Artifact: SmartSPEC: customizable smart space datasets via event-driven simulations

Andrew Chio, Daokun Jiang, Peeyush Gupta, Georgios Bouloukakis, Roberto Yus, Sharad Mehrotra, Nalini Venkatasubramanian

Artifact: SmartSPEC: Customizable Smart Space Datasets via Event-driven Simulations

Andrew Chio*, Daokun Jiang*, Peeyush Gupta*, Georgios Bouloukakis**, Roberto Yus†, Sharad Mehrotra*, Nalini Venkatasubramanian*
*Dept. of Computer Science, University of California, Irvine, {achio,daokunj,peeyushg,sharad,nalini}@uci.edu
**Dept. of Computer Science, Télécom SudParis, IP Paris, georgios.bouloukakis@telecom-sudparis.eu
†Dept. of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, ryus@umbc.edu

I. Introduction

This artifact abstract is a guideline for SmartSPEC [1], a simulator for generating customizable smart space datasets using semantic models of spaces, people, events and sensors. SmartSPEC is based on two main components: (i) *Scenario Learning* which produces *metamodels* of events and people using input seed data; and (ii) *Scenario Generation* which uses SmartSPEC data to generate a realistic smart space dataset.

SmartSPEC provides three modes of operation to generate synthetic data varying in the level of user involvement/automation (see Fig. 1). The steps to use our system are as follows:

- Define the simulated space and its embedded sensors ❶.
- Define MetaPeople and MetaEvents manually (❷a) or automatically (❷b).
- Define specific people and events based on the previous metamodels manually (❸a) or automatically (❸b).
- Configure simulation/generate the synthetic dataset (❹).

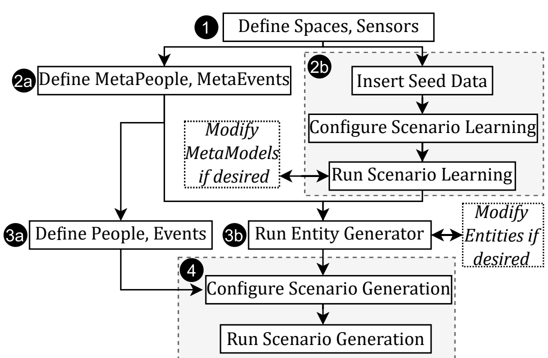

Fig. 1: SmartSPEC workflow.

II. Using SmartSPEC

We describe the steps required to generate a synthetic dataset following the flow ❶ → ❷b → ❸b → ❹ in Fig. 1. For details on other modes of operation and model parameters, please refer to the guide in the SmartSPEC GitHub repository [2].

Installation & Dependencies

The SmartSPEC code and the full list of dependencies are publicly available in [2]. We recommend using an Anaconda environment to run the Scenario Learning component (written with Python 3.8) and using a C++ editor/Linux environment to run the Scenario Generation component (written with C++17).

```
[{"id": 1,
  "description": "lobby",
  "coordinates": [30,50,10],
  "capacity": 30,
  "neighbors": [2,3] }, ...]
```

(a)

```
[{"id": 3,
  "description": "AP-2081"
  "mobility": "static",
  "coverage": [1,3],
  "interval": 60 }, ...]
```

(b)

Fig. 2: Sample definition files: space (a) and sensor (b).

❶ - *Defining Spaces & Sensors*

After installation, we first define *Spaces.json*, which contains the logical representation of the smart space (i.e., rooms, regions). Each element of the file is a JSON object uniquely identified by a nonzero integer `id`[1] that contains: a 3-tuple of XYZ `coordinates` to represent its centroid, a maximum `capacity` (i.e., number of people allowed in space), and a list of `neighbors` (i.e., accessible, adjacent spaces). Fig. 2a shows a definition for the lobby of a smart building.

In addition, we generate a *Sensors.json* file to define sensors deployed in the above space as in Fig. 2b. Each sensor is a JSON object uniquely identified by `id` that contains: its `mobility` ("static" or "mobile"), its `coverage` (i.e., set of spaces it can cover), and observation production `interval`.

We recommend defining *Spaces.json* and *Sensors.json* in a subdirectory of `scenario-learning/data`. In [2] we provide sample space and sensor files in the directory `scenario-learning/data/demo`.

❷b - *Scenario Learning: Generating MetaModels*

Next, we define metamodels for events (i.e., *MetaEvents*) and people (i.e., *MetaPeople*) characterizing types of events/people in the smart space. SmartSPEC can extract such metamodels automatically based on seed connectivity data (e.g., set of WiFi probe requests, ❷b), or manually using the definitions in [2] (i.e., ❷a). For ❷b, we start with seed data as in Fig. 3a and populate a MySQL database as created in Fig. 3b. Each field represents a client device `client_id` that connects to access point `wifi_ap` at time `cnx_time`.

Then, we define a configuration file using the learning parameters in Fig. 4. The list of parameters to specify include: `start` and `end` to denote start/end dates; `unit` to denote intervals (number of minutes) to group elements from the seed data; `validity` to denote time periods (number of minutes) for which a client is assumed

[1]The space with `id=0` represents "outside of simulated space".

```
wifi_ap,cnx_time,client_id
1,2017-01-01 07:30:31,81
9,2017-01-01 10:39:13,72
8,2017-01-01 10:40:08,72
...
```

(a)

```
CREATE TABLE
simulation_seed.connectivity(
  wifi_ap VARCHAR(32) NULL,
  cnx_time DATETIME NULL,
  client_id VARCHAR(64) NULL);
```

(b)

Fig. 3: Seed connectivity (a); Setting up database (b).

to remain near the access point after connecting to it; smooth/window to denote a smoothening function to apply with specified window size; time-thresh to denote the minimum number of minutes to realize an event; and occ-thresh to denote the minimum number of people to realize an event. The paths of the previously defined spaces/sensors should also be listed under the filepaths section. See scenario-learning/data/demo/config.txt in [2] for a full sample configuration file.

```
[learners]
start       = 2017-04-01
end         = 2017-05-01
unit        = 5
validity    = 10
smooth      = EMA
window      = 10
time-thresh = 30
occ-thresh  = 1

[filepaths]
spaces     = data/demo/Spaces.json
sensors    = data/demo/Sensors.json
metaevents = data/demo/MetaEvents.json
metapeople = data/demo/MetaPeople.json
...
```

Fig. 4: Sample Scenario Learning configuration file.

To execute the Scenario Learning component, run python main.py <config> from the scenario-learning directory, where <config> is the configuration file path. This step produces *MetaEvents.json* and *MetaPeople.json* in the provided path, which serves as input for Scenario Generation. Note that these files can be modified if desired. For the Scenario Learning component, we provide mock connectivity data and a corresponding configuration file. However, this mock data is randomly generated, resulting in randomly learned metamodels. Under normal operation, the user should copy the generated *MetaEvents.json* and *MetaPeople.json* into a desired subdirectory of scenario-generation/data. For demonstration purposes, we provide a separate set of metamodels in [2] for the Scenario Generation component, located at scenario-generation/data/demo.

3b - *Scenario Generation: Generating Entities*

Using the generated metamodels from the previous step, we initialize a set of events and people to use in the Scenario Generation component. Similar to the generation of metamodels, this can be done automatically by running the Entity Generator module (i.e., 3b), or manually using the definitions provided in [2] (i.e., step 3a). A configuration file such as the one in Fig. 5 is needed for the entity generator. Here, the important parameters include: number and generation for each of the sections people and events; number refers to the number of entities to simulate and generation refers to the manner in which new entities (if any) should be added. For example, for people, if generation=none, then number is ignored and the people specified in filepaths/people will be used. If generation=diff, then one of each metaperson will first be generated (up to number), then additional people will be added (up to number). If generation=all, then number people will be generated using metapeople.

The entity generator should be compiled with make entitygen and run with entitygen <config>, where <config> is a scenario generation configuration file. See scenario-generation/data/demo/config.txt in [2] for a full sample configuration file.

```
[people]
number = 500
generation = all

[events]
number = 5000
generation = diff

[synthetic-data-generator]
start = 2018-01-08
end   = 2018-01-29

[filepaths]
metapeople  = data/demo/MetaPeople.json
metaevents  = data/demo/MetaEvents.json
spaces      = data/demo/Spaces.json
sensors     = data/demo/Sensors.json
people      = data/demo/People.json
events      = data/demo/Events.json
output      = data/demo/output/
...
```

Fig. 5: Sample Scenario Generation configuration file.

4 - *Scenario Generation: Generating Synthetic Data*

After generating people/event files, the synthetic data generator produces a smart space dataset. To run this module, compile it with make datagen and run with datagen <config>, where <config> is a scenario generation configuration file as specified in Fig. 5. We note the start and end options in the synthetic-data-generator section, which denote the start and end dates of the simulation. The output of this step includes two files in the output directory. First, output/trajectory.csv contains the semantic trajectories of individuals (i.e., semantic location at a given point of time). Second, output/observations.csv contains sensor observations which represent that a specific sensor "observed" phenomena caused/influenced by the presence of a person in its coverage area.

III. Conclusion

The realistic synthetic smart space dataset generated by SmartSPEC can be used for different tasks including, but not limited to, testing and validating approaches for sensor placement, location-based technologies, and sensor data management.

References

[1] A. Chio, D. Jiang, P. Gupta, G. Bouloukakis, R. Yus, S. Mehrotra, and N. Venkatasubramanian, "Smartspec: Customizable smart space datasets via event-driven simulations," in *20th Int. Conf. on Pervasive Computing and Communications (PerCom)*, 2022.

[2] "SmartSPEC," https://github.com/andrewgchio/SmartSPEC, 2022.