.

Dataset Generation Strategy

Smart houses typically generate datasets as sequences of measurements generated by installed sensors. The main disadvantage of this is that gathering a critical mass of data requires a big amount of time; additionally the dataset cannot be augmented adding new sensors without a physical installation and a new recording session. Generating synthetic data on the other hand, speeds up this process but it requires the existence of a simulation environment, reproducing a realistic one at a certain degree of fidelity.

We propose a layered dataset generation strategy which employs techniques from business process modeling and from classical planning in order to drive characters to perform realistic sequences of operations inside a virtual environment and collect synthetic sensor measurements.

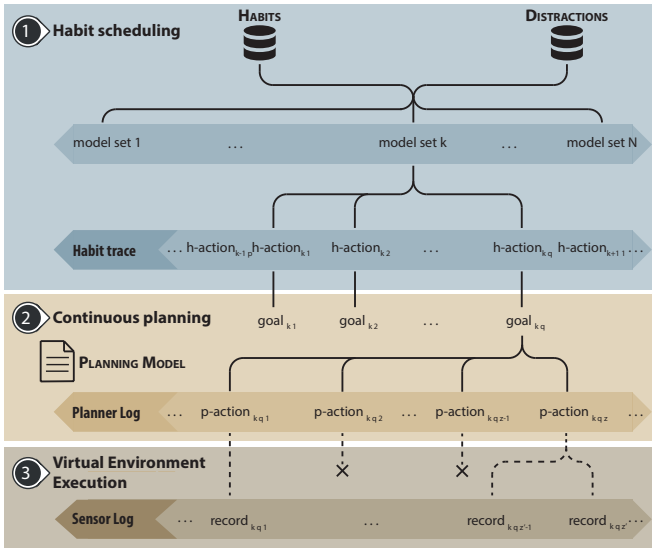

Figure 2: The sensor log generation strategy.

Figure 2 summarizes the strategy employed for dataset generation. Our tool takes as input two repositories describing habit and distraction templates in the DECLARE formalism, and a PDDL file that models the smart home as a planning domain. A designer, through an appropriate interface (textual or graphical) is in charge of defining them. Habit templates relate to achieving an articulate goal in the smart space, e.g. cook dinner, while distraction templates model casual behaviors, e.g., turn-on TV. The planning domain provides lower level actions that may be used to realize the high-level actions of the DECLARE templates.

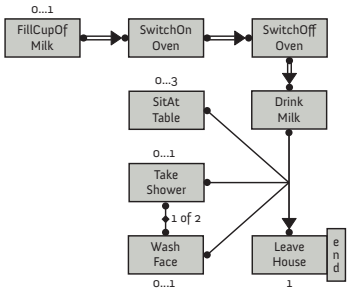

Figure 3: `MorningRoutine`

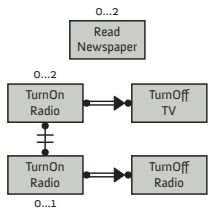

Figure 4: `NewsOfTheWorld`

Figure 5: `TalkingOnThePhone`

In the first stage an *interleaved habit trace* is produced. At each step, a number of habit and distraction templates are chosen, and a random instance of their interleaving is extracted. The number of habits and distractions to be combined in each iteration as well as the number of repetitions (that influence the length of the generated dataset) can be set as parameters of our tool. The resulting log specifies an interleaved sequence of high-level actions that we call h-*actions* based on the given templates.

In the second stage, our tool takes as additional input a PDDL planning model that specifies the virtual environment, and produces a *planner log* in the following way. For each h-action in the interleaved habit trace, a planning problem is instantiated and solved, having as a goal to realize the h-action. The solution generates a sequence of atomic p-*actions*, which are executed updating the planning environment.

In the final stage all the p-*actions* contained in the planner log are executed into a 3D virtual simulation environment. This environment is populated with a customizable set of sensors that monitor the environment. Each p-*actions* may trigger (none or multiple) sensor measurements in the sensor log.

Habit Modeling and Scheduling

Figures 3,4 and 5 depict DECLARE templates describing two daily habits and one distraction. For example, the `MorningRoutine` habit template specifies that the agent either takes a shower or wash his face (modeled using a branching `succession` and a `not_coexistence` DECLARE constraints).

DECLARE templates representing habits and distractions impose constraints about the execution order of composing h-actions. Each DECLARE template can be translated into an LTL formula by just conjuncting the LTL formulas corresponding to each single constraint. As a consequence, it is also possible to combine different templates through a simple logical conjunction, allowing for the execution of more habits/distractions in an interleaved way.

At each step, our tool selects a configurable number of habit and distraction templates following a probability function. This function associates to each habit template the probability of being executed at a certain time of the day. Starting from this set, a random trace is generated, with an approach similar to the one adopted in [5]. In essence, the trace generation is performed by our tool by first translating the LTL formula into a regular expression [6] and then by randomly executing the finite state automata corresponding to the regular expression over the subset of h-actions (considered as an alphabet) contained in the original DECLARE template. If no traces can be generated we can infer that habits and/or distractions are incompatible and a new set of templates is randomly selected.

As an example, an execution trace compliant to the combination of the templates previously introduced is the following:

```
(1) LeaveBed (2) FillCupOfMilk (3) TurnOnRadio (4) ReadNewspaper
(5) TurnOffRadio (6) TalkOnThePhone (7) SittingKitchen
(8) StartOven (9) ReadNewspaper (10) StopOven (11) DrinkMilk
(12) TakeShower (13) LeaveHouse
```

This sequence of h-actions represents a high level strategy employed by the virtual agent to execute his habits.

Continuous Planning

Each of the h-actions in the habit trace is treated as a goal that the agent needs to achieve in the context of a planning model reproducing a smart home.

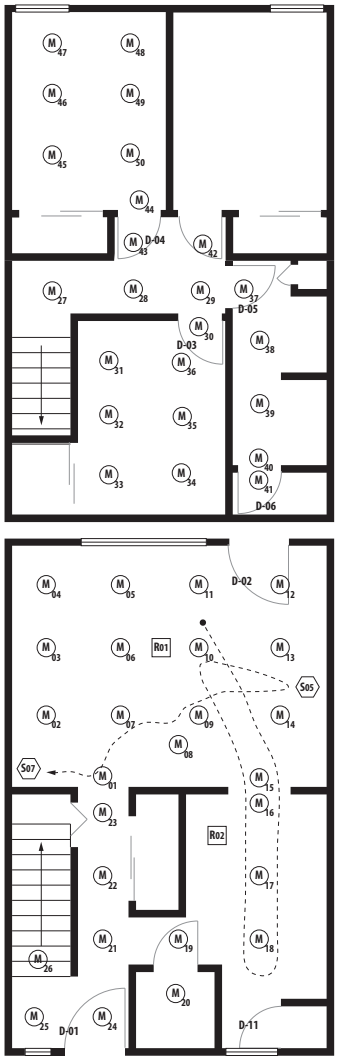

Figure 6: The virtual smart space with deployed sensors.

The PDDL domain specifies four different types of objects: `room`, `device`, `state`. The `device` type abstracts objects that the agent can interact with in the smart home such as *doors*, *windows*, electronic devices and household objects. Some devices are *stateful*, having a current `state` object associated (e.g., either `closed` or `open` for a door device) stored into the `deviceState` predicate.

The arrangement of the smart home is fixed and specified using the predicates `(adjacent ?r1 - room ?r2 - room ?d - device)` and `(deviceAt ?d - device ?r - room)` with the intuitive meaning. Devices used in the context of the `adjacent` predicate as passages between rooms are specified by a `waypoint` predicate. The topology used in our examples and during the evaluation, depicted in Figure 6, is inspired by the Kyoto setup of CASAS project; additionally the planning problem defines a wide set of devices that, for the sake of readability, are not shown.

The basic predicates affected by actions are the following:

- `(characterAt ?r - room ?d - device)` specifies the device the agent is next to;
- `(deviceState ?d - device ?st - state)` specifies, for each stateful device, the current state;
- `(usedDevice ?d - device)` is intended to denote that a device has been used by the agent in the scope of a h-*action* execution.

The available actions model how the agent moves in the environment (`moveToRoom` and `moveToDevice`) and how it interacts with devices (`changeDeviceState` and `useDevice`). The specification of the `moveToRoom` follows:

```
(:action moveToRoom
:parameters (?r1 - room ?d1 - device ?r2 - room ?w - device)
:precondition (and (characterAt ?r1 ?d1) (waypoint ?w)
 (adjacent ?r1 ?r2 ?w) (deviceState ?w open)
:effect (and (not (characterAt ?r1 ?d1)) (characterAt ?r2 null)))
```

Similarly, `moveToDevice` allows the character to move from one device to another belonging to the same room.

The interaction with devices can be performed either by changing the state of a device by means of a `changeDeviceState` action, or by a `useDevice` action that represents the execution of an atomic complex task over the device. The effect of the latter is to change the `usedDevice` predicate, which is reset after each planner execution to allow reuse of devices.

From h-actions to planning problems and p-actions. In order to produce a planner log as part of our data generation methodology, our tool continuously tries to realize new goals following the h-actions in the interleaved habit tracer. For each h-action, a new planning problem is instantiated and solved. The resulting action sequence is then reported and *executed* in the domain so that the current state is *updated* according to it.

The goals for the planning problems need to be specified using predicates available in the planning domain, so a correspondence between h-actions and goals is required. An extended description of the DECLARE input is assumed by our tool, in which for each h-action one or more goal conditions are also specified using the predicates of the PDDL domain. For example the template in Figure 3 is annotated to specify that h-action `LeaveHouse` may be realized by pursuing the goal `(and (deviceState entranceDoor closed) (characterAt outside null))`.

The level of detail by which an h-action will be transformed into a goal and eventually to a sequence of p-actions, depends on the degree of detail of the virtual environment expressed as a planning domain. Note also that the PDDL representation provides a principled way for specifying a wide range of different configurations and

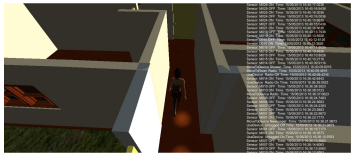

Figure 7: Movement Sensors.

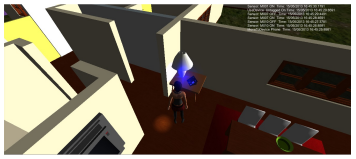

Figure 8: Phone device.

Figure 9: Radio device.

allows for automated methods for re-configuring and exploring the space of design options for smart spaces.

Example.
Suppose that the character, currently in the living room close to the newspaper, wants to fill a cup of milk; the planner may produce the following plan:

```
GOAL deviceState cupOfMilk filled
(1) (moveToDevice livingroom newspaper kitchenDoor)
(2) (changeDeviceState livingroom kitchenDoor closed open)
(3) (moveToRoom livingroom kitchenDoor kitchen kitchenDoor)
(4) (moveToDevice kitchen null cupOfMilk)
(5) (changeDeviceState kitchen cupOfMilk empty filled)
```

After executing the previous plan, the state of the environment is changed: the character is not in the living room anymore, the door of the kitchen is open and the cup is full of milk. Suppose now the character after turning on the radio, answers the phone. The planner will start from the current state of the world and will compute:

```
GOAL deviceState radio on
(1) (moveToRoom kitchen cupOfMilk livingroom kitchenDoor)
(2) (moveToDevice livingroom null radio)
(3) (changeDeviceState livingroom radio off on)
GOAL usedDevice livingroom phone
(1) (moveToDevice livingroom radio phone)
(2) (useDevice livingroom phone)
```

The continuous execution of the planner produces a planner log which contains p-actions. Post-processing removes all the p-actions that do not come from sensors.

Virtual Environment Execution

The planning environment introduced in the previous section can be easily reproduced into a rapid development environment for video-games such as Unity[4]. As a consequence, the *p*-actions contained into the planning log can be executed by a *non-player* character (NPC), thus obtaining a 3D simulation environment.

Beside all the devices defined into the planning problem, the virtual 3D environment contains a customizable set of *sensors* monitoring the house and providing measurements. Each sensor is in charge of detecting and reporting the timestamp of a particular event. Figure 6 shows the set of sensors employed during our tests. As an example, PIR - Presence InfraRed sensors are frequently deployed into real smart houses; they trigger whenever a person is inside their detection cone and can be easily modeled into Unity as colliders. If these kind of sensors are mounted on the ceiling following a grid layout, the triggering sequence can capture the trajectory a person is following into the environment.

Sensors in our Unity virtual environment are completely decoupled from devices and rooms coming from the planning problem. Sets of sensors differing for number and types, as further discussed in the following, give different insights about ADLs performed into the environment. As a consequence, none or more sensor log entries may correspond to a single p-*action*. The virtual 3D environment allows to obtain customized datasets by adding and removing sensors. Additionally, sensors can be customized aiming at representing the real world at the desired level, for example by integrating physics and introducing white noise.

Moreover, the virtual environment allows us to leave timing issues outside the habit and planning layer. The time that executing a p-*action* takes to be executed depends on the way the agent and the devices are modeled into the virtual environment.

Evaluation

In order to illustrate how different sensor configurations influence the obtained dataset, we first look into two sensor logs generated using our tool. Both logs are

[4] http://unity3d.com

```
20-05-13 08:06:24:5232 M010 ON
20-05-13 08:06:25:1766 M010 OFF
20-05-13 08:06:26:5278 M015 ON
20-05-13 08:06:30:5666 M015 OFF
20-05-13 08:06:30:6832 M016 ON
20-05-13 08:06:31:2484 M016 OFF
20-05-13 08:06:46:6807 M017 ON
20-05-13 08:06:47:2657 M017 OFF
20-05-13 08:06:47:6974 M018 ON
20-05-13 08:06:48:1249 M018 OFF
20-05-13 08:06:48:3299 M018 ON
20-05-13 08:06:52:0331 M018 OFF
20-05-13 08:06:52:5234 M017 ON
20-05-13 08:06:53:0781 M017 OFF
20-05-13 08:06:53:5392 M016 ON
20-05-13 08:06:54:0743 M016 OFF
20-05-13 08:06:54:2236 M015 ON
20-05-13 08:06:54:7582 M015 OFF
20-05-13 08:06:56:1208 M010 ON
20-05-13 08:06:56:6013 M010 OFF
20-05-13 08:06:57:9044 M009 ON
20-05-13 08:06:58:5216 M009 OFF
20-05-13 08:06:59:3668 M007 ON
20-05-13 08:06:59:7478 M007 OFF
20-05-13 08:07:00:4976 M001 ON
20-05-13 08:08:32:1452 M001 OFF
```

Figure 10: Sensor log Example 1

```
20-05-13 08:06:25:4872 R001 OFF
20-05-13 08:06:30:4773 R002 ON
20-05-13 08:06:47:5623 R002 OFF
20-05-13 08:06:52:2981 R002 OFF
20-05-13 08:06:54:8979 R001 ON
20-05-13 08:06:57:1171 S005 ON
20-05-13 08:07:00:3328 S007 ON
20-05-13 08:08:32:3230 S007 OFF
```

Figure 11: Sensor log Example 2

generated starting from the very same planner log (shown in the previous example) but using different set of sensors.

The first log (see Figure 10) is obtained by placing into the virtual environment a grid of fine grain movement sensors (e.g., M001, M010) that trigger while the virtual agent is moving following a trajectory (shown in Figure 6). The second log (see Figure 11) is obtained by equipping the environment *(i)* presence sensors in every room (e.g., R001, R002) and *(ii)* switch sensors attached to both the radio and the phone (S005 for the radio and S007 for the phone). It is easy to see from these examples that different sensor configurations provide a completely different point of view w.r.t. habit recognition.

The use of the same habit templates and planning domain over different sets of sensors, enables the evaluation of techniques for smart homes under different assumptions. In order to illustrate the type of analysis, we use a number of datasets generated with our tool as input to the state-of-the-art habit recognition (HR) tool developed by the CASAS project [4] (freely available at the project website). CASAS tool allows to perform activity recognition using various techniques. In our analysis we used a Naive Bayes Classifier (similar results are obtained by using the other provided methods). Also, the DECLARE templates used consisted of 10 habits and 5 distractions inspired by the work of [10] in ADLs.

In order to perform a comparison over sensor configurations, we kept the planner log fixed and varied the type of sensors deployed in the smart home through the planning environment. The configurations were obtained by generating all combinations of classes of generic sensors, denoted as follows: P - Presence InfraRed (PIRs), D - door, L - light, S - shutter and W - window. A small subset of specific sensors to appliances, such as the heating control, radio, TV, vacuum cleaner and oven, were included in all the configurations.

Figure 12 shows the performance of the the HR tool for each configuration (represented w.r.t. black checkboxes). The number reported for each category is the average recognition rate over all habits with respect to the ground truth that is provided by our tool:

- Red bars in the histogram refer to configurations without PIR sensors while blue bars refer to those including PIR sensors. By comparing the two categories we observe that PIR sensors definitely improve the performance of habit recognition producing more accurate results (minimum 71% of accuracy) and a consistently better performance for all categories that include them.
- It is clear that the presence of window sensors always improves performance.
- The last four bars lead to the best accuracy rates reached with the simultaneous presence of PIRs, door and window sensors, with the P/D/W configuration being a minimal set of sensors that achieves best results.

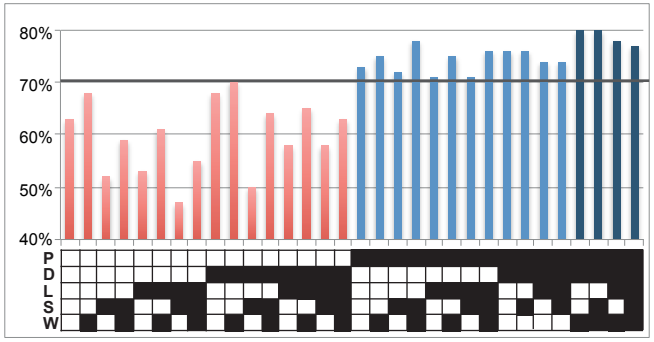

Figure 12: CASAS activity recognition performance.

Concluding Remarks

Generic tools for generating datasets of agents moving into pervasive environments are presented in [9] and [12]. The former tool generates a dataset based on a transition matrix representing the probability-causal relations between all the sensors; as a consequence, the generated sensor log is tightly coupled to the model. The latter tool (based on Repast Symphony simulation environment[5]) aims at the massive generation of big datasets, but no evaluation over whatsoever technique for smart spaces has been – to the best of our knowledge – presented so far.

A customized dataset generation strategy is presented in [11], which is employed to evaluate and optimize performance of a multimodal identification and tracking algorithm.

Recently the UbiREAL[6] [13] prototype has been released (end of September 2012); this is conceived as a framework for building virtual smart spaces, but it does not provide any way to generate datasets. Another example is DIVAS[7] [1], a plugin-based multi-agent 3D simulation framework focusing on large scale events.

Currently, the approach we described is used to generate sensing data related to a single character that moves and acts in the smart space. Real smart spaces, however, may feature multiple persons at the same time each one performing his own habits. This adds an additional complication to processes that analyze sensing data, as most of the widely used sensors generate measurements that are not correlated to a particular person at the time of sensing. As a consequence it is important that the system is able to generate a dataset depicting the activities of multiple agents. From the point of view of our methodology this represents a challenge because agents can execute habits concurrently and this has to be modeled in a way that a realistic dataset can be generated.

We supposed so far that virtual non player agents are the only ones inhabiting the 3D virtual environment. Part of our future work is to couple this type of automated behaviors with user-generated behaviors inside the simulated environment. One idea is to provide the environment to real people as a *serious game*. In such a way, the human user can "play" a portion of his daily life (possibly by giving incentives to reproduce his everyday activities or act in hypothetical scenarios) and out tool can record such traces. In turn these traces (appropriately abstracted) are used as possibly more realistic habit templates, to be used as "generation seeds" for more realistic logs. The ultimate aim is to exploit *crowdsourcing* in order to generate large amounts of data that also exhibit a higher degree of variability still being grounded to human activities. Real people may be instructed to perform daily activities while virtual agents driven by our scheduling strategy interact with them executing daily activities of their own. In this sense, a direct comparison should be done with games like *The Sims*: a major difference here lies on the fact that The Sims does not intend to model extremely realistic situations and the agents act following their own fitness function instead of a structured schedule.

A last remark is about the need for a tool as the one presented in this paper. Indeed we aim at investigating novel techniques for the automatic mining of habit templates. In order to validate our proposed techniques (and many other researchers as well have similar needs), we need large datasets, to be used as benchmarks, that

[5] http://repast.sourceforge.net/
[6] http://ubireal.org/
[7] http://mavs.utdallas.edu/projects/divas/

currently cannot be derived by real smart homes. A tool as the one proposed in this paper, especially when grounded to real users through the envisioned gamification approach, can solve similar issues: by using an evocative parallel, as a wind tunnel is essential for engineers in aerospace/automotive, the same is true for smart space makers.

Acknowledgements

This work has been supported by the EU project GreenerBuildings (FP7-258888), by AriSLA through the Italian project Brindisys, and by SAPIENZA Università di Roma and DIAG through the grants SUPER and "Assegni di Ricerca 2012".

References

[1] Araujo, F., Al-Zinati, M., Valente, J., Kuiper, D., and Zalila-Wenkstern, R. Divas 4.0: A framework for the development of situated multi-agent based simulation systems. In *12th Intl. Conf. on Autonomous Agents and Multiagent Systems* (2013).

[2] Ashton, K. That "internet of things" thing. *RFID Journal* (2009).

[3] Atallah, L., and Yang, G.-Z. The use of pervasive sensing for behaviour profiling: a survey. *Pervasive and Mobile Computing 5*, 5 (2009), 447 – 464.

[4] Cook, D. Learning setting-generalized activity models for smart spaces. *IEEE intelligent systems* (2010).

[5] Di Ciccio, C., and Mecella, M. Mining constraints for artful processes. In *BIS* (2012), 11–23.

[6] Diekert, V., and Gastin, P. First-order definable languages. In *Logic and Automata* (2008), 261–306.

[7] Fikes, R., and Nilsson, N. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2*, 3 (1972), 189–208.

[8] Fox, M., and Long, D. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR) 20* (2003), 61–124.

[9] Helal, A., Mendez-Vazquez, A., and Hossain, S. Specification and synthesis of sensory datasets in pervasive spaces. In *IEEE Symposium on Computers and Communications* (2009), 920–925.

[10] Krishnan, N., and Cook, D. Activity recognition on streaming sensor data. *Pervasive and Mobile Computing* (2012).

[11] Menon, V., Jayaraman, B., and Govindaraju, V. Multimodal identification and tracking in smart environments. *Pers. and Ubiquitous Comp.* (2010).

[12] Merico, D., and Bisiani, R. An agent-based data-generation tool for situation-aware systems. In *7th Intl. Conf. on Intelligent Environments* (2011).

[13] Nishikawa, H., Yamamoto, S., Tamai, M., Nishigaki, K., Kitani, T., Shibata, N., Yasumoto, K., and Ito, M. UbiREAL: Realistic Smartspace Simulator for Systematic Testing. In *Proc. 8th Int'l Conf. on Ubiquitous Computing* (2006).

[14] Pesic, M., Schonenberg, H., and van der Aalst, W. Declare: Full support for loosely-structured processes. In *11th IEEE International Enterprise Distributed Object Computing Conference* (2007), 287–287.

[15] Rosenthal, L., and Stanford, V. M. Nist smart space: Pervasive computing initiative. In *Proc. of the 9th IEEE Intl. Workshop on Enabling Technologies: Infrastructure for Collab. Enterprises* (2000), 6–11.

[16] Ye, J., Dobson, S., and McKeever, S. Situation identification techniques in pervasive computing: A review. *Pervasive and Mobile Computing 8*, 1 (2012).